Moment under a point load in Thin Shell Structures

Optimization of design equation

Karan Taneja

Technische Universiteit Delft

TU Delft Delft University of Technology

Challenge the future

MOMENT UNDER A POINT LOAD IN THIN SHELL STRUCTURES

OPTIMIZATION OF DESIGN EQUATION

by

Karan Taneja

as an additional graduation work (CIE5050-09) for partial fulfilment of

Master of Science
in Structural Engineering

at the Delft University of Technology.

Student number:	4505468	
Supervisors:	Dr. ir. P.C.J. Hoogenboom,	TU Delft
	Dr. ir. drs. C.R. Braam,	TU Delft

TU Delft Delft University of Technology

PREFACE

This report was made as an Additional thesis for the course CIE-5050-09. The inspiration to work on this topic came through a discussion with Dr. Pierre Hoogenboom, during the course *Shell Analysis, Theory and Application* (CIE-4143) which I undertook. I would like to thank him for his unflinching support and guidance throughout the duration of the work.

I would also like to thank my colleagues at TU Delft and my friends and family back in India for their exuberant motivation during the course of the project and report.

Karan Taneja
Delft, November 2016

SUMMARY

- The objective of this additional thesis was to estimate and optimize the design equation for moments in x-direction under a point load in a shell structure. For that, the Finite Element package ANSYS Mechanical was used to obtain the moment results and the method of fitting a function to data was used to estimate a formula that produced results within a tolerable error percentage. Based on a few critical assumptions about the effect of the material properties, the structural characteristics and parametric studies, the formula found in [1] was the best estimation that could be obtained.
- To begin with, in chapter 1, the method used to run ANSYS and the implementation of the script used to create the finite element model has been described. The important parts of the scripts have been explained.
- An important part of the study was to find out the ideal parameters of the finite element model that could be used to study the variation of moments due to change in curvature of the shell. Hence parametric studies (**chapter 3**) were done for a reference shape, where the lengths in x and y, the number of elements along x and y axes, the parameters used for the distribution of the point load and the thickness of the structure were varied and studied.
- In the final and the most critical part of the project, it was assumed that the formula to calculate the moment was dependent on **three variables**, namely $k_{xx} \cdot t$, $k_{yy} \cdot t$ and the $\frac{d}{t}$ ratio (where k_{xx} and k_{yy} are the curvatures in x and y directions at the origin of the shell structure, **t** is the thickness of the shell and **d** is the diameter of point load distribution area) and that assumption proved to be true as equation **6.1** in **chapter 6** of this report validated the claim.

CONTENTS

1

INTRODUCTION

Shell structures are designed to carry loads via membrane forces but even then moments occur at edge disturbances which can be anything from point loads to change in boundary conditions or anything that leads to a sudden change in the flow of forces through the structure. Through previous works, we know analytically that the effects of these edge disturbances are local in nature and they decrease to negligible proportions of their peak values when measured at and beyond the influence length.

Previously, solutions have been explored analytically and numerically leading to design equations for estimating moments under a point load for a shell structure. The methods to solve this problem included solving partial differential equations [2] and usage of Bessel functions to simplify the results [3],[4]. Some research led to equations which could represent the moments as a series that could be numerically evaluated [5] as well as presenting curves for the same [6], [7]. [8] found a relatively simpler equation for moments but under quite a few assumptions.

In this report (parts of which have been used in [1]), the data generated via computational analysis was collected and curve fitting was tried on it. In other words, after observing the pattern formed by moment results obtained from ANSYS, a formula was estimated and fitting to the moment data obtained as previously mentioned, was conducted.

2

THE ANSYS SCRIPTS

To understand how the moments are calculated in ANSYS for this study, it was crucial to know how the APDL Scripts used work.

2.1. SHELL1.MAC

The script can be found in **Annex A.1**. Some important parts of it are discussed below. The purpose of this script was to understand the basic APDL commands.

1. **Insertion of nodes and Formation of Elements.**
   This had to be in accordance with the shape which was governed by the curvature value provided. In this script, only the twist curvature k_{xy} was provided and the surface generated would be a ***hyperbolic paraboloid***, the equation of which is provided below[9]:

   $$z = x \cdot y \cdot k_{xy} \tag{2.1}$$

   The x and y co-ordinates were obtained using the conditions set in the nested loop and the z-coordinate was calculated using the above equation.

   Next, elements were formed between these nodes and Element type **SHELL281** from the ANSYS Element Library was used [10].

2. **Boundary Conditions.**
   As only a *quarter of the model was created,* additional supports were added to account for *symmetry along the x and y axes.*

   At the edges of the quarter model, Moments in all directions were released and the translations in the model were fixed in global X, Y and Z directions. This can be seen in figure 2.1.

3. **Application of Load.**
   In this script, a pressure load was converted to Point Loads in *positive z-direction* which were applied to the corner nodes of all the elements **except** node at the origin and the nodes at the edge $y = \frac{l}{2}$. This can be seen in figure 2.2.

4. **Moment Results.**
   As moment results at the middle of the structure are required, only stresses at **node 1** were obtained

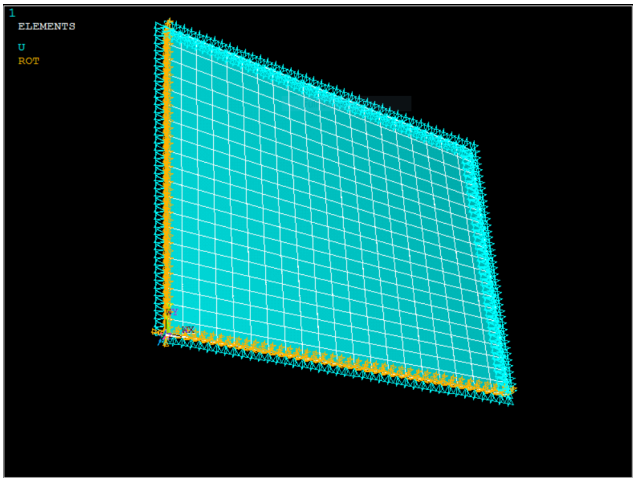


Figure 2.1: Boundary Conditions for ***Shell1.mac***. The *Symmetric* boundary conditions can be differentiated from the *Model* boundary conditions due to the presence of rotational constraints.

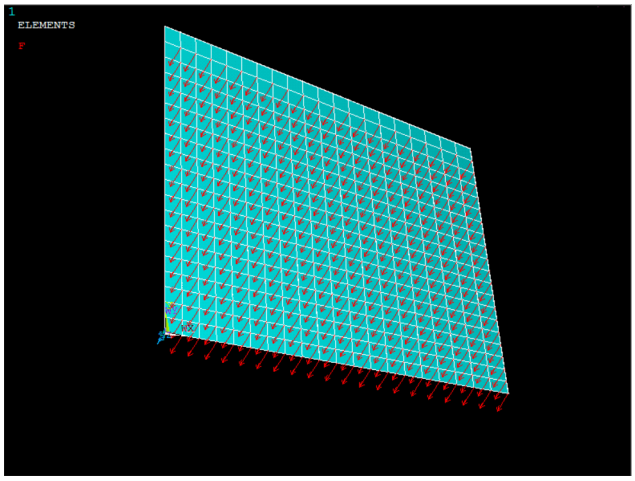


Figure 2.2: Applied load for **Shell1.mac**.

from the post-processing data of ANSYS[11]. Using **Euler's bending theory**, the moment at node 1 was derived by *first calculating the moment caused by stresses in the **bottom layer** and then subtracting the moment caused by stresses in the **top layer***. This was done keeping in mind the sign convention as z being positive in the downwards direction and the x and y axes being in conjunction with the '*right hand thumb rule*'. A strip of half a unit width (*due to symmetry, the other half was assumed to lie in the part of the structure which isn't modelled*) and height equal to the '*thickness*' used in the model was assumed to calculate the moment at Node 1. While figure 2.5 describes the situation for the second script, Shell24.mac, the logic used was the same.

2.2. SHELL24.MAC

This script was used to run the ANSYS simulations in batch mode to *collect the data on which further analysis was done*. The important functions of the script are discussed below. This script is presented in Annex B.1.

1. **Insertion of nodes and formation of elements.**
   In the script, parameters gx, gy, mx, my, qx and qy were created specifically with the aim to generate the x and y co-ordinates of the nodes. The z co-ordinate was calculated using the equation of a paraboloid [9], presented below.

$$z = \frac{1}{2} k_{xx} \cdot x^2 + \frac{1}{2} k_{yy} \cdot y^2 \tag{2.2}$$

In this script, the curvatures at the origin, k_{xx} and k_{yy}, were provided and curvature k_{xy} was not considered. The advantage of using curvatures at the origin was that these can be oriented in the direction of the principal curvatures and hence k_{xy} could be ignored in the formation of the surface without losing out on anything. Had curvatures been associated with any other point on the surface, the use of k_{xy} was necessary.

After creation of the nodes, elements of type **SHELL281**[10] were inserted between them.

2. **Boundary Conditions.**
   As only a quarter of the structure was modelled based on the principle of symmetry, appropriate boundary conditions were assigned to the axes of symmetry (2.3).
   The boundary conditions at the edges were assigned to *"carry the normal forces and the in-plane shear forces"*[1]. Hence nodal axes of the nodes at the edges were *rotated* to carry forces in the Global X and Y directions so as to avoid surplus reactions which would make the schematized boundary conditions invalid.

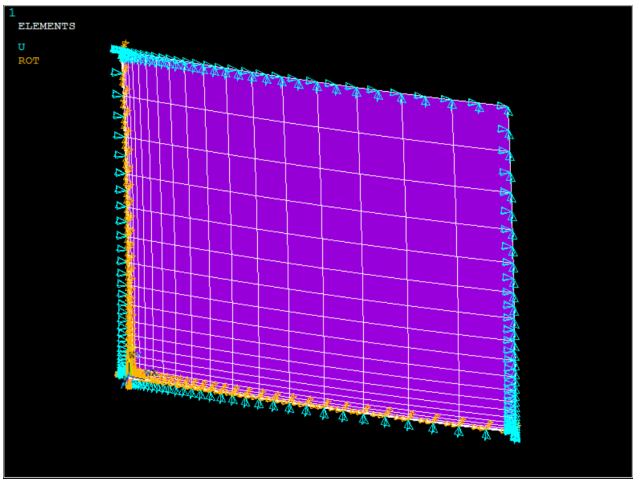


Figure 2.3: Boundary Conditions for ***Shell24.mac***. The *Symmetric* boundary conditions can be differentiated from the *Model* boundary conditions due to the presence of rotational constraints.

3. **Application of load.**
   The **point load** was not simply put at the middle node of the structure because **singularities** or very high results under the point load need to be avoided. It was broken down into many nodal loads which were distributed within circle whose radius was assumed to be a linear function of t. This was the reason that the mesh was finer at the origin and to be computationally efficient, became coarser away from the load. Also, because it was known via previous studies [12] that the effect of such loads was local, it made sense to make a finer mesh near the load and coarser away from it.

   The first part of the code calculates the number of elements along x and y which lie within the radius of influence.
   The load was distributed in those elements as follows:

   (a) Within the radius of influence, the area was divided into 'n' rings where n was calculated based on a designated load patch size which was successively subtracted from the radius of influence until it became zero.

   (b) The ring width was calculated by dividing the radius by '*n*'.

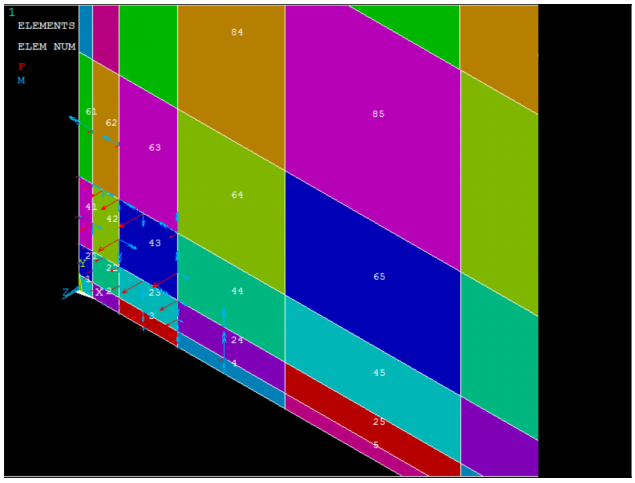


Figure 2.4: Applied Load for ***Shell24.mac***. Node 1 can also be distinguished as the node at the origin.

(c) The *n* rings were then iterated over and for each ring the following steps were under taken:

- Number of sectors within a ring were calculated by subtracting minimum sector size which is the ring width from the circumference of the ring.
- Load within a sector dP, was calculated based on the proportion of the sector area from the whole circle and applied on the Center of Gravity.
- This load was then shifted to the nearest node by finding out the node with the least distance from the Center of Gravity of the sector.
- Moments were also added to the nodes based on the lever arms in x and y directions.

(d) This process was then repeated until all the sectors in all the rings had been iterated through.

4. **Moment Results.**
   They are obtained the same way as in the previous script,**Shell1.mac**. Figure 2.5 offers the analytical model used in calculating the results.

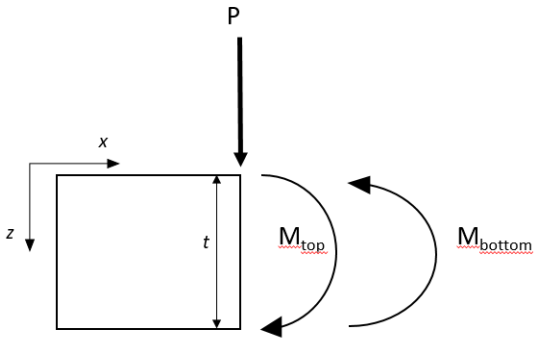


Figure 2.5: Analytical model used for calculation of moment. Here, the view from the xz plane is given. The 'strip' of half a unit width should be imagined to be along the y-axis.

5. **Influence Length.**
   The influence length mathematically may be defined as the distance from one root of the function of an edge disturbance to another or simply the distance from a disturbance at which the influence of the disturbance can be neglected. Based on this logic, the influence length in a direction was calculated as follows:

(a) The moment in the respective direction was chosen from node 1.

(b) The moments in the consecutive edge nodes of the elements were calculated the same way as the moment from node 1.

(c) The consecutive nodes were then iterated over. If the sign remained the same as the moment from node 1, the lengths were added and where the signs changed the last calculated length with all the additions from the previous iterations was chosen and a final increment was added to account for the length required for the moment to go to zero.

The calculated influence length turned out to be 63.2 mm and can be seen in the script in the appendix B.1.

3

PARAMETRIC STUDIES

It was assumed based on the initial conditions that the formula would depend on the thickness of the shell (uniform globally in the model) and the curvatures in x and y directions at the origin. Hence, the second task was to do a study of the other parameters apart from curvatures and thickness, which were namely the element size, length of the shell in both directions and the patch size used in the distribution of the load on a **hypar shell**, which was the reference shape.

The batch mode of ANSYS was used to run multiple analyses one after the other. Python was decided as the medium to generate the .bat files required to run the Batch Mode of ANSYS Mechanical. A Python script would be used to gather the parameters from the requisite excel files, calculate the parameter if need be or else store the variable parameter and write the required code in a .bat file which can then directly run ANSYS without the need to call the program again and again.

The $m_{xx}$and m_{yy} output results of the initial script (Shell24.mac) were compared to the moment results produced by the variation of different parameters. If the moments reached a difference of around 1% with a lower value of the parameter being checked, it would be modified and then used in the final version of the script after deliberation.

3.1. LENGTH IN X AND Y DIRECTIONS

The **first parameter** to be checked was the length in x and y which was calculated by calculating the root of thickness over the curvature of the direction perpendicular to the respective length and multiplying it with *a numerical constant* which was to be decided via the study. The quantities multiplied by the constants are known as L_{0i} in this study with 'i' standing for the direction. These lengths can be visualised in figure 4.1. In the model used in ANSYS, the lengths are divided by 2 because only a quarter of the actual model is created because of the symmetry.

The **reference result** was chosen to be the one obtained from the earlier ANSYS script, Shell24.mac. The numerical constants were varied from 1 to 20 (whole numbers only) in both x and y directions and the moment results were compared.

It was found that from 9 onwards in both directions, the percentage difference was less than 1% and was acceptable. It was decided to use 9 as the numerical constant in both directions and not higher numbers as it would reduce computational time and the error was acceptable.

This **conclusion** can be viewed in figure 3.1. In the figure, L_{0x} is defined as $\sqrt{\frac{t}{abs(k_{yy})}}$ and L_{0y} as $\sqrt{\frac{t}{abs(k_{xx})}}$.

Most of the higher percentage differences can be viewed for lengths shorter than the influence length. As the length increases, the error reduces and goes to zero. It was interesting to note that for m_{xx}, having a

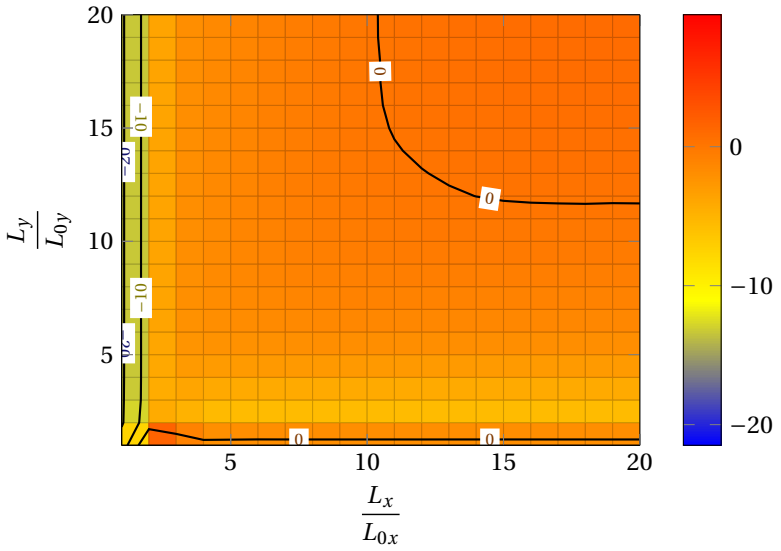


Figure 3.1: Contour plot of % difference in Moment results due to variation of lengths in x and y directions.

higher length in x did give an error of close to 0% but it *led to high errors* when the moments were checked in y-direction as it meant having a length lower than the influence length in y. To have both m_{xx} and m_{yy} within a tolerable % difference, *equal lengths in x and y were used.*

Different coefficients could have been used in both directions but they wouldn't have made a difference as the effect of the point load is local and as along as the total length was higher than the influence length (found to be around 63.2 mm from the output of Shell24.mac), the results would be okay. The chosen length is around 9 times the estimated influence length.

3.2. NUMBER OF ELEMENTS IN X AND Y DIRECTIONS

The **second parameter** was the number of elements in x and y. The reference number was 80 in both directions and *the variation* was from 10 to 80 in multiples of 10. For **lower element numbers** like 10 in x and 10 in y, the results varied highly. That could be due to lesser elements to account for the disturbance due to load. Otherwise, the desired results didn't diverge much.

This becomes evident on viewing figure 3.2. It can be seen that any number of elements beyond 20 in x direction **irrespective** of the number of elements in y would have satisfied our criteria. However, even though the number of elements could be lowered, it wasn't changed to take into account future incorporation of complicated shapes.

3.3. LOAD PATCH SIZE

The third parameter to be checked was the patch size. The point load applied on the hypar in the finite element model was distributed over all nodes inside a certain diameter and the load was applied based on the proximity of the node to the center of the circle in which the load was distributed. This was done to *avoid numerical inaccuracies* that occur by checking stresses directly under a point load.

The initial ratio between the diameter and the element size ($\frac{d}{h}$) in the middle was 20 and taken as the reference and the ratio between the element size and the size of the load patch ($\frac{h}{b}$) which would be used in the application of loads at the nodes was 4.

It can be seen in figure 3.3 where after $\frac{d}{h}$ equal to 4 onwards the percentage difference decreases and at 10 hits **less than** 1%. It also became quite clear that a coarser mesh with the mesh element size being equal

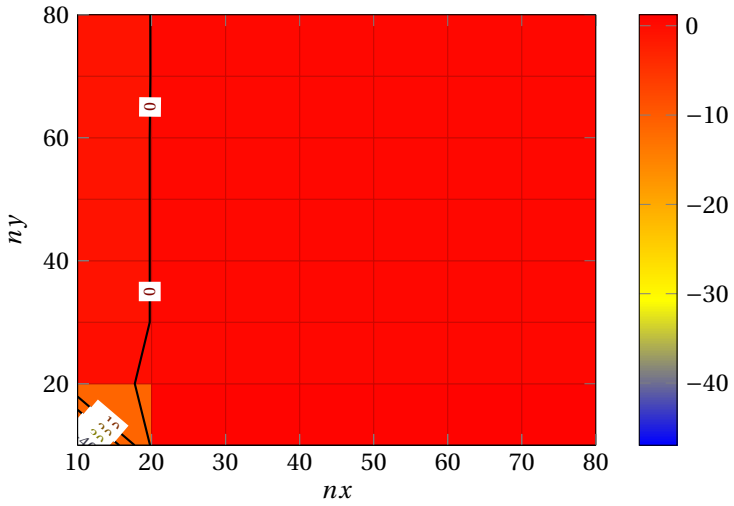


Figure 3.2: Contour plot of % difference in Moment results due to variation of **number of elements** in x and y directions.

to the load patch size would also work (h=b). The results of the study showed that we could go with a coarser mesh with the respective ratios to be 10 and 1 as the percentage change in m_{xx} and m_{yy} was about 0.34%.

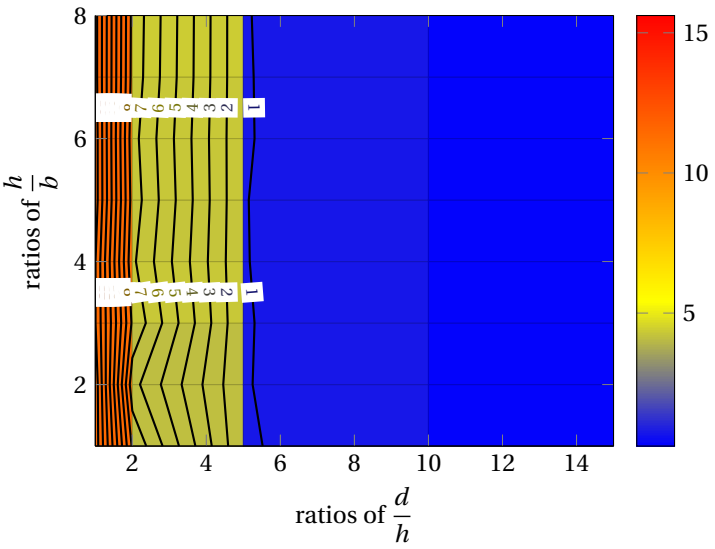


Figure 3.3: Contour plot of % difference in Moment results due to variation of element size in the middle (h) and load patch size (b).

3.4. THICKNESS

The assumption to verify was that moments depended on the $\frac{d}{t}$ (diameter of distribution of point load to Thickness) ratio and not on the thickness itself.
This was verified by checking the results in x and y directions while the following conditions were in place:

1. The $k_{xx} \cdot t$ would remain $\frac{1}{3000}$ and $k_{yy} \cdot t$ would remain $\frac{1}{30}$.

2. The $\frac{d}{t}$ ratio would remain the same.

Keeping these in mind, 3 analyses were conducted with the following parameters:

Table 3.1: Parameters in Shell24.mac for checking effect of thickness.

t (mm)	**d (mm)**	k_{xx} ($\frac{1}{mm}$)	k_{yy} ($\frac{1}{mm}$)
1	5	$\frac{1}{3000}$	$\frac{1}{30}$
2	10	$\frac{1}{6000}$	$\frac{1}{60}$
3	15	$\frac{1}{9000}$	$\frac{1}{60}$

This assumption turned out to be true and **no change** was observed in the m_{xx} and m_{yy} results from ANSYS.

Based on this study, the following changes were made in the script:

1. L_x and L_y were changed to **9 times** L_{0x} and L_{0y} respectively **from 14 times** originally.
2. $\frac{d}{h}$ was changed from 20 to 10 and $\frac{h}{b}$ was changed from 4 to 1.

The number of elements was not altered.

4

MOMENT RESULTS FROM SHELL24.MAC

The third task was to perform simulations of varying 10 values of curvatures in x and in y for each ratio of $\frac{d}{t}$ from 1 to 8 and collect the moment results directly at node 1 which for our theoretical model is directly beneath the point load (refer figure 4.1). Again, Python was used to generate the batch files. The initial parameters were as follows:

Table 4.1: Initial parameters for ANSYS simulations.

Thickness (mm)	**Young's Modulus ($\frac{N}{mm^2}$)**	**Poisson's Ratio**	**Point Load in the middle (N)**
1	10^5	0	100

The curves obtained were symmetrical around the origin, rose smoothly and peaked at the curvature close to zero and then decreased the same way. This was expected because as the structure became flatter, more and more load would be carried by bending moments instead of normal forces as is in the case of shells. There was an empty space observed as curvature equal to zero wasn't analyzed. It was interesting to note that as $\frac{d}{t}$ became higher, the maximum value of the moments became lower. This could be attributed to the fact that as the diameter of influence increased, the area under influence also increased and hence the value of the load applied near the center decreased, leading to a lower moment value.

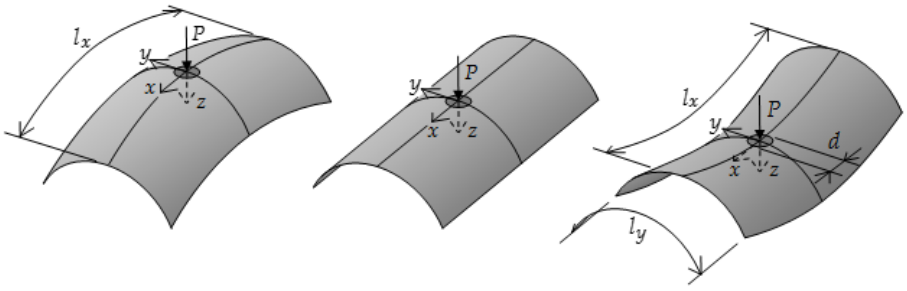
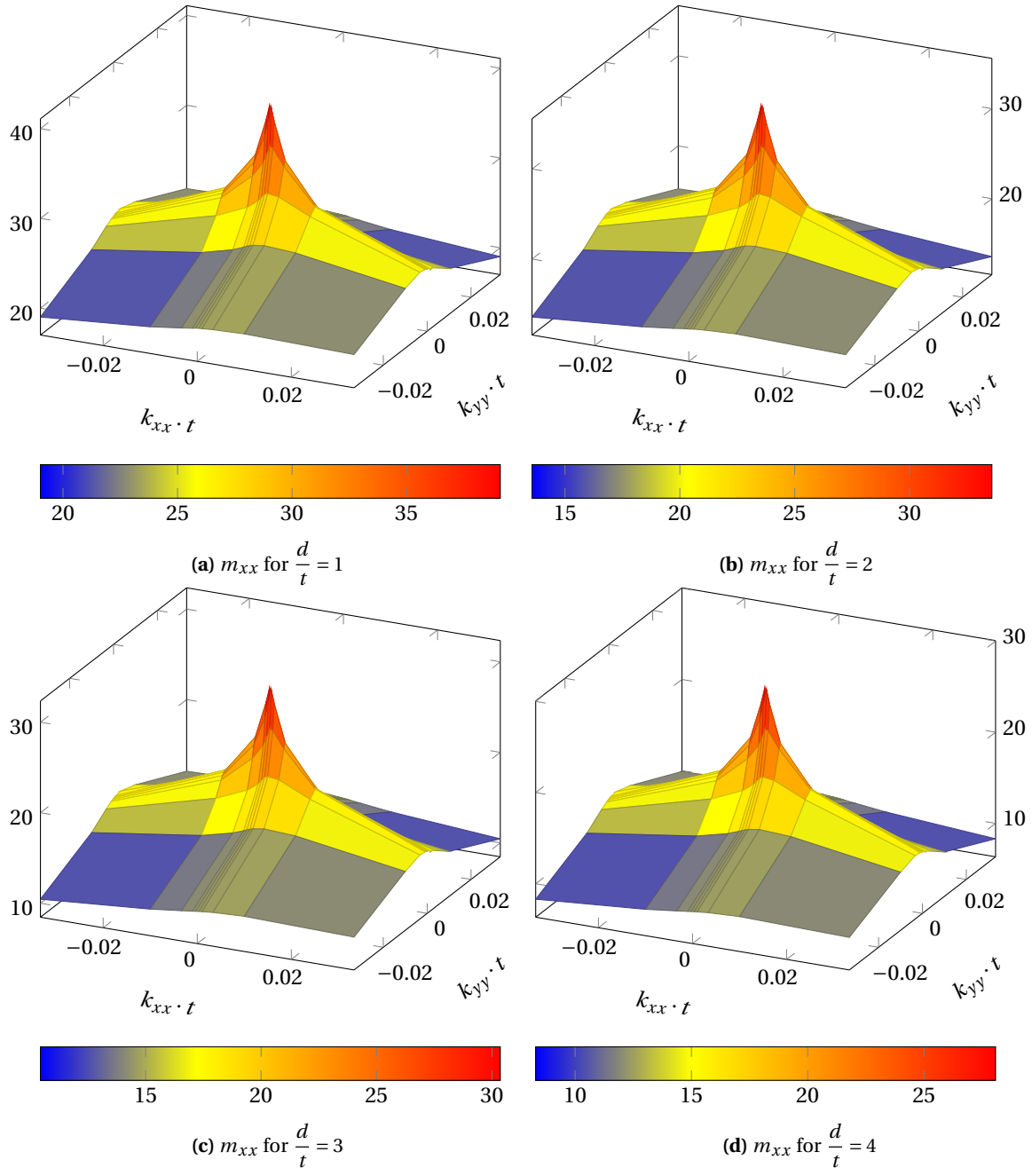


Figure 4.1: Simulations of shells with a point load at the origin and different curvatures. The coordinate system used is also shown (figure 2 from [1]). It is important to note that only a quarter of the model was analysed due to symmetry of loading.



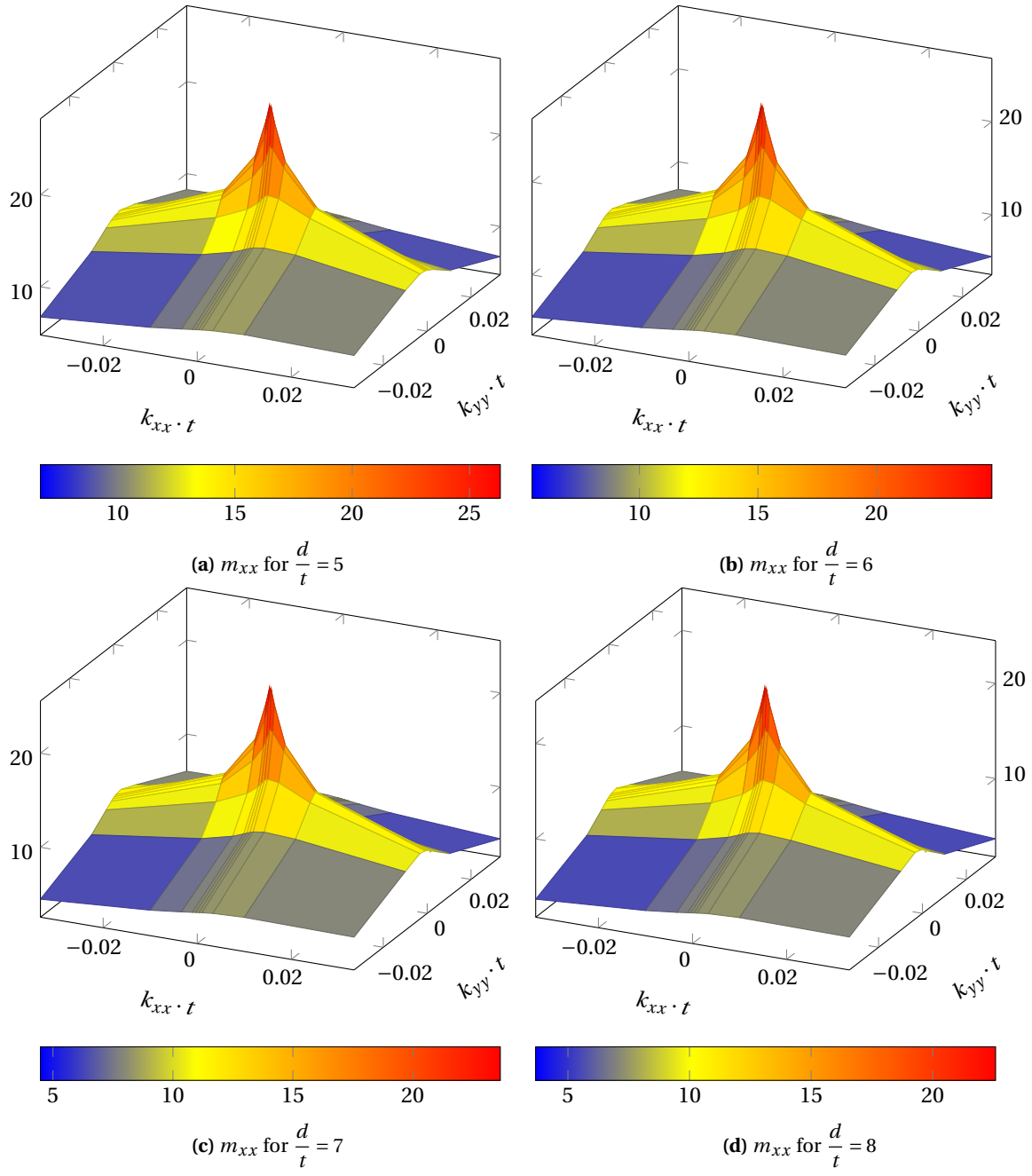
(a) m_{xx} for $\frac{d}{t} = 1$

(b) m_{xx} for $\frac{d}{t} = 2$

(c) m_{xx} for $\frac{d}{t} = 3$

(d) m_{xx} for $\frac{d}{t} = 4$

Figure 4.2: Plots of M_{xx} due to variation of curvatures in x and y for $\frac{d}{t} = 1$ **to** 4.



(a) m_{xx} for $\frac{d}{t} = 5$

(b) m_{xx} for $\frac{d}{t} = 6$

(c) m_{xx} for $\frac{d}{t} = 7$

(d) m_{xx} for $\frac{d}{t} = 8$

Figure 4.3: Plots of M_{xx} due to variation of curvatures in x and y for $\frac{d}{t} = 5$ **to** 8.

5

OPTIMIZATION OF CONSTANTS

The next step was to use the data obtained and do the curve fitting to obtain the constants in the formula proposed in [1].

5.1. ASSUMPTIONS FOR DERIVATION OF THE FORMULA

Following are the assumptions made for the derived formula:

1. In the parametric study, it was found that the length stopped playing a role as soon as we keep the model length higher than the influence length. Hence it is assumed that the lengths in x and y will always be higher than the influence length of the edge disturbance cause by the point load.
2. As there were no prescribed displacements or thermal strains, the Young's Modulus 'E' will not play a role.
3. Poisson's Ratio was assumed to not have a major influence on the results hence it was assumed to be zero for the initial curve fitting. Its influence will be discussed in the next chapter.
4. Since the moment shares a linear relation with a Point Load in almost all applications of structural mechanics, it was assumed to be constant for all the simulations.

The search for this formula has been on since the last century ([2],[6], [13], [4], [7], [8], [5], [3]). Previous solutions to calculate the Moments under a point load included solving Bessel Functions, partial differential equations among other tools.

In [1], a different approach was tried. It was decided to use the method of **curve fitting** to the numerical data produced by ANSYS. The curves produced by plotting the curvature in x, k_{xx} on the x axis, curvature in y, k_{yy} on the y axis and the obtained moment in x direction directly beneath the theoretical point load, m_{xx} along the z axis were of the nature $\ln(\frac{1}{|x|+|y|})$. It was also assumed that the moments were a function of the ratio of the load distribution diameter (d) and the thickness (t). The formula initially suggested was:

$$m_{xx} = \mathbf{a}P(\ln(\frac{t}{d^2(|\mathbf{b}k_{xx}+\mathbf{c}k_{yy}|+|\mathbf{e}k_{xx}+\mathbf{f}k_{yy}|)}))(1-\nu^2) \tag{5.1}$$

where a, b, c, e, f were constants to be optimized by curve fitting.

The curve fit function from the SciPy library of Python was used [14]. This function uses the method of **least squares** for fitting the data.

5.2. METHOD OF NON-LINEAR LEAST SQUARES

Fitting data in the least-squares formulation minimizes the sum of the squared residuals where a residual is "*the difference between an observed value and the fitted value provided by a model*" [15].

Least squares problems can broadly be defined as ordinary least squares OR non-linear least squares, depending on whether or not the parameters that need to be optimized can have a linear relation. The non-linear problem is usually solved by running an iterative analysis where "algorithms are used to find the value of the parameters that minimizes the objective" [15].

SciPy uses the Levenberg-Marquardt algorithm [14] for unbounded problems (*i.e* no bounds given for the parameters to be in) which needs initial values for the parameters. Then, the parameters are refined iteratively, that is, the values are obtained by successive approximation.

For our case, optimization can be regarded as follows: There are two sets of variables, one being the **varied data** and the other being the data **dependent** on the variation of the first set. We decide what parameters were to be varied in the equation 5.1 and those parameters were members of the set called the **varied data**. The results of that variation, the data obtained from ANSYS, became members of the set called the **dependent data**. They were then put through the aforementioned algorithm after giving initial values to the constants and then based on least square minimization of the residuals, the constants were found and then the % difference between the values calculated by the formula and the ANSYS results was compared.

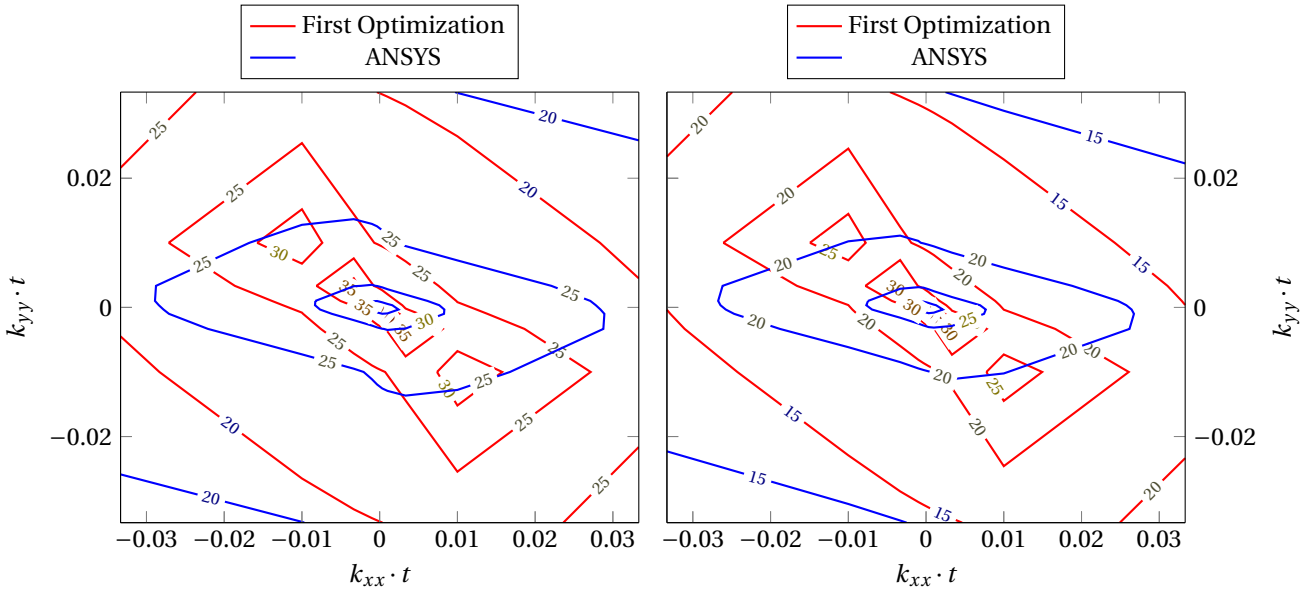
5.3. CURVE FITTING TRIAL 1

Initially, only two parameters in the equation 5.1 were selected namely, $k_{xx} \cdot t$ and $k_{yy} \cdot t$. Based on the authors definition of the optimization problem, they were members of the varied data set. This was done to see whether the variation of $\frac{d}{t}$ led to quantifiable variation in the value of the optimized constants that would be obtained via Python. In other words, to see whether the constants were dependent on the $\frac{d}{t}$ ratio or not.

The difference in ANSYS vs the formula was quite a lot suggesting better optimization was possible(refer figures 5.1 and 5.2). However, the relation between the constants for each $\frac{d}{t}$ ratio looked quite linear but did not lead to any fruitful conclusions except that they were quite close to each other. It was then decided to try another variation which is described in the next section.

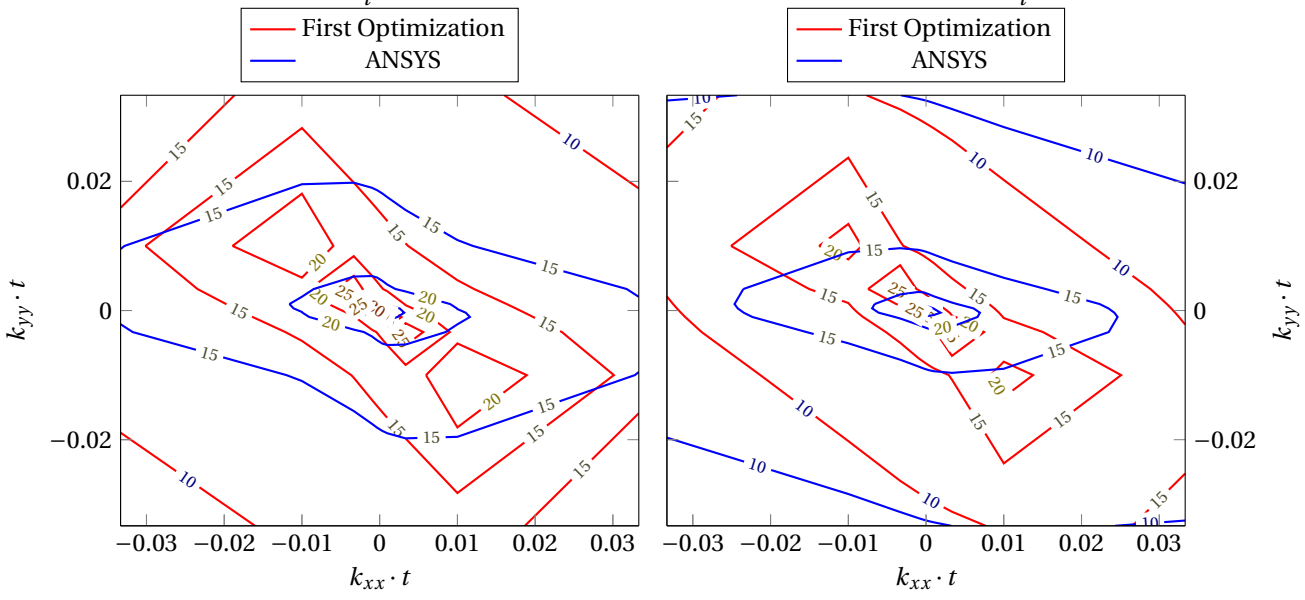
Table 5.1: Values of optimized variables for trial 1.

0.03977	0.039706	0.039526	0.039267	0.038927	0.038361	0.037736	0.036932	**a**
0.034141	0.034129	0.033212	0.032137	0.030988	0.02946	0.027992	0.019781	**b**
0.062153	0.060493	0.058967	0.057283	0.055118	0.051106	0.047276	0.04071	**c**
-0.13606	-0.13522	-0.13152	-0.12689	-0.12172	-0.11486	-0.10827	-0.10603	**e**
-0.00843	-0.00808	-0.00788	-0.00765	-0.00734	-0.00686	-0.00645	-0.01212	**f**
$\frac{d}{t}=1$	$\frac{d}{t}=2$	$\frac{d}{t}=3$	$\frac{d}{t}=4$	$\frac{d}{t}=5$	$\frac{d}{t}=6$	$\frac{d}{t}=7$	$\frac{d}{t}=8$	



(a) Contours of m_{xx} for $\frac{d}{t} = 1$

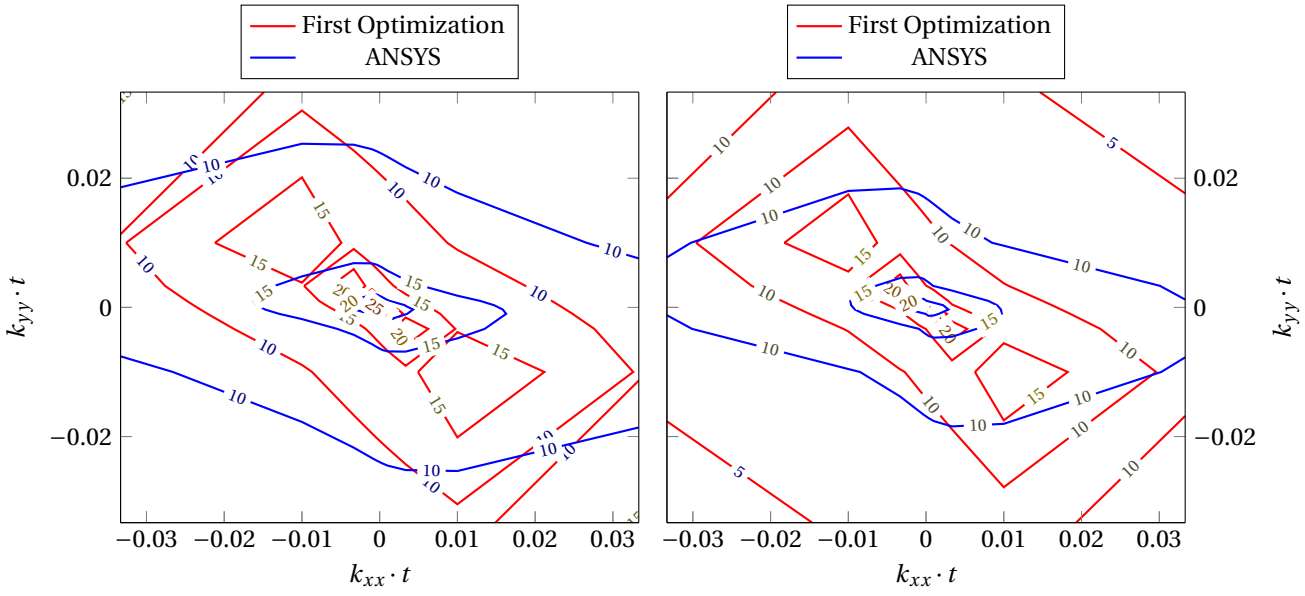
(b) Contours of m_{xx} for $\frac{d}{t} = 2$



(c) Contours of m_{xx} for $\frac{d}{t} = 3$

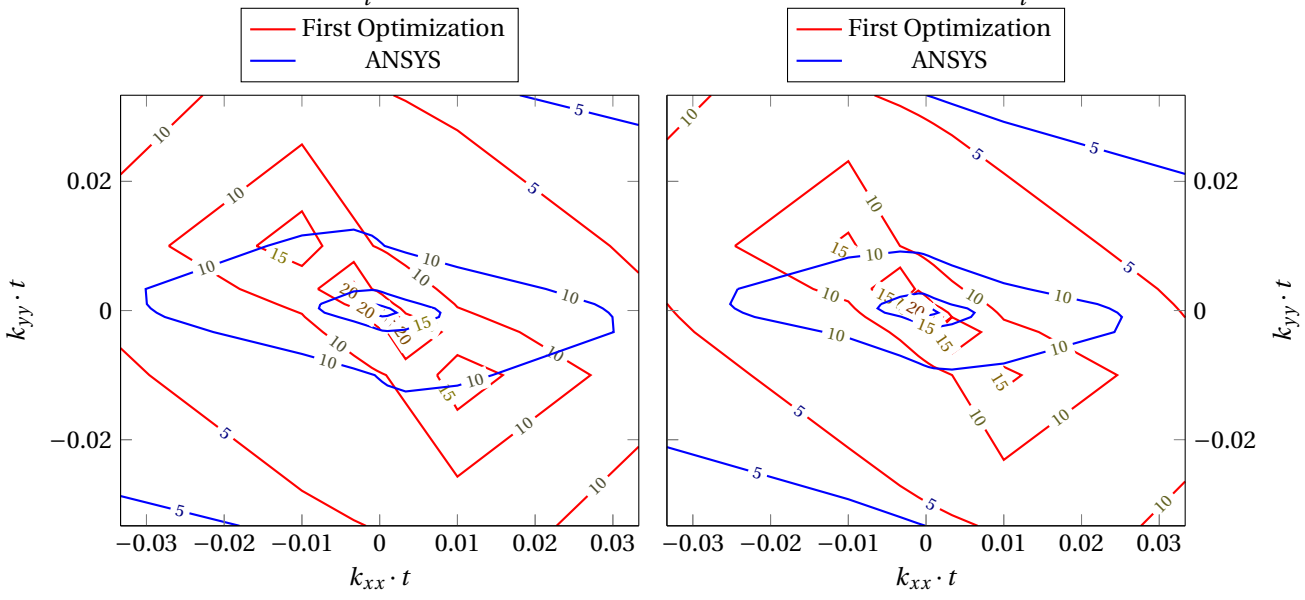
(d) Contours of m_{xx} for $\frac{d}{t} = 4$

Figure 5.1: Comparison of Mxx values of Ansys and first optimization of the formula for $\frac{d}{t} = 1$ **to** 4.



(a) Contours of m_{xx} for $\frac{d}{t} = 5$

(b) Contours of m_{xx} for $\frac{d}{t} = 6$



(c) Contours of m_{xx} for $\frac{d}{t} = 7$

(d) Contours of m_{xx} for $\frac{d}{t} = 8$

Figure 5.2: Comparison of Mxx values of Ansys and first optimization of the formula for $\frac{d}{t} = 5$ **to** 8.

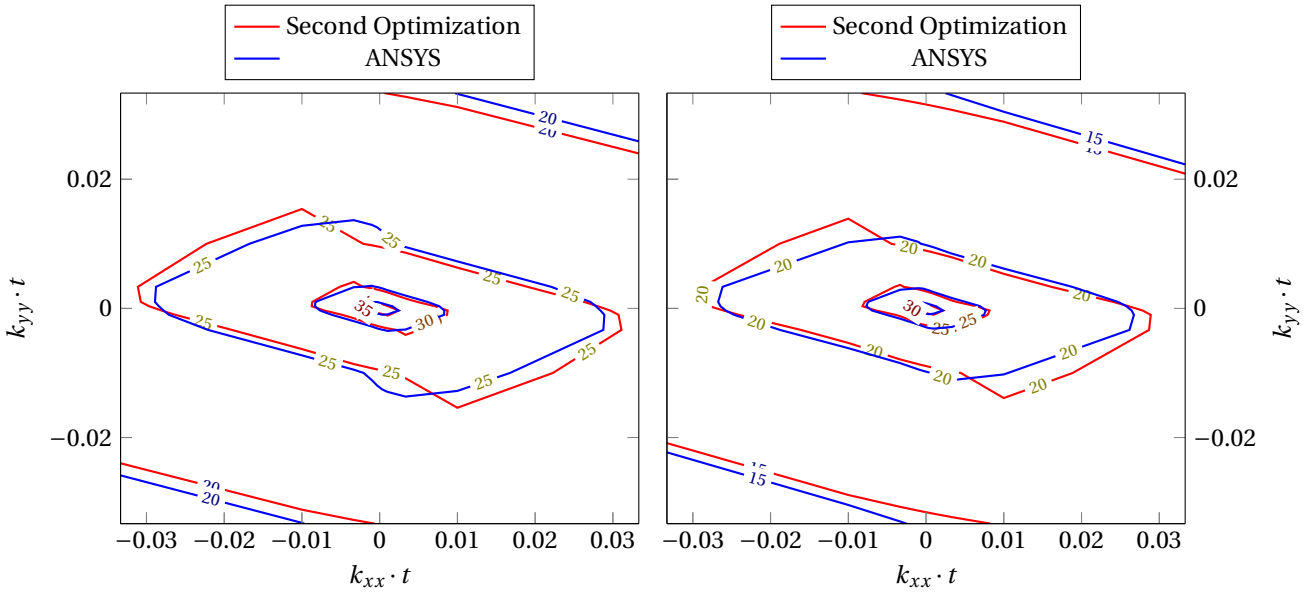
5.4. CURVE FITTING TRIAL 2

It was found that the constants were always very close by to each other when $\frac{d}{t}$ was varied and hence it was decided to add $\frac{d}{t}$ to the parameters in the varied data set. The advantage was that instead of 5 constants for each ratio that would make a total of 40 values of constants as found in the previous trial, there would be just 5 constants applicable for all the ratios.

This time, most of the results were within a percentage difference of ±20%. Figures 5.3 and 5.4 clearly show an improvement from figures 5.1 and 5.2 respectively.

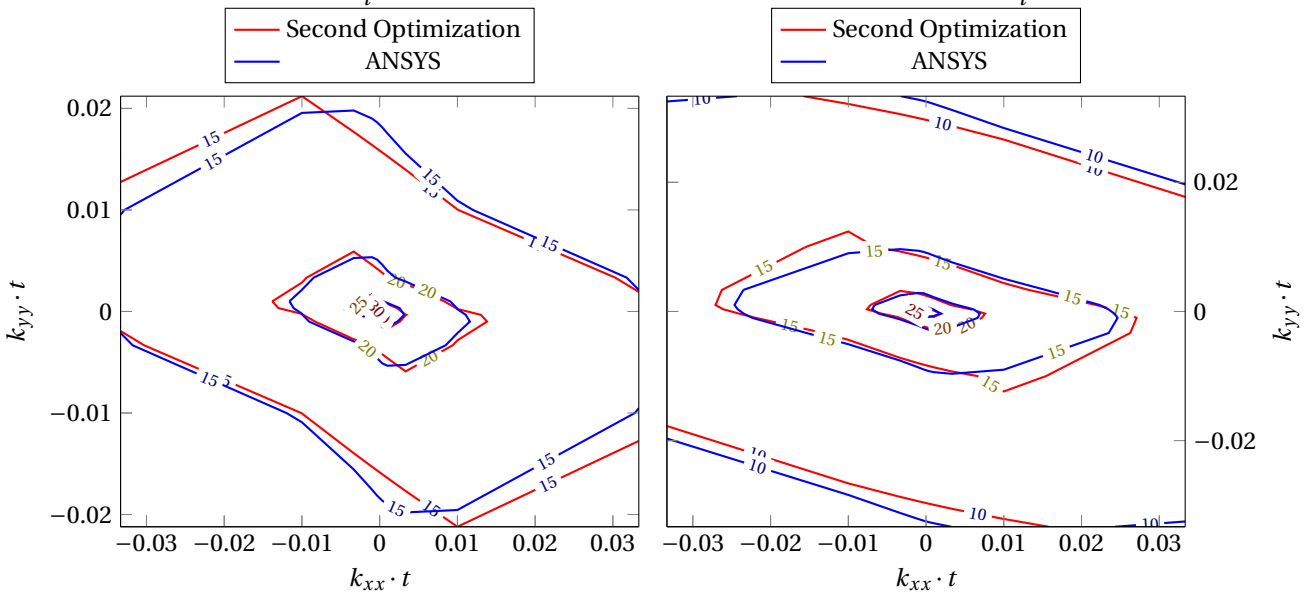
Table 5.2: Variation of optimized variables for trial 2.

0.038808	**a**
0.007246	**b**
0.119327	**c**
-0.05291	**e**
-0.0298	**f**



(a) Contours of m_{xx} for $\frac{d}{t} = 1$

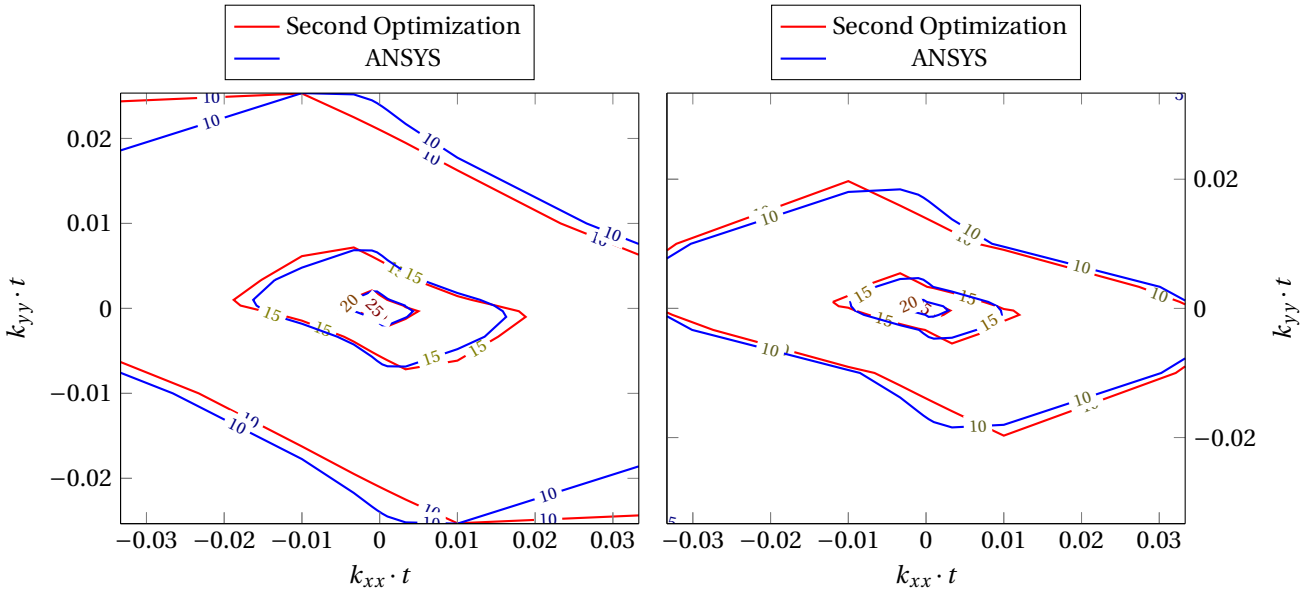
(b) Contours of m_{xx} for $\frac{d}{t} = 2$



(c) Contours of m_{xx} for $\frac{d}{t} = 3$

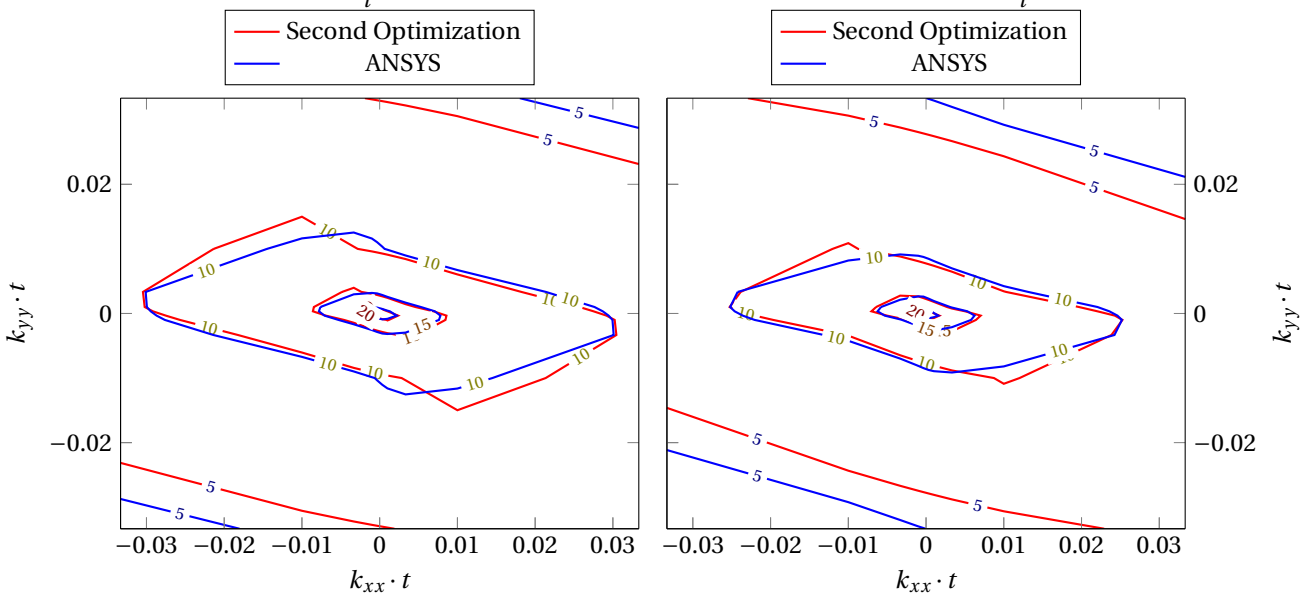
(d) Contours of m_{xx} for $\frac{d}{t} = 4$

Figure 5.3: Comparison of Mxx values of Ansys and second optimization of the formula for $\frac{d}{t} = 1$ **to** 4.



(a) Contours of m_{xx} for $\frac{d}{t} = 5$

(b) Contours of m_{xx} for $\frac{d}{t} = 6$



(c) Contours of m_{xx} for $\frac{d}{t} = 7$

(d) Contours of m_{xx} for $\frac{d}{t} = 8$

Figure 5.4: Comparison of Mxx values of Ansys and second optimization of the formula for $\frac{d}{t} = 5$ **to** 8.

6

THE EFFECT OF POISSON'S RATIO ON THE FORMULA

It was then decided to check whether the dependence on Poisson's ratio in the formula could be linear instead of quadratic.

For this purpose, the following scenario was set:

1. $\frac{d}{t}$ was set at 5.
2. 3 sets of curvatures were checked:

Table 6.1: Set of curvatures for checking the effect of Poisson's Ratio.

Shape 1 (k_{xx} , k_{yy})	**Shape 2** (k_{xx} , k_{yy})	**Shape 3** (k_{xx} , k_{yy})
$0.033333, 0.03333$	$0.0333, 0.000333$	$0.0333, -0.0333$

3. For each shape, 3 analyses were done with the Poisson's ratio set at 0, 0.35 and 0.49 respectively.

The reason why these three shapes were chosen lies in the fact that these three shapes are at the extreme ends of the spectrum of choices available to us in terms of curvatures in x and y measured at the origin. **Shape 1** signified a **sphere**, **Shape 2** a **Cylinder** and **Shape 3** a **Hypar shell**. It was found that the relation was **more linear** than previously observed (refer fig 6.1). A change in the formula was made and the effect of Poisson's ratio was made into a linear function.

$$m_{xx} = \mathbf{a}P(\ln(\frac{t}{d^2(|\mathbf{b}k_{xx} + \mathbf{c}k_{yy}| + |\mathbf{e}k_{xx} + \mathbf{f}k_{yy}|)}))(1+\nu) \tag{6.1}$$

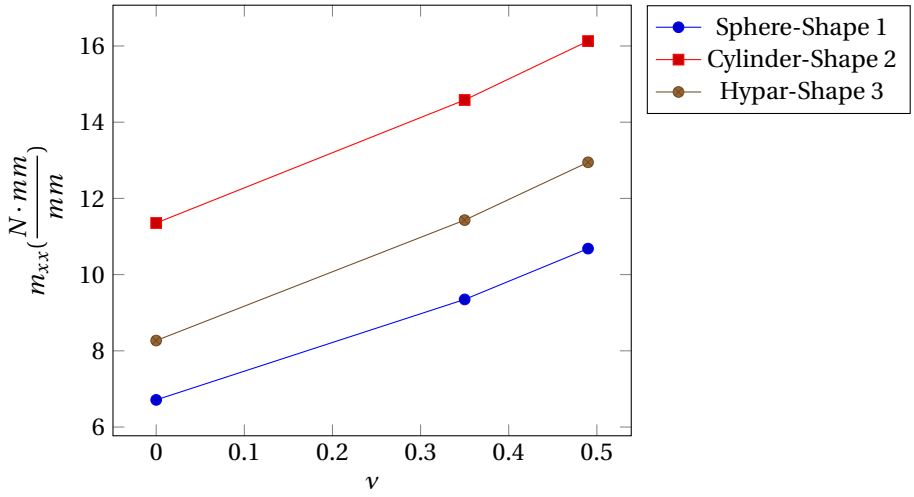


Figure 6.1: Variation of m_{xx} with change in Poisson's Ratio

7

CONCLUSIONS AND DISCUSSIONS

- A formula was developed for estimating the moments under a point load on a shell structure when the curvatures in x and y directions along with the diameter of distribution of the point load and the thickness are known. Only one kind of boundary condition was used throughout.
- The number of elements along x and y directions could be reduced to 20 in each direction for the purpose of this exercise. However, keeping in mind that shells can have complicated shapes, it was decided to keep the number of elements as 80 in both directions.
- The lengths of the finite element model of the shell structure in x and y directions were chosen to be such that they were sufficiently larger than the influence length of the structure. The current parameter for length keeps it at around 9 times the influence length.
- Varying the thickness does not change the value of the moments in x obtained from ANSYS **provided** that the $k_{xx} \cdot t$, $k_{yy} \cdot t$ and the $\frac{d}{t}$ ratio remains the same for the model.
- Every shell structure has curvatures in x and y directions. In the study conducted for obtaining the moment results, every simulation was a different shell structure with different curvatures. The range of curvatures considered in the current study limited us to only a 100 different combinations of curvatures per $\frac{d}{t}$ ratio. For further studies, more types of curvatures could be checked to improve the accuracy of the equation derived.
- Poisson's ratio ν, which was initially assumed to have a quadratic influence in the form of the term $1-\nu^2$ was found to have a linear influence in the form of $1+\nu$.
- ANSYS Parametric Design Language or APDL [16] offered flexibility in terms of creation of the finite element model as algorithms were defined for creation of nodes and elements and placement of boundary conditions and loads which manually would take immense amount of time. APDL Scripts also allowed usage of batch mode which was crucial in obtaining the results from the program for the hundreds of simulations conducted throughout this additional thesis.
- Python [17] [18] was the programming language of choice in this project because of the multitude of options to manipulate text files and excel sheets as well as scientific computing through the usage of package SciPy [14] which was used to conduct the optimization trials of equation 5.1 to find out the values of the constants. This was advantageous as the author did not need to call other programs or software for different functions. Python could handle everything itself.
- The method of non-linear least squares optimization [15] was used to fit the data obtained from ANSYS, reason being that it was the method available in SciPy. Other methods of data fitting which were not explored due to restrictions of time could maybe give better estimations of the constants.

- The plotting capabilities of MS Excel were not sufficient for some of the plots that were needed to observe the fitting of data. For this purpose, other programs such as MATLAB [19], gnuplot [20] and LaTeX[21] offer more options after the data is put in the format desired by the respective programs. It should be noted however that gnuplot is used by LaTeXto make contour plots. The author used Python to generate input files for usage of these programs for plotting. For this report, LaTeXwas used to plot all the figures.

BIBLIOGRAPHY

[1] P. Hoogenboom, Y. Chenjie, and K. Taneja, *Moments due to concentrated loads on shell structures,* Heron Vol. 61 (2016) No. 3. under review.

[2] G. Chernyshev, *On the action of Concentrated Forces and Moments on an Elastic Thin Shell of Arbitrary Shape,* Prikl. Mat. Mekh. **27**, 126 (1963).

[3] S. S. Łukasiewicz, *The solution for concentrated loads on shells by means of Thompson functions,* ZAMM - Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik (1968), 10.1002/zamm.19680480405.

[4] S. A. Lukasiewicz, *Introduction of Concentrated Loads in Plates and Shells,* Progress in Aerospace Science **17**, 109 (1976).

[5] J. Simmonds and C. Tropf, *The Fundamental (Normal Point Load) Solution for a Shallow Hyperbolic Paraboloidal Shell,* SIAM J. APPL. MATH **27**, 102 (1974).

[6] W. Flügge and R. E. Elling, *Singular solutions for shallow shells,* International Journal of Solids and Structures **8**, 227 (1972).

[7] T. Matsui and O. Matsuoka, *The fundamental solution in the theory of shallow shells,* International Journal of Solids and Structures **14**, 971 (1978).

[8] M. Samuchin and J. Dundurs, *Transmission of Concentrated Forces into Prismatic Shells-I ,* International Journal of Solids and Structures **7**, 1627 (1971).

[9] J. Blaauwendraad and J. H. Hoefakker, *Structural Shell Analysis- Understanding and Application,* (Springer Netherlands, 2014) Chap. 1, p. 8.

[10] ANSYS Inc., *ANSYS Mechanical APDL Element Reference*, release 17.0 ed. ().

[11] ANSYS Inc., *ANSYS Mechanical APDL Command Reference*, release 17.0 ed. ().

[12] S. Lukasiewicz, *Local loads in plates and shells,* (Springer Netherlands, 1979) Chap. 9.

[13] K. Forsberg and W. Flugge, *Point Load on a Shallow Elliptic Paraboloid,* Journal of Applied Mechanics **3**, 575 (1966).

[14] E. Jones, T. Oliphant, P. Peterson, *et al.*, *SciPy: Open source scientific tools for Python,* (2001–).

[15] Wikipedia, *Least squares — wikipedia, the free encyclopedia,* (2016).

[16] ANSYS Inc., *ANSYS Parametric Design Language Guide*, release 17.0 ed. ().

[17] A. Sweigart, *Automate the boring stuff-python,* .

[18] Python Software Foundation, *Python 2.7.12 documentation,* .

[19] MATLAB, *version 9.1.0.441655 (R2016b)* (The MathWorks Inc., Natick, Massachusetts, 2016).

[20] T. Williams, C. Kelley, and many others, *Gnuplot 4.4: an interactive plotting program,* (2016).

[21] Wikibooks, *Latex — wikibooks, the free textbook project,* (2016).

A

APPENDIX-A

A.1. SHELL1.MAC

Following is the ANSYS APDL script Shell1.mac.

```
! Rectangular shell, paraboloid, 8 node elements
! version 30 april 2015, Pierre Hoogenboom
!
t   = 1  ! mm thickness
kxy = 1/3000 ! 1/mm twist curvature
E   = 1.0e5 ! N/mm2 Young's modulus
nu  = 0.0 ! -  Poisson's ratio
p   = 0.001 ! N/mm2 distributed load
lx  = 2000 ! mm span in the x direction (half the span is modelled.)
ly  = 2000 ! mm span in the y direction (half the span is modelled.)
nx  = 20 ! -number of elements in the x direction
ny  = 20 ! -number of elements in the y direction

/PREP7
MPTEMP,,,,,,,,  ! material: isotropic
MPTEMP,1,0
MPDATA,EX,1,,E
MPDATA,PRXY,1,,nu
ET,1,SHELL281 ! element type: 8 node quadrilateral
R,1,t,t,t,t, , ,  ! element thickness

ty=1 ! insert nodes
*DO,j,0,2*ny
 ty=-ty
 tx=1
 *DO,i,0,2*nx
  tx=-tx
  *IF,tx+ty,LT,1,THEN
   x=i*lx/nx/4
   y=j*ly/ny/4
   z=x*y*kxy
   N,,x,y,z,,,
  *ENDIF
 *ENDDO
*ENDDO
```

```
SHPP,OFF ! no warning aspect ratio
*DO,j,1,ny ! insert elements
 *DO,i,1,nx
  k1=1+(i-1)*2+(j-1)*(3*nx+2)
  k2=1+i+(2+(j-1)*3)*nx+(j-1)*2
  k3=1+(i-1)*2+j*(3*nx+2)
  E,k3,k3+2,k1+2,k1,k3+1,k2+1,k1+1,k2
 *ENDDO
*ENDDO

*DO,i,1,2*nx+1 ! insert symmetry boundary conditions
 D,i,,0,,,,UY,ROTX,ROTZ,,,
*ENDDO
*DO,j,1,ny+1
 D,(j-1)*(3*nx+2)+1,,0,,,,UX,ROTY,ROTZ,,,
*ENDDO
*DO,j,1,ny
 D,2*nx+2+(j-1)*(3*nx+2),,0,,,,UX,ROTY,ROTZ,,,
*ENDDO

*DO,j,1,2*nx+1 ! insert roller edges
 i=(3*nx+2)*ny+j
 D,i,,0,,,,UX,UY,UZ,,,
*ENDDO
*DO,j,0,ny
 i=2*nx+1+j*(3*nx+2)
 D,i,,0,,,,UX,UY,UZ,,,
*ENDDO
*DO,j,1,ny
 i=j*(3*nx+2)
 D,i,,0,,,,UX,UY,UZ,,,
*ENDDO

*DO,j,1,ny ! add load
 *DO,i,1,nx
   F,(j-1)*(3*nx+2)+i*2+1,FZ,p*lx/nx*ly/ny
 *ENDDO
*ENDDO

FINISH

/SOLU ! compute
SOLVE
FINISH

/POST1
*GET,w,NODE,1,U,Z ! deflection w in the middle
SHELL,TOP
*GET,sxxt,NODE,1,S,X ! stress sxx in the top surface (z>0)
*GET,syyt,NODE,1,S,Y ! stress syy
*GET,sxyt,NODE,1,S,XY ! stress sxy
SHELL,BOT
*GET,sxxb,NODE,1,S,X ! stress sxx in the bottom surface (z<0)
*GET,syyb,NODE,1,S,Y ! stress syy
*GET,sxyb,NODE,1,S,XY ! stress sxy
nxx=(sxxb+sxxt)*t/2 ! membrane forces
```

```
nyy=(syyb+syyt)*t/2
nxy=(sxyb+sxyt)*t/2
mxx=(sxxb-sxxt)*t*t/12 ! moments
myy=(syyb-syyt)*t*t/12
mxy=(sxyb-sxyt)*t*t/12


*CFOPEN,out,txt,,APPEND ! open file out.txt
*VWRITE,kxx,kyy,t,E,nu,p,lx,ly,nx,ny,h,w,mxx,myy,nxx,nyy,
(F13.9,F13.9,F7.2,F10.0,F7.2,F10.0,F10.0,F10.0,F5.0,F5.0,F6.2,F13.5,F13.5,F13.5,F13.5,F13.5)
*CFCLOS ! close out.txt
*UILIST,out.txt ! pop up out.txt

FINISH
```

B

APPENDIX-B

B.1. SHELL24.MAC

Following is the ANSYS APDL script Shell24.mac.

```
!Hypar shell, paraboloid, 8 node elements
! version 19 October 2016, Pierre Hoogenboom
!
tk = 1  ! mm thickness
kyy = 1/3000/tk ! 1/mm curvature in the y direction
kxx = -kyy !Reference shape is a Hypar.
Ek = 1.0e5 ! N/mm^2 Young's modulus
nu = 0.0  ! -  Poisson's ratio
Pk = 100 ! N point load
dk = 4*tk ! mm diameter of the point load distribution area
!MODIFIED LENGTH. Changed from 14 to 9 after parametric study.
*IF,kyy,EQ,0,THEN !
 lx = 9*((tk/ABS(kxx))**0.5)
*ELSE
 lx = 9*((tk/ABS(kyy))**0.5)
*ENDIF
*IF,kxx,EQ,0,THEN
 ly = 9*((tk/ABS(kyy))**0.5)
*ELSE
 ly = 9*((tk/ABS(kxx))**0.5)
*ENDIF
nx = 80 ! -number of elements in the x direction
ny = 80 ! -number of elements in the y direction
hk = dk/10 ! mm element size in the middle_MODIFIED from 20 to 10.
bk = hk ! mm load patch size (load is applied in small patches)_MODIFIED from
bk/hk =4 to 1.

/PREP7
MPTEMP,,,,,,,,  ! material: isotropic
MPTEMP,1,0
MPDATA,EX,1,,Ek
MPDATA,PRXY,1,,nu
ET,1,SHELL281 ! element type: 8 node quadrilateral
R,1,tk,tk,tk,tk, , ,  ! element thickness

gx=lx/2-nx*hk ! insert nodes
gy=ly/2-ny*hk
```

```
mx=lx-(8*nx-6)*nx**2*hk
my=ly-(8*ny-6)*ny**2*hk
qx=4*nx*(1+3*nx-4*nx**2)
qy=4*ny*(1+3*ny-4*ny**2)
ty=1
*DO,j,0,2*ny
 ty=-ty
 tx=1
 *DO,i,0,2*nx
  tx=-tx
  *IF,tx+ty,LT,1,THEN
   x=i*(i*(3-2*i)*gx+mx)/qx
   y=j*(j*(3-2*j)*gy+my)/qy
   z=(x*x*kxx+y*y*kyy)/2
   N,,x,y,z,,,
  *ENDIF
 *ENDDO
*ENDDO

SHPP,OFF ! no warning aspect ratio
*DO,j,1,ny ! insert elements
 *DO,i,1,nx
  k1=1+(i-1)*2+(j-1)*(3*nx+2)
  k2=1+i+(2+(j-1)*3)*nx+(j-1)*2
  k3=1+(i-1)*2+j*(3*nx+2)
  E,k3,k3+2,k1+2,k1,k3+1,k2+1,k1+1,k2
 *ENDDO
*ENDDO

*DO,i,1,2*nx+1 ! insert symmetry boundary conditions
 D,i,,0,,,,UY,ROTX,ROTZ,,,
*ENDDO
*DO,j,1,ny+1
 D,1+(j-1)*(3*nx+2),,0,,,,UX,ROTY,ROTZ,,,
*ENDDO
*DO,j,1,ny
 D,2*nx+2+(j-1)*(3*nx+2),,0,,,,UX,ROTY,ROTZ,,,
*ENDDO

*DO,j,1,2*nx+1 ! insert roller edges
 i=(3*nx+2)*ny+j
 NANG,i,1,0,NX(i)*kxx,,,,-NX(i)*kxx,-NY(i)*kyy,1
 D,i,,0,,,,UX,UY,,,,
*ENDDO
*DO,j,1,ny
 i=2*nx+1+(j-1)*(3*nx+2)
 NANG,i,,,,0,1,NY(i)*kyy,-NX(i)*kxx,-NY(i)*kyy,1
 D,i,,0,,,,UX,UY,,,,
 i=j*(3*nx+2)
 NANG,i,,,,0,1,NY(i)*kyy,-NX(i)*kxx,-NY(i)*kyy,1
 D,i,,0,,,,UX,UY,,,,
*ENDDO

sx=1 ! insert point load
a=dk/2-NX(2*sx+1) ! determine the range sx and sy of loaded elements
*DOWHILE,a
```

```
 sx=sx+1
 a=dk/2-NX(2*sx+1)
*ENDDO
sy=1
a=dk/2-NY((3*nx+2)*sy+1)
*DOWHILE,a
 sy=sy+1
 a=dk/2-NY((3*nx+2)*sy+1)
*ENDDO

FCUM,ADD
n=0 ! n = number of rings
a=dk/2
*DOWHILE,a
 n=n+1
 a=a-bk
*ENDDO
dr=dk/2/n ! dr = ring width
*DO,k,1,n ! k = ring number
 m=0 ! m = number of sectors
 a=3.1415/2*k*dr
 *DOWHILE,a
  m=m+1
  a=a-dr
 *ENDDO
 df=3.1415/2/m ! df = sector angle
 dP=Pk/(1/4*3.1415*dk*dk)*(2*k-1)/2*dr*dr*df ! dP = load on the sector
 ro=(12*k*k-12*k+4)/(6*k-3)*dr*SIN(df/2)/df ! ro = sector centre of gravity
 *DO,g,1,m ! g = sector number
  x=ro*cos((g-0.5)*df)
  y=ro*sin((g-0.5)*df)

  r=dk*dk ! find the closest node
  q1=1 ! (look no further than sx or sy elements from the origin)
  *DO,j,1,sy
   *DO,i,1,sx
    q=(j-1)*(3*nx+2)+2*i-1
    a=(NX(q)-x)**2+(NY(q)-y)**2
    *IF,a,LT,r,THEN
     r=a
     q1=q
    *ENDIF
    q=(j-1)*(3*nx+2)+2*i
    a=(NX(q)-x)**2+(NY(q)-y)**2
    *IF,a,LT,r,THEN
     r=a
     q1=q
    *ENDIF
    q=(j-1)*(3*nx+2)+(2*nx+1)+i
    a=(NX(q)-x)**2+(NY(q)-y)**2
    *IF,a,LT,r,THEN
     r=a
     q1=q
    *ENDIF
   *ENDDO
  *ENDDO
```

```
  F,q1,FZ,dP ! move force to node and add
  F,q1,MX,dP*(y-NY(q1))
  F,q1,MY,dP*(NX(q1)-x)
*ENDDO
*ENDDO
FINISH

/SOLU ! compute
SOLVE
FINISH

/POST1
*GET,w,NODE,1,U,Z ! deflection w under the point load
SHELL,TOP
*GET,sxxt,NODE,1,S,X ! stress sxx under point load in the top surface (z>0)
*GET,syyt,NODE,1,S,Y ! stress syy
*GET,sxyt,NODE,1,S,XY ! stress sxy
SHELL,BOT
*GET,sxxb,NODE,1,S,X ! stress sxx under point load in the bottom surface (z<0)
*GET,syyb,NODE,1,S,Y ! stress syy
*GET,sxyb,NODE,1,S,XY ! stress sxy
nxx=(sxxb+sxxt)*tk/2 ! membrane forces
nyy=(syyb+syyt)*tk/2
nxy=(sxyb+sxyt)*tk/2
mxx=(sxxb-sxxt)*tk*tk/12 ! moments
myy=(syyb-syyt)*tk*tk/12
mxy=(sxyb-sxyt)*tk*tk/12

l=0 ! influence length in the x direction
v=1
i=1
mi=mxx
*DOWHILE,v
 j=i+2
 a=SQRT((NX(j)-NX(i))**2+(NZ(j)-NZ(i))**2)
 SHELL,TOP
 *GET,sxxt,NODE,j,S,X
 SHELL,BOT
 *GET,sxxb,NODE,j,S,X
 mj=(sxxb-sxxt)*tk*tk/12
 *IF,mi*mj,GT,0,THEN
  l=l+a
 *ELSE
  l=l+a*mi/(mi-mj)
  v=-1
 *ENDIF
 *IF,j,EQ,2*nx+1,THEN
  v=-1
 *ENDIF
 i=j
 mi=mj
*ENDDO
lix=2*l

l=0 ! influence length in the y direction
```

```
v=1
i=1
mi=myy
*DOWHILE,v
 j=i+3*nx+2
 a=SQRT((NY(j)-NY(i))**2+(NZ(j)-NZ(i))**2)
 SHELL,TOP
 *GET,syyt,NODE,j,S,Y
 SHELL,BOT
 *GET,syyb,NODE,j,S,Y
 mj=(syyb-syyt)*tk*tk/12
 *IF,mi*mj,GT,0,THEN
  l=l+a
 *ELSE
  l=l+a*mi/(mi-mj)
  v=-1
 *ENDIF
 *IF,j,EQ,ny*(3*nx+2)+1,THEN
  v=-1
 *ENDIF
 i=j
 mi=mj
*ENDDO
liy=2*l

*CFOPEN,out,txt,,APPEND ! open file out.txt
*VWRITE,kxx,kyy,tk,Ek,nu,Pk,dk,lx,ly,nx,ny,hk,bk,mxx,myy
(F13.9,F13.9,F7.2,F10.0,F7.2,F10.0,F7.2,F10.0,F10.0,F5.0,F5.0,F6.2,F13.5,F13.5,F13.5)
*CFCLOS ! close out.txt
*UILIST,out.txt ! pop up out.txt


FINISH
```

C

APPENDIX-C

The following codes were used to conduct the Parametric Studies.

C.1. CODE FOR VARIATION OF LENGTH IN X AND Y

```
'''
Python Modules used.
'''
import math
import csv
'''
Load all rows as strings from the input parameter file (.csv file) which are then saved as a list of lists IP which contains as many lists as there are rows.
 The data in the file is as follows:
tk/ Em / nu / Pk / dk / kxx / kyy
1 / 1.00E+05 / 0 / 100 / 4 / 0.000333333 / -0.000333333

where :
tk - thickness of shell (mm)
Em - Young's Modulus (N/mm2)
nu - Poisson's Ratio
Pk - Point Load applied in Mechanical Model (N)
dk - Diameter of Influence of Point Load in FE Model (mm)
kxx - Curvature in x (1/mm)
kyy - Curvature in y (1/mm)

'''
# Load all rows as strings from the input parameter file into a variable called IP
with open("D:\\Additional Thesis\\Batch mode scripts\\Input_Parameters.csv", "r") as f_input:
    csv_reader = csv.reader(f_input)
    IP = list (csv_reader)
print IP
'''
Load all rows as strings from the file containing values of the coefficients in the formula for lengths in x and y (.csv file) which are then saved as a list of lists VL which contains as many lists as there are rows.
The values are :
lx  ly
1   1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19 20
2   1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19 20
3   1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19 20
4   1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19 20
5   1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19 20
6   1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19 20
7   1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19 20
8   1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19 20
9   1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19 20
10  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19 20
11  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19 20
12  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19 20
13  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19 20
```

```
14 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
15 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
16 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
17 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
18 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
19 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
20 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20


The Data is to be read as follows:
Coefficients of Length in x values are in the column beneath 'lx' AND Coefficients of Length in y values are in the rows following a value of 'lx'.
The set of formulae governing the magnitude of length can be found in the ANSYS Script.
'''
# Load all rows as strings from the parameter varying file into a variable called VL
with open("D:\\Additional Thesis\\Batch mode scripts\\Length\\Varying_Length.csv", "r") as Chameleon:
    csv_reader = csv.reader(Chameleon)
    VL = list (csv_reader)
print VL
KV=[] #empty list to store IPs as floats
SFS=[]#empty list to store VLs as floats

'''
Load all numerical values in IP and VL into seperate lists. KV will contain only the numerical values of the input parameters.
SFS will contain lists within a list which have the lx and ly values. A list within SFS is interpreted as follows:
The first value is the 'lx' value to be used in a batch of analyses with the rest of the values being the 'ly' values for that batch of analyses.

'''
for i in range(1,len(IP)):
    KV.append([float(x) for x in IP[i ]]) #New Input Paramter list

for i in range(1,len(VL)):
    SFS.append([float(y) for y in VL[i]]) #New Varying Paramter list
'''
Next code block creates the batch file (.bat) neede to run ANSYS in Batch mode.
It creates seperate lists LX and LY from which the values are then picked up
as inputs for each individual analysis in ANSYS. Each line represents one analysis.
An example line will look like:
"C:\Program Files\ANSYS Inc\v160\ANSYS\bin\winx64\ansys160.exe" -b -i Shell24_191016.txt -tk 1.0 -kyy 0.0333 -kxx 0.0333333333333 -Ek 100000.0 -nu 0.0 -Pk 100.0 -dk 5.0 -o out0.txt
'''

# creates the first line for the batch file.
hans_zimmer = open("D:\\Additional Thesis\\Batch mode scripts\\Length\\Parameter_Length.txt","w")
hans_zimmer.write('REM This code will run multiple times on a single script but will change the length in x and y each time. \n')
hans_zimmer.close()

# appends the commands for analysis and replacement of parameter for each run.
batch_file = open("D:\\Additional Thesis\\Batch mode scripts\\Length\\Parameter_Length.txt","a")

for i in range(len(KV)): #loops through each set of input parameters
    for j in range(len(SFS)): #loop for coefficient of lx based on input parameter. Loops through column index 1 in each row of SFS.
        for k in range(1,len(SFS[j])):#loop for coefficient of ly based on input parameter. Loops through each row of SFS.
            if KV[i][1]==0: #check parameter index
                LX = SFS[j][0]*(KV[i][0]/abs(KV[i][2])**0.5)
            else:
                LX = SFS[j][0]*(KV[i][0]/abs(KV[i][1])**0.5)
            if KV[i][2]==0:
                LY = SFS[j][k]*(KV[i][0]/abs(KV[i][1])**0.5)
            else:
                LY = SFS[j][k]*(KV[i][0]/abs(KV[i][2])**0.5)
            batch_file.write('"C:\\Program Files\\ANSYS Inc\\v160\\ANSYS\\bin\\winx64\\ansys160.exe" -b -i Shell24.txt -'+IP[0][0]+' '+IP[1][0]+' -'+IP[0][1]+' '+IP[1][1]+' -'+IP[0][2]+' '+IP[1][2]+' -'+IP[0][3]+' '+IP[1][3]+' -'+IP[0][4]+' '+IP[1][4]+' -'+IP[0][5]+' '+IP[1][5]+' -'+IP[0][6]+' '+IP[1][6]+' -lx '+str(LX)+' -ly '+str(LY)+' -o out'+str(j+1)+str(k)+'.txt \n')#output file is stored as out(abcd), ab-LX,cd-LY loop.
batch_file.close()
''' This part changes the extension of the file to .bat '''
txt2bat = os.path.join("D:\\Additional Master Thesis\\Batch mode scripts\\Length",'Parameter_Length' + '.txt')#changes .txt to .bat
base = os.path.splitext(txt2bat)[0]
```

```
108 os.rename(txt2bat,base + ".bat")
```

Listing C.1: Variation of Length

C.2. CODE FOR VARIATION OF NO. OF ELEMENTS ALONG X AND Y AXES

```
'''
Python Modules used.
'''
import math
import csv
'''
Load all rows as strings from the input parameter file (.csv file) which are then saved as a list of lists IP which contains as many lists as there are rows.
 The data in the file is as follows:
tk/ Em / nu / Pk / dk / kxx / kyy
1 / 1.00E+05 / 0 / 100 / 4 / 0.000333333 / -0.000333333

where :
tk - thickness of shell (mm)
Em - Young's Modulus (N/mm2)
nu - Poisson's Ratio
Pk - Point Load applied in Mechanical Model (N)
dk - Diameter of Influence of Point Load in FE Model (mm)
kxx - Curvature in x (1/mm)
kyy - Curvature in y (1/mm)

'''

# Load all rows as strings from the input parameter file into a variable called IP
with open("D:\\Additional Thesis\\Batch mode scripts\\Input_Parameters.csv", "r") as f_input:
    csv_reader = csv.reader(f_input)
    IP = list (csv_reader)
print IP
'''
Load all rows as strings from the file containing values of the number of elements in x and y directions (.csv file) which are then saved as a list of lists VL which contains as many lists as there are rows.
The values are :
nx ny
10 10 20 30 40 50 60 70 80
20 10 20 30 40 50 60 70 80
30 10 20 30 40 50 60 70 80
40 10 20 30 40 50 60 70 80
50 10 20 30 40 50 60 70 80
60 10 20 30 40 50 60 70 80
70 10 20 30 40 50 60 70 80
80 10 20 30 40 50 60 70 80

The Data is to be read as follows:
nx values are in the column next to 'nx' AND ny values are in the rows following a value of 'nx'.

'''
# Load all rows as strings from the parameter varying file into a variable called VL
with open("D:\\Additional Thesis\\Batch mode scripts\\ElementSize\\Parameter_ElementSize.csv", "r") as Chameleon:
    csv_reader = csv.reader(Chameleon)
    VL = list (csv_reader)
print VL

'''
Load all numerical values in IP and VL into seperate lists. KV will contain only the numerical values of the input parameters.
SFS will contain lists within a list which have the nx and ny values. A list within SFS is interpreted as follows:
The first value is the 'nx' value to be used in a batch of analyses with the rest of the values being the 'ny'values for that batch of analyses.

'''
KV=[] #empty list to store IPs as floats
SFS=[]#empty list to store VLs as floats
for i in range(1,len(IP)):
    KV.append([float(x) for x in IP[i ]]) #New Input Paramter list

for i in range(1,len(VL)):
    SFS.append([float(y) for y in VL[i]]) #New Varying Paramter list

'''
Next code block creates the batch file (.bat) neede to run ANSYS in Batch mode.
```

```
It creates seperate lists NX and NY from which the values are then picked up
as inputs for each individual analysis in ANSYS. Each line represents one analysis.
An example line will look like:
"C:\Program Files\ANSYS Inc\v160\ANSYS\bin\winx64\ansys160.exe" -b -i Shell24_191016.txt -tk 1.0 -kyy 0.0333 -kxx
    0.0333333333333 -Ek 100000.0 -nu 0.0 -Pk 100.0 -dk 5.0 -o out0.txt
'''

# creates the first line for the batch file.
hans_zimmer = open("D:\\Additional Thesis\\Batch mode scripts\\ElementSize\\Parameter_ElementSize.txt","w")
hans_zimmer.write('REM This code will run multiple times on a single script but will change the element size in x and y each time. \n')
hans_zimmer.close()

# appends the commands for analysis and replacement of parameter for each run.
batch_file = open("D:\\Additional Thesis\\Batch mode scripts\\ElementSize\\Parameter_ElementSize.txt","a")

for i in range(len(KV)): #loops through each set of input parameters
    for j in range(len(SFS)): #loop for coefficient of nx based on input parameter. Loops through column index 1 in each row of SFS.
        for k in range(1,len(SFS[j])):#loop for coefficient of ny based on input parameter. Loops through each row of SFS.
            NX = SFS[j][0]
            NY = SFS[j][k]
            batch_file.write('"C:\\Program Files\\ANSYS Inc\\v160\\ANSYS\\bin\\winx64\\ansys160.exe" -b -i Shell24.txt -'+IP
    [0][0]+' '+IP[1][0]+' -'+IP[0][1]+' '+IP[1][1]+' -'+IP[0][2]+' '+IP[1][2]+' -'+IP[0][3]+' '+IP[1][3]+' -'+IP[0][4]+' '+IP[1][4]+' -'+IP
    [0][5]+' '+IP[1][5]+' -'+IP[0][6]+' '+IP[1][6]+' -nx '+str(NX)+' -ny '+str(NY)+' -o out'+str(j+1)+str(k)+'.txt \n')#output file is stored
     as out(abcd), ab-NX,cd-NY loop.
batch_file.close()
''' This part changes the extension of the file to .bat '''
txt2bat = os.path.join("D:\\Additional Master Thesis\\Batch mode scripts\\ElementSize",'Parameter_ElementSize' + '.txt')#changes .
    txt to .bat
base = os.path.splitext(txt2bat)[0]
os.rename(txt2bat,base + ".bat")
```

Listing C.2: Variation of Number of Elements along x and y axes

C.3. CODE FOR VARIATION OF $\frac{d}{h}$ AND $\frac{h}{b}$ RATIOS

```
'''
Python Modules used.
'''
import math
import csv
'''
Load all rows as strings from the input parameter file (.csv file) which are then saved as a list of lists IP which contains as many lists as there are rows.
 The data in the file is as follows:
tk/ Em / nu / Pk / dk / kxx / kyy
1 / 1.00E+05 / 0 / 100 / 4 / 0.000333333 / -0.000333333

where :
tk - thickness of shell (mm)
Em - Young's Modulus (N/mm2)
nu - Poisson's Ratio
Pk - Point Load applied in Mechanical Model (N)
dk - Diameter of Influence of Point Load in FE Model (mm)
kxx - Curvature in x (1/mm)
kyy - Curvature in y (1/mm)

'''
# Load all rows as strings from the input parameter file into a variable called IP
with open("D:\\Additional Thesis\\Batch mode scripts\\Input_Parameters.csv", "r") as f_input:
    csv_reader = csv.reader(f_input)
    IP = list (csv_reader)
print IP
'''
Load all rows as strings from the file containing values of the coefficients in the formula for lengths in x and y (.csv file) which are then saved as a list of lists VL which contains as many lists as there are rows.
The values are :
    hk      bk
    dk/1    hk/1    hk/2    hk/3    hk/4    hk/5    hk/6    hk/7    hk/8
    dk/2    hk/1    hk/2    hk/3    hk/4    hk/5    hk/6    hk/7    hk/8
    dk/5    hk/1    hk/2    hk/3    hk/4    hk/5    hk/6    hk/7    hk/8
    dk/10   hk/1    hk/2    hk/3    hk/4    hk/5    hk/6    hk/7    hk/8
    dk/15   hk/1    hk/2    hk/3    hk/4    hk/5    hk/6    hk/7    hk/8
    dk/20   hk/1    hk/2    hk/3    hk/4    hk/5    hk/6    hk/7    hk/8

The Data is to be read as follows:
hk is the element size in the middle and it's variation along the length can be seen in the formulae in the ANSYS script.
bk is the load patch size. They are related to dk via the ratio dk/hk and hk/bk.
The values of 'hk' are given as a function of dk and dk is picked from the list 'IP'.
Values of 'hk' are in the column next to hk while the corresponding values of 'bk', which are given as a function of hk, are in the rows following a value of hk.

'''
# Load all rows as strings from the parameter varying file into a variable called VL
with open("D:\\Additional Thesis\\Batch mode scripts\\HandB\\Varying_HandB.csv", "r") as Chameleon:
    csv_reader = csv.reader(Chameleon)
    VL = list (csv_reader)
print VL
KV=[] #empty list to store IPs as floats
SFS=[]#empty list to store VLs as floats
'''
Load all numerical values in IP and VL into seperate lists. KV will contain only the numerical values of the input parameters.
SFS will contain lists within a list which have the hk and bk values. A list within SFS is interpreted as follows:
The first value is the 'hk' value to be used in a batch of analyses with the rest of the values being the 'bk' values for that batch of analyses.

'''
for i in range(1,len(IP)):
    KV.append([float(x) for x in IP[i ]]) #New Input Paramter list

for i in range(1,len(VL)):
    SFS.append([float(y) for y in VL[i]]) #New Varying Paramter list
'''
```

```
Next code block creates the batch file (.bat) neede to run ANSYS in Batch mode.
It creates seperate lists HK and BK from which the values are then picked up
as inputs for each individual analysis in ANSYS. Each line represents one analysis.
An example line will look like:
"C:\Program Files\ANSYS Inc\v160\ANSYS\bin\winx64\ansys160.exe" -b -i Shell24_191016.txt -tk 1.0 -kyy 0.0333 -kxx
    0.0333333333333 -Ek 100000.0 -nu 0.0 -Pk 100.0 -dk 5.0 -o out0.txt

'''

# creates the first line for the batch file.
hans_zimmer = open("D:\\Additional Thesis\\Batch mode scripts\\HandB\\Parameter_HandB.txt","w")
hans_zimmer.write('REM This code will run multiple times on a single script but will change the element size in x and y each time. \n')
hans_zimmer.close()

# appends the commands for analysis and replacement of parameter for each run.
batch_file = open("D:\\Additional Thesis\\Batch mode scripts\\HandB\\Parameter_HandB.txt","a")

for i in range(len(KV)): #loops through each set of input parameters
    for j in range(len(SFS)): #loop for coefficient of hk based on input parameter. Loops through column index 1 in each row of SFS.
        for k in range(1,len(SFS[j])):#loop for coefficient of bk based on input parameter. Loops through each row of SFS.
            HK = SFS[j][0]*KV[i-1][6]
            BK = SFS[j][k]*HK
            batch_file.write('"C:\\Program Files\\ANSYS Inc\\v160\\ANSYS\\bin\\winx64\\ansys160.exe" -b -i Shell24.txt -'+IP
    [0][0]+' '+IP[1][0]+' -'+IP[0][1]+' '+IP[1][1]+' -'+IP[0][2]+' '+IP[1][2]+' -'+IP[0][3]+' '+IP[1][3]+' -'+IP[0][4]+' '+IP[1][4]+' -'+IP
    [0][5]+' '+IP[1][5]+' -'+IP[0][6]+' '+IP[1][6]+' -hk '+str(HK)+' -bk '+str(BK)+' -o out'+str(j+1)+str(k)+'.txt \n')#output file is stored
     as out(abcd), ab-HK,cd-BK loop.
batch_file.close()

''' This part changes the extension of the file to .bat '''
txt2bat = os.path.join("D:\\Additional Master Thesis\\Batch mode scripts\\LoadPatchSize",'Parameter_HandB' + '.txt')#changes .txt to
    .bat
base = os.path.splitext(txt2bat)[0]
os.rename(txt2bat,base + ".bat")
```

Listing C.3: Variation of $\frac{d}{h}$ and $\frac{h}{b}$ ratios

D

APPENDIX-D

The following code was used to conduct the variation of curvature and obtain the moment results to be used in curve fitting.

D.1. CODE FOR VARIATION OF CURVATURE

```
'''
Python Modules used.
'''
import math
import csv
import os

'''
Load all rows as strings from the input parameter file (.csv file) which are then saved as a list of lists IP which contains as many lists
      as there are rows.
 The data in the file is as follows:
tk/ Em / nu / Pk / dk
1 / 1.00E+05 / 0 / 100 / 4
where :
tk – thickness of shell (mm)
Em – Young's Modulus (N/mm2)
nu – Poisson's Ratio
Pk – Point Load applied in Mechanical Model (N)
dk – Diameter of Influence of Point Load in FE Model (mm)
'''


with open("H:\\Desktop\\Additional Master Thesis\\Batch mode scripts\\Curvature\\Input_Parameters_2.csv", "r") as f_input:
    csv_reader = csv.reader(f_input)
    IP = list (csv_reader)
print IP

'''
Load all rows as strings from the file containing values of the curvature (.csv file) which are then saved as a list of lists VL which
      contains as many lists as there are rows.
The values are :
kxx kyy
0.033333333   0.033333333 0.01   0.003333333 0.001   0.000333333 −0.000333333   −0.001 −0.003333333   −0.01 −0.033333333
0.01          0.033333333 0.01   0.003333333 0.001   0.000333333 −0.000333333   −0.001 −0.003333333   −0.01 −0.033333333
0.003333333   0.033333333 0.01   0.003333333 0.001   0.000333333 −0.000333333   −0.001 −0.003333333   −0.01 −0.033333333
0.001         0.033333333 0.01   0.003333333 0.001   0.000333333 −0.000333333   −0.001 −0.003333333   −0.01 −0.033333333
0.000333333   0.033333333 0.01   0.003333333 0.001   0.000333333 −0.000333333   −0.001 −0.003333333   −0.01 −0.033333333
−0.000333333  0.033333333 0.01   0.003333333 0.001   0.000333333 −0.000333333   −0.001 −0.003333333   −0.01 −0.033333333
−0.001        0.033333333 0.01   0.003333333 0.001   0.000333333 −0.000333333   −0.001 −0.003333333   −0.01 −0.033333333
−0.003333333  0.033333333 0.01   0.003333333 0.001   0.000333333 −0.000333333   −0.001 −0.003333333   −0.01 −0.033333333
−0.01         0.033333333 0.01   0.003333333 0.001   0.000333333 −0.000333333   −0.001 −0.003333333   −0.01 −0.033333333
−0.033333333  0.033333333 0.01   0.003333333 0.001   0.000333333 −0.000333333   −0.001 −0.003333333   −0.01 −0.033333333

The Data is to be read as follows:
```

```
Kxx values are in the column next to 'kxx' AND Kyy values are in the rows following a value of 'kxx'.

'''

with open("H:\\Desktop\\Additional Master Thesis\\Batch mode scripts\\Curvature\\Varying_Curvature.csv", "r") as Chameleon:
    csv_reader = csv.reader(Chameleon)
    VL = list (csv_reader)
print VL

KV=[] #empty list to store IPs as floats
SFS=[]#empty list to store VLs as floats

'''
Load all numerical values in IP and VL into seperate lists. KV will contain only the numerical values of the input parameters.
SFS will contain lists within a list which have the kxx and kyy values. A list within SFS is interpreted as follows:
The first value is the 'kxx' value to be used in a batch of analyses with the rest of the values being the 'kyy' values for that batch of
    analyses.

'''
for i in range(1,len(IP)):
    KV.append([float(x) for x in IP[i]])

for i in range(1,len(VL)):
    SFS.append([float(y) for y in VL[i]])


'''
Next code block creates the batch file (.bat) neede to run ANSYS in Batch mode.
It creates seperate lists KXX and KYY from which the values are then picked up
as inputs for each individual analysis in ANSYS. Each line represents one analysis.
An example line will look like:
"C:\Program Files\ANSYS Inc\v160\ANSYS\bin\winx64\ansys160.exe" -b -i Shell24_191016.txt -tk 1.0 -kyy 0.0333 -kxx
    0.0333333333333 -Ek 100000.0 -nu 0.0 -Pk 100.0 -dk 5.0 -o out0.txt
'''

l=1
for folderName, subfolders, filenames in os.walk('D:\\Additional Thesis\\Batch mode scripts\\Curvature'):
    for subfolder in subfolders:#loops over each d/t folder i.e ratio from 1 to 8. The rest of the code follows the curvature variation
        hans_zimmer = open(os.path.join(folderName,subfolder,'Parameter_Curvature_' + str(l) + '.txt'),"w")
        hans_zimmer.write('REM This code will run multiple times on a single script but will change the curvature in x and y each time.
    \n')#REM signifies beginning of a comment.
        hans_zimmer.close()
        batch_file = open(os.path.join(folderName,subfolder,'Parameter_Curvature_' + str(l) + '.txt'),"a")
        for i in range(len(KV)): #loops through each set of input parameters
            for j in range(len(SFS)): #loop for coefficient of kxx based on input parameter. Loops through column index 1 in each row
    of SFS.
                for k in range(1,len(SFS[j])):#loop for coefficient of kyy based on input parameter. Loops through each row of SFS.
                    KXX = SFS[j][0]
                    KYY = SFS[j][k]
                    batch_file.write('"C:\\Program Files\\ANSYS Inc\\v160\\ANSYS\\bin\\winx64\\ansys160.exe" -b -i
    Shell24_180716.txt -'+IP[0][0]+' '+IP[1][0]+' -'+IP[0][1]+' '+IP[1][1]+' -'+IP[0][2]+' '+IP[1][2]+' -'+IP[0][3]+' '+IP[1][3]+' -kxx '+str
    (KXX)+' -kyy '+str(KYY)+' -dk ' + str(l*int(IP[0][1]))+' -o out'+str(j+1)+str(k)+'.txt \n')#output file is stored as out(abcd), ab-NX,
    cd-NY loop.
        batch_file.close()
        txt2bat = os.path.join(folderName,subfolder,'Parameter_Curvature_' + str(l) + '.txt')#changes .txt to .bat
        base = os.path.splitext(txt2bat)[0]
        os.rename(txt2bat,base + ".bat")
        l=l+1
```

Listing D.1: Variation of curvature

E

APPENDIX-E

The following code was used to conduct the curve fitting in python. [14]

E.1. Code for Curve Fitting in Python

```
'''
Python Modules Used
'''
from scipy import optimize
import numpy as np
import openpyxl
import os
'''
This code block opens required excel file to obtain the moment results from ANSYS and the respective curvature values in x and y.
'''
os.chdir('H:\\Desktop\\Additional Master Thesis\\Batch mode scripts\\Optimization')
jb = openpyxl.load_workbook('Results_Curvature_Contours.xlsx') #open desired excel file.
nm = jb.get_sheet_names()
sg = []
for x in range(len(nm)):
    sg.append(str(nm[x]))
'''
The following code block creates empty lists for all the variables required.

'''
kxx = [] #empty lists to store values of curvature in x.
kyy = [] #empty lists to store values of curvature in y.
mxx=[]#empty list for mxx values.
p=[] #empty list to store optimized constants in the formula.
pcov=[] #empty list to store covariance values obtained during the optimization process.
dk =[] #empty list to store the values of the diameter of influence which is treated as a parameter.
for m in range(1,9):
    dk.extend([m]*100) #Creates an 800x1 list to consider dk as a variable.
for i in range(len(nm)):
    jz = jb.get_sheet_by_name(sg[i])
    for j in range(1,int(jz.get_highest_row())):
        kxx.append(jz.cell(row=j,column=0).value)
        kyy.append(jz.cell(row=j,column=1).value)
        mxx.append(jz.cell(row=j,column=13).value)
'''
Data will be optimized by reading the values as follows:
mxx = f(kxx, kyy, dk/tk) where
kxx: curvature in x.
kyy: curvatrue in y.
dk: diameter of influence of point load.
tk: thickness of shell = 1 for all calcualtions.
'''

def f(X,a,b,c,d,e):#g,h,m,n,o):
```

```
    kxx,kyy,dk = X
    return a*100*np.log(1/(((dk)**2)*(np.abs(b*kxx+c*kyy)+np.abs(d*kxx+e*kyy))))#+g+h*(kxx/kyy)+m*((kxx/kyy)**0.5)+n*(kxx/kyy)**o
    #a=0.039599475;b=0.016995719;c=0.141017581;d=-0.055995379;e=-0.022969291;# old initial guesses
a=0.04;b=0.044;c=0.084;d=0.202;e=0.028;

p,pcov = optimize.curve_fit(f,(kxx,kyy,dk),mxx,p0=[a,b,c,d,e])#g,h,m,n,o])#optimized paramters are stored as lists.
'''
This code block stores the data of the optimized constants in an excel file.
'''

wb = openpyxl.Workbook()
sheet = wb.active
sheet.title = 'Parameter Values'
for l in range(len(p)):
    sheet.cell(row=l,column=0).value=p[l] #writes row-wise value of abcde in respective order in 1 column respectively.

wb.save('Parameter Value_new_formula_2.xlsx')
```

Listing E.1: Curve Fitting in Python