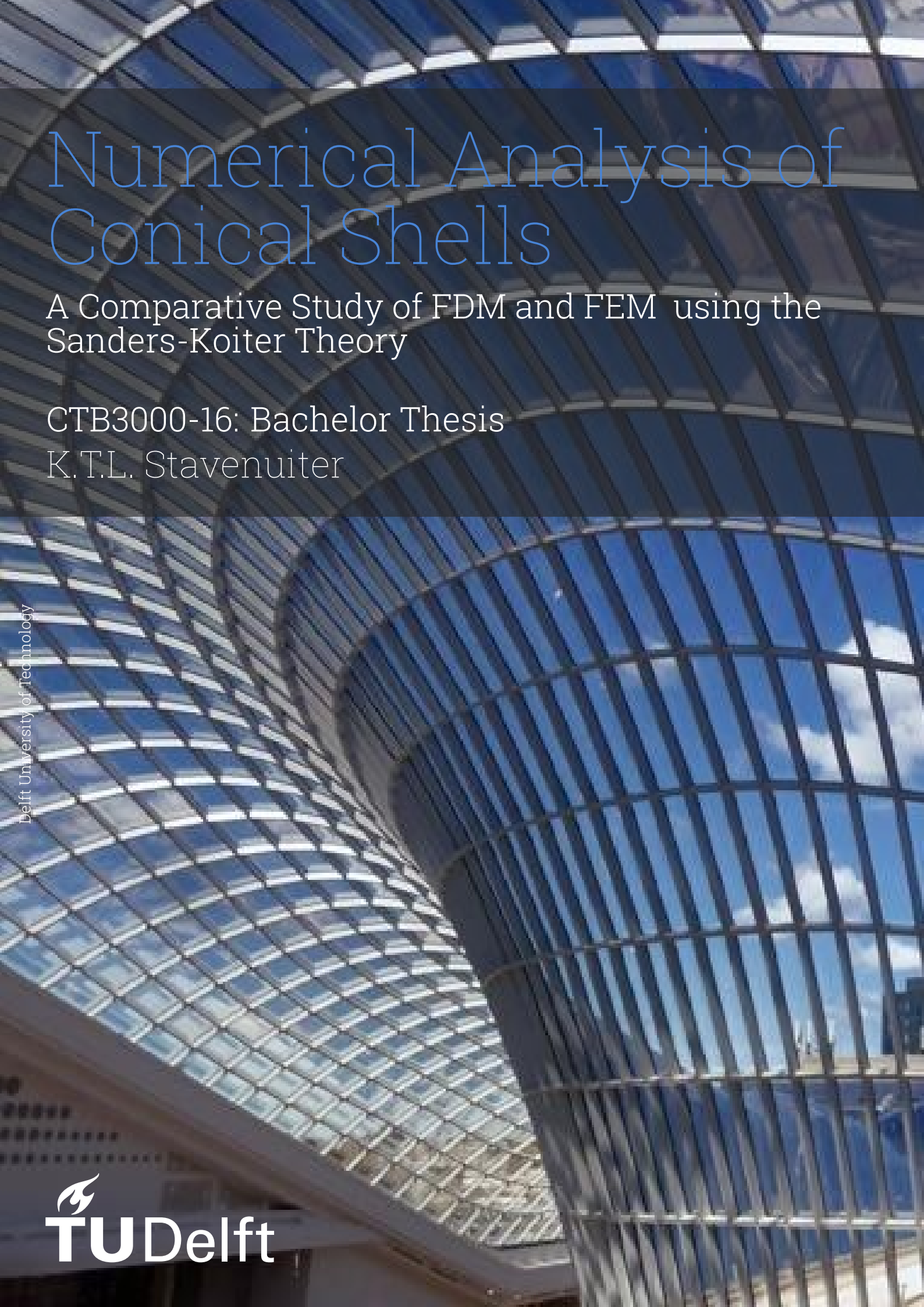


Numerical Analysis of Conical Shells

A Comparative Study of FDM and FEM using the Sanders-Koiter Theory

CTB3000-16: Bachelor Thesis

K.T.L. Stavenuiter

Delft University of Technology

TU Delft

Numerical Analysis of Conical Shells

A Comparative Study of FDM and FEM using the Sanders-Koiter Theory

by

K.T.L. Stavenuiter

Supervisor 1:	Dr. Ir. P.C.J. Hoogenboom
Supervisor 2:	Dr. Ir. Iuri B. C. M. Rocha
Project Duration:	Februari, 2026 - March, 2026
Faculty:	Faculty of Civil Engineering, Delft

Cover: issuu.com rahmanalikhan type of steel structures
Style: TU Delft Report Style, with modifications by Daan Zwaneveld

TU Delft

Preface

This thesis is submitted for the Bachelor of Science degree in Civil Engineering at Delft University of Technology. The research and writing of this project were carried out between February and March 2026.

Working on this project provided me with a valuable opportunity to dive deep into the mathematical complexities of shell mechanics and numerical modeling. I would like to express my gratitude to my supervisors, Dr. Rocha and Dr. P.C.J. Hoogenboom, for their guidance, critical feedback, and support throughout this process. A special thanks goes to Dr. Hoogenboom for providing the foundational Python framework, without which the development of the conical FDM model would not have been possible.

Finally, I would like to thank my family, friends, and fellow students for their continuous encouragement and valuable discussions during these intense weeks of programming and writing.

K.T.L. Stavenuiter
Delft, April 2026

Summary

The structural analysis of thin elastic shells is commonly done using the Finite Element Method (FEM), a method which is derived from a solid finite element with a well-proven reliability. An alternative approach is based on solving the governing differential equations from Sanders-Koiter shell theory and the Finite Difference Method (FDM). Both methods should describe the same physical reality for small displacements of thin elastic shells. The independence of both methods provides an opportunity for verifying one method with the other. However, there is no framework capable of calculating the deformations of thin elastic shells using the Sanders-Koiter equations that approaches the FEM.

The primary objective of this research is to verify the FEM by comparing it against the FDM. By comparing these two mathematically independent approaches, this study analyzes if FEM is the most optimal tool for engineers or that FDM provides a good alternative. Furthermore, this research seeks to find the best possible grid resolutions and element sizes required for both methods.

Prior to this study, a FDM script was available that could generate structural approximations for cylindrical canopies. In order to accurately model the geometry of a cone, the initial FDM script used for cylinders, is expanded to include a variable radius $R(x)$, physical circumferential curvature k_{yy}, and geodesic curvature k_y on a staggered grid. The next step is to test and validate the newly developed FDM model. To achieve this, the results were compared against a FEM model. The evaluation was divided into four distinct scenarios introducing different structural challenges in each scenario.

Validation across the four different load scenarios demonstrates that the FDM approaches the FEM model for symmetric, continuous surface loads, giving a result of at least 6% difference between FEM and FDM. In the scenario with a surface load covering the half of the canopy (Scenario D), the FDM framework also proves the decoupling of in-plane membrane shear forces ($n_{xy} \neq n_{yx}$) giving a difference of 10.9% between the two, something that is simplified in standard FEM packages. However, the validation also shows limitations of the current FDM model. The accuracy drastically decreases when subjected to more complicated conditions, like point loads, asymmetric loading or multiple supports. The main cause for this inaccuracy is suspected to lie in the implementation of the Python script. This results in the conclusion that the FDM model is not able to verify the FEM model yet.

Contents

Preface i

Summary ii

1 Introduction 1
- 1.1 Background . . . 1
- 1.2 Problem definition . . . 1
- 1.3 Research Questions . . . 2
- 1.4 Research Approach . . . 2

2 Theoretical Framework 3
- 2.1 Sanders-Koiter Shell Theory . . . 3
  - 2.1.1 Kinematic Equations . . . 3
  - 2.1.2 Constitutive Equations . . . 3
  - 2.1.3 Equilibrium Equations . . . 4
- 2.2 Numerical Solution: Finite Difference Method (FDM) . . . 4
- 2.3 Finite Element Method (FEM) . . . 5

3 Methodology 6
- 3.1 Geometrical Extension to a Conical Shell . . . 6
- 3.2 Spatial Discretization . . . 7
  - 3.2.1 The Staggered Grid . . . 7
  - 3.2.2 Interpolation on the Staggered Grid . . . 8
- 3.3 Boundary Conditions and Ficticious Nodes . . . 8
  - 3.3.1 Formulation of Boundary Conditions . . . 8
  - 3.3.2 Boundary Treatment Using Fictitious Nodes . . . 9
- 3.4 Test Scenarios and Load Cases . . . 9

4 Results 13
- 4.1 Results Scenario A: Single Clamped Edge with Point Load . . . 13
- 4.2 Results Scenario B: Single Clamped Edge with Surface Load . . . 15
  - 4.2.1 Grid optimization . . . 15
- 4.3 Results Scenario C: Clamped-Hinged Edges under Distributed Surface Loading . . . 17
- 4.4 Results Scenario D: Evaluation of Membrane Shear Forces (n_{xy} and n_{yx}) . . . 19

5 Discussion 21
- 5.1 Interpretation of Results and Model Robustness . . . 21
- 5.2 Data Reliability . . . 21
- 5.3 Limitations of the Current FDM Framework . . . 22
- 5.4 Usability in Practice . . . 22

6 Conclusion and Outlook 23
- 6.1 Summary of Findings . . . 23
- 6.2 Future Work . . . 24

References 25

A Source Code Example 26
- A.1 Main Python Code . . . 26
- A.2 Scenario B adjustments . . . 32
- A.3 Scenario C adjustments . . . 33
- A.4 Scenario D adjustments . . . 34

B Contour plots scenario D 36

1

Introduction

1.1. Background

Shell structures have played a significant and highly visible role in the civil, architectural, and aerospace industries for many years. Because of their curved geometry, thin-walled shells carry applied loads primarily through in-plane membrane forces rather than bending moments, resulting in highly efficient structures with an exceptional strength-to-weight ratio (Billington, 1982; Calladine, 1983).

The foundation for the structural analysis of shells was laid by Love's first approximation in the late 19th century (Timoshenko & Woinowsky-Krieger, 1959). Because early theories contained mathematical inconsistencies regarding rigid-body rotations, the framework was refined in the mid-20th century by the theories of Sanders and Koiter (Koiter, 1960; Sanders Jr, 1959). Their consistent first-approximation theory accurately describes the complex interactions between membrane stresses and bending moments in thin elastic shells.

Because analytical solutions to these differential equations are only available for highly simplified cases, numerical approximation methods are required. In the 1970s, the Finite Difference Method (FDM) emerged as a highly efficient computational approach, used by organizations like NASA for calculating stresses in symmetric thin shells (Noor & Stephens, 1973). In recent decades, however, FDM has been largely superseded in engineering practice by the Finite Element Method (FEM) (Zienkiewicz & Taylor, 2005).

1.2. Problem definition

Currently, the Finite Element Method is the standard in structural engineering. Due to its capacity to handle complicated geometries and asymmetric loading conditions, commercial FEM software dominates the industry. While FDM is not expected to surpass FEM in general engineering practice, it is highly valuable to investigate whether FDM can serve as a viable, independent mathematical alternative for specific continuous shell problems. Because the two methods have no direct mathematical correlation, exploring the limits of FDM helps determine if it could complement FEM as a theoretical verification tool.

Despite this academic potential, there is currently no FDM framework capable of computing the displacements and internal stresses of conical canopies using the complete Sanders-Koiter theory. An existing foundational Python script, developed by Dr. P.C.J. Hoogenboom (2023), provides calculation of internal forces for cylindrical canopies. While this script calculates the general topological distribution of these forces, the specific numerical values do not yet accurately align with FEM reference models. Furthermore, this code cannot handle geometries other than perfect cylinders. Transitioning from a perfect cylinder to a cone introduces a linearly decreasing radius and adds terms in the equations. Therefore, the core problem lies in determining whether the FDM model can be successfully expanded to show the true deformations of a conical shell.

1.3. Research Questions

The primary objective of this thesis is to check the Finite Element Method with the Finite Different method using the Sanders-Koiter equations. To achieve this objective, the research is structured around the following four sub-questions:

1. *How does the geometry of a conical shell change the Sanders-Koiter equations compared to a standard cylinder?*
2. *To what extent can the FDM model accurately predict the deformations of a canopy under continuous and asymmetric load configurations when compared to FEM?*
3. *What is the numerical impact of local disturbances, such as singular point loads and different boundary conditions, on the accuracy of the FDM model?*
4. *What grid- and element size are needed for a reliable result for FDM and FEM respectively?*

1.4. Research Approach

In order to answer the formulated research questions, the methodology is made by analyzing the equations of the Sanders-Koiter shell theory. Here, the differences between FDM and FEM are described as well. With this knowledge, the existing FDM-model is expanded to describe the geometry of a conical shell. In order to test the results and accuracy of the model, four different scenarios are programmed in the FDM model and compared to the FEM model. Finally, the numerical results are critically analyzed to discuss the interpretation of the data and show the possible limitations of the FDM model. Ultemately, based on these findings, a final conclusion is drawn on whether it is a viable method to verify FEM using FDM.

2

Theoretical Framework

2.1. Sanders-Koiter Shell Theory

To mathematically describe the internal forces and deformations of the canopy including bending and stretching, the consistent first-approximation shell theory is used (Koiter, 1960; Sanders Jr, 1959). The theory consists of kinematic, constitutive, and equilibrium equations. The Sanders-Koiter equations are described by a total of 21 equations and are valid for small displacements for elastic material behavior.

2.1.1. Kinematic Equations

The first set of equations defines the kinematics of the shell. These relations link the displacements of the middle-surface (u_x, u_y, u_z) to the resulting membrane strains (ε, γ) and the changes in curvature and twist (κ, ρ, φ). The complete kinematic fields are described by the following nine equations (P. Hoogenboom, 2024):

$$\varepsilon_{xx} = \frac{\partial u_x}{\partial x} - k_{xx}u_z + k_x u_y \tag{2.1a}$$

$$\varepsilon_{yy} = \frac{\partial u_y}{\partial y} - k_{yy}u_z + k_y u_x \tag{2.1b}$$

$$\gamma_{xy} = \frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} - 2k_{xy}u_z - k_x u_x - k_y u_y \tag{2.1c}$$

$$\varphi_x = -\frac{\partial u_z}{\partial x} - k_{xx}u_x - k_{xy}u_y \tag{2.1d}$$

$$\varphi_y = -\frac{\partial u_z}{\partial y} - k_{yy}u_y - k_{xy}u_x \tag{2.1e}$$

$$\varphi_z = \frac{1}{2}\left(-\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} - k_x u_x + k_y u_y\right) \tag{2.1f}$$

$$\kappa_{xx} = \frac{\partial \varphi_x}{\partial x} - k_{xy}\varphi_z + k_x\varphi_y \tag{2.1g}$$

$$\kappa_{yy} = \frac{\partial \varphi_y}{\partial y} + k_{xy}\varphi_z + k_y\varphi_x \tag{2.1h}$$

$$\rho_{xy} = \frac{\partial \varphi_x}{\partial y} + \frac{\partial \varphi_y}{\partial x} + (k_{xx} - k_{yy})\varphi_z - k_x\varphi_x - k_y\varphi_y \tag{2.1i}$$

2.1.2. Constitutive Equations

Once the strain and curvature components are established, they are used for calculating internal stress resultants. These constitutive equations relate the membrane forces (n) and bending moments (m) to the kinematic parameters defined in Equations 2.1, with the Young's modulus E, Poisson's ratio ν, and the shell thickness t. What is special about the constitutive equations, is that the in plane shear forces are not equal ($n_{xy} \neq n_{yx}$) (P. Hoogenboom, 2024):

$$n_{xx} = \frac{Et}{1-\nu^2}(\varepsilon_{xx} + \nu\varepsilon_{yy}) \tag{2.2a}$$

$$n_{yy} = \frac{Et}{1-\nu^2}(\varepsilon_{yy} + \nu\varepsilon_{xx}) \tag{2.2b}$$

$$\frac{n_{xy} + n_{yx}}{2} = \frac{Et}{2(1+\nu)}\gamma_{xy} \tag{2.2c}$$

$$m_{xx} = \frac{Et^3}{12(1-\nu^2)}(\kappa_{xx} + \nu\kappa_{yy}) \tag{2.2d}$$

$$m_{yy} = \frac{Et^3}{12(1-\nu^2)}(\kappa_{yy} + \nu\kappa_{xx}) \tag{2.2e}$$

$$m_{xy} = \frac{Et^3}{24(1+\nu)}\rho_{xy} \tag{2.2f}$$

2.1.3. Equilibrium Equations

Finally, the internal forces and moments must satisfy the conditions of static equilibrium alongside the externally applied loads (p_x, p_y, p_z). The force and moment are calculated for an infinitely small shell element, and results in the following six differential equations (P. Hoogenboom, 2024):

$$v_x = \frac{\partial m_{xx}}{\partial x} + \frac{\partial m_{xy}}{\partial y} + k_y(m_{xx} - m_{yy}) + 2k_x m_{xy} \tag{2.3a}$$

$$v_y = \frac{\partial m_{yy}}{\partial y} + \frac{\partial m_{xy}}{\partial x} + k_x(m_{yy} - m_{xx}) + 2k_y m_{xy} \tag{2.3b}$$

$$n_{xy} - n_{yx} = -k_{xy}(m_{xx} - m_{yy}) + (k_{xx} - k_{yy})m_{xy} \tag{2.3c}$$

$$p_x = -\frac{\partial n_{xx}}{\partial x} - \frac{\partial n_{yx}}{\partial y} - k_y(n_{xx} - n_{yy}) - k_x(n_{xy} + n_{yx}) + k_{xx}v_x + k_{xy}v_y \tag{2.3d}$$

$$p_y = -\frac{\partial n_{yy}}{\partial y} - \frac{\partial n_{xy}}{\partial x} - k_x(n_{yy} - n_{xx}) - k_y(n_{xy} + n_{yx}) + k_{yy}v_y + k_{xy}v_x \tag{2.3e}$$

$$p_z = -k_{xx}n_{xx} - k_{xy}(n_{xy} + n_{yx}) - k_{yy}n_{yy} - \frac{\partial v_x}{\partial x} - \frac{\partial v_y}{\partial y} - k_y v_x - k_x v_y \tag{2.3f}$$

These three coupled sets of differential equations form the complete mathematical foundation required to solve for the displacement and stress fields of the shell structure. The equilibrium equations are solved for displacements and rotations later this report, where the other equations will provide the link between stresses and displacements.

2.2. Numerical Solution: Finite Difference Method (FDM)

Because the differential equations of the Sanders-Koiter theory are hard to solve analytically for practical boundary conditions, a numerical approach is required. The Finite Difference Method (FDM) is a widely used technique that is approximated by converting differential equations into a system of algebraic equations (LeVeque, 2007). FDM can be derived from Taylor series expansions: Consider a continuous function $f(x)$, which is divided over a grid, with a step size Δx. The value of the neighboring point can be expressed by the Taylor series (Chapra & Canale, 2015):

$$f(x + \Delta x) = f(x) + \Delta x f'(x) + \frac{\Delta x^2}{2!}f''(x) + \frac{\Delta x^3}{3!}f'''(x) + \mathcal{O}(\Delta x^4) \tag{2.4}$$

Equation 6 can be rearranged by isolating $f'(x)$, the remaining higher-order terms are moved to the other side. Because the dominant term in this remaining infinite series scales linearly with the step size, the entire remainder is grouped and represented by the first-order truncation error $\mathcal{O}(\Delta x)$. This results in the standard *forward difference approximation*:

$$f'(x) = \frac{f(x + \Delta x) - f(x)}{\Delta x} - \mathcal{O}(\Delta x) \tag{2.5}$$

The term $\mathcal{O}(\Delta x)$ represents the truncation error, indicating that the accuracy of the forward difference method scales linearly with the step size. For getting a reliable result, which is much needed, a higher accuracy is strictly required. This is achieved by evaluating both the forward step ($x + \Delta x$) and the backward step ($x - \Delta x$) Taylor expansions. Subtracting the backward expansion from the forward expansion eliminates the second-order derivative terms entirely, finding the *central difference* approximation:

$$f'(x) = \frac{f(x + \Delta x) - f(x - \Delta x)}{2\Delta x} - \mathcal{O}(\Delta x^2) \tag{2.6}$$

The central difference method is more accurate because its truncation error is $\mathcal{O}(\Delta x^2)$. This quadratic relationship implies that multiplying the grid spacing by 10, Δx reduces the approximation error by a factor of 100. By applying these central difference approximations, the continuous function is transformed into a discrete grid, forming a large system of linear equations that determines the variables at every grid point.

2.3. Finite Element Method (FEM)

Instead of approximating exact mathematical derivatives at specific grid points like with FDM, in FEM the continuous structure is discretized into small, volumetric or planar elements (Zienkiewicz & Taylor, 2005). The mechanical behavior within each individual element is approximated using shape functions, after which all elements are assembled to satisfy global equilibrium. For standard three-dimensional shell structures in commercial software, each node typically has six degrees of freedom (DOFs): Three translations (u_x, u_y, u_z) and three rotations ($\varphi_x, \varphi_y, \varphi_z$). The mechanical behavior within each individual element is approximated using shape functions based on these discrete DOF's.

The Sanders-Koiter theory relies on thin-shell Kirchoff-Love assumptions and ignores out-of-plane shear transformation. While normal FEM packages often rely on the Mindlin-Reissner theory which does include this shear deformation, in this report the SCIA Engineer program is used, with the Kirchhoff-Love assumptions to ensure a fair comparison. However, the FEM formulation does not solve the same Sanders-Koiter equations used in the FDM model. While the Kirchhoff-Love assumptions succesfully eliminates out-of-plane deformation in FEM, SCIA's approximation still fundamentally differs from the FDM approximation using the Sanders-Koiter equation. Furthermore, the exact derivation of the SCIA shell elements is not available. Despite these different mathematical methods, both are designed to approximate the exact same reality. Therefore both the FDM and FEM approaches should converge towards the same results.

3

Methodology

This chapter outlines the research methodology employed to validate the numerical Finite Difference Method (FDM) script against the commercial Finite Element Method (FEM) software, SCIA Engineer. Due to the complexities of shell mechanics, this section details the geometrical expansion to a conical shell, the spatial discretization using a staggered grid, and the selected test scenarios.

The numerical approximation and the assembly of the global system matrix ($M \cdot u = f$) are executed using a Python code. The script used for this procedure is developed and provided by Dr. (P. C. J. Hoogenboom, 2023). This code contains the fundamental algorithms for establishing the grid logic, setting the system matrix, and using a linear solver to compute the system of equations for a cylindrical canopy. The following parameters are chosen for the simulation, based on a realistic canopy:

Table 3.1: Parameters used in Python FDM-simulation.

Parameter	Value	Unit
Begin Radius(r_b)	1.5	m
End Radius(r_s)	0.25	m
Cantiliver length (l)	6	m
Canopy thickness (t)	0.06	m
Modulus of elasticity (E)	1×10^7	kN/m^2
Poisson-factor (ν)	0.15	-

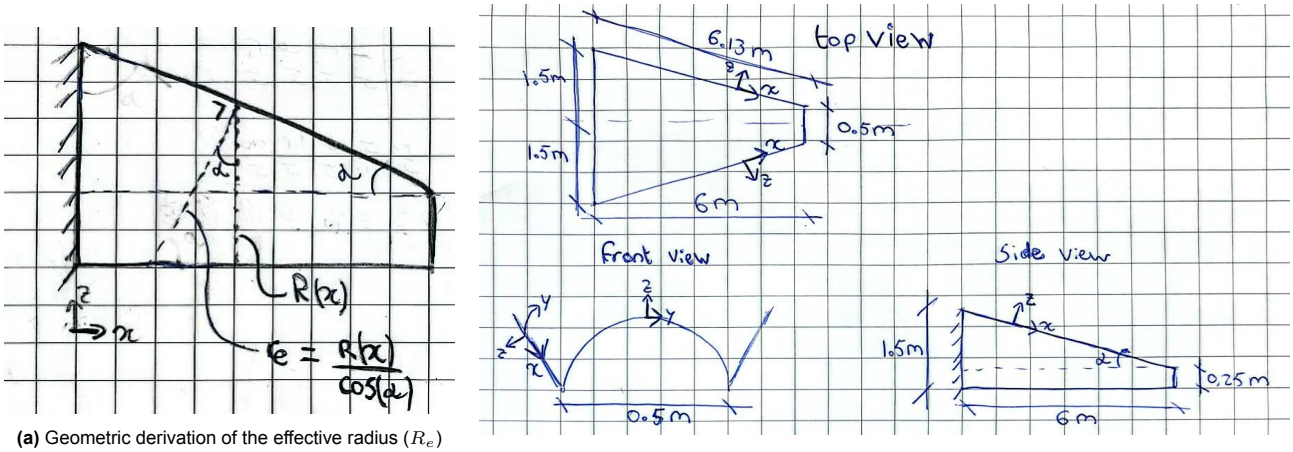
3.1. Geometrical Extension to a Conical Shell

The initial FDM model, based on a perfect cylinder, is expanded to represent a conical canopy. While a cylinder has a constant radius and a constant circumferential curvature ($k_{yy} = 1/R$), the geometry of a cone continuously varies along its longitudinal axis (x).

To correctly apply the Sanders-Koiter theory to this geometry, several fundamental adjustments are implemented in the Python model:

- **Variable Radius:** The radius $R(x)$ decreases linearly from the support to the free tip. The variable radius is an array with $mm - 1$ equal steps. The angle with horizontal is defined as α.
- **Physical Curvature (k_{yy}):** The Physical curvature is equal to $1/r_e$, with r_e as the effective radius. The effective radius of a cone does not intersect the central axis perpendicularly, see also figure 3.1a. Consequently, the effective physical curvature is mathematically defined as $k_{yy} = -\cos(\alpha)/R(x)$. Because the radius is not constant, the k_{yy} changes over the length x. To prevent a division by 0, radius at the tip of the cone can be infinitely close to zero, but not equal to zero.
- **Geodesic Curvature (k_y):** Because the radius changes along the x-axis, the local coordinate system distorts. This distortion is introduced as the geodesic curvature $k_y = -\sin(\alpha)/R(x)$. This parameter is critical for the correct distribution of in-plane membrane forces. For the cylinder, $k_y = -\sin(0)/R(x) = 0$, which means this is a new term in the python code.

The introduction of this geometry with a varying radius $R(x)$ and a non-zero geodesic curvature k_y mathematically activates these terms that were previously equal to 0. For instance, the equilibrium equations now



(a) Geometric derivation of the effective radius (R_e) and circumferential curvature (k_{yy}).

(b) Geometry of conical canopy (not to scale).

Figure 3.1

feature additional membrane force-curvature interactions, and the kinematic equations for strains and rotations become more coupled. Furthermore, because k_{yy} now varies along the x-axis, it can no longer be factored out as a global constant; it must be locally evaluated and interpolated at every grid node.

The implementation of the geometrical parameters in the Python script is shown in Listing 3.1, the full Python code can be found in Appendix A.

Listing 3.1: Geometrical setup and curvature definition for the conical shell in FDM.

```
# --- GEOMETRY AND CURVATURES ---
x_length = 6    # Cantilevered length
r_start = 1.5   # Radius at the support
r_end   = 0.25  # Radius at the free tip
l  = np.sqrt(x_length**2 + (r_start-r_end)**2) # length x-axis (slanted)
t  = 0.060  # m     Canopy thickness
F  = 100.0  # kN    Load

# Grid settings
mm = 125  # Grid points
nn = 125    # Grid points

Radius = np.linspace(r_start, r_end, mm)
alpha = np.arctan((r_start-r_end) / x_length)

k_yy = np.zeros(mm)
k_y  = np.zeros(mm)

for i in range(mm):
    if Radius[i] > 0.0001:
        #physical curvature (circumferential)
        k_yy[i] = -np.cos(alpha) / Radius[i]
        #Geodesic curvature (flaring effect)
        k_y[i]  = -np.sin(alpha) / Radius[i]
```

3.2. Spatial Discretization

3.2.1. The Staggered Grid

As discussed before, the continuous middle-surface of the shell is discretized into a finite number of segments with an equal step size Δh. To translate the continuous differential equations into a solvable matrix structure and prevent numerical instabilities, a *staggered grid* layout is employed rather than a traditional collocated grid.

In a collocated grid (where displacements, forces, and moments are all calculated at the exact same node), numerical models of shell mechanics often suffer from oscillations without physical equilibrium. To prevent these instabilities, the staggered grid physically shifts the calculation points of different variables, dividing the numerical mesh into two distinct sets of nodes:

- **Main Nodes (i):** The primary kinematic variables, specifically the displacements (u_x, u_y, u_z), are defined and evaluated at the primary grid intersections.
- **Intermediate Nodes ($i + 1/2$):** The dynamic and constitutive variables, such as the membrane forces

(n_{xx}, n_{yy}), bending moments (m_{xx}, m_{yy}), and strains ($\varepsilon_{xx}, \kappa_{xx}$), are evaluated exactly halfway between the main nodes (in the center of the grid cells).

The reason for this separation lies in both the definition of strain and the mathematical accuracy of central difference schemes. Physically, strain is defined as the deformation between two adjacent points. Therefore, if displacements are defined at main nodes i and $i+1$, the resulting elongation naturally exists in the space directly between them.

Mathematically, calculating this strain requires the first spatial derivative of the displacement (e.g., $\partial u_x/\partial x$). The central difference approximation is most accurate when evaluated exactly halfway between the known data points: $(u_{x,i+1} - u_{x,i})/\Delta x$. This yields a result that is mathematically valid at the intermediate node $i+1/2$. If a traditional collocated grid were used, computing the strain at node i with a central difference scheme would require using the adjacent nodes $i-1$ and $i+1$, skipping node i entirely. By evaluating the derivatives at the intermediate nodes, the staggered grid forces a direct relationship between adjacent displacement nodes, setting physical equilibrium.

3.2.2. Interpolation on the Staggered Grid

Because the structural variables (displacements, strains, and internal forces) are physically separated on the staggered grid, computing the variable curvature coupling terms (k_y and k_{yy}) and mixed derivatives requires spatial interpolation. Depending on the relative location of a required variable compared to the governing equation's central node, specific averaging techniques are systematically applied in the matrix assembly:

- **1D Interpolation (2-Point Average):** When a variable needs to be shifted along a single grid axis (either longitudinally along x or circumferentially along y) to align with the equilibrium equation, a linear average of the two adjacent nodes is utilized. This results in a scaling factor of 0.5 for each node in the sparse matrix.
- **2D Interpolation (4-Point Average):** For coupling terms such as the interaction with the torsional moment (m_{xy}), variables need to be evaluated diagonally across the grid cell. In these cases, a bi-linear interpolation using the four surrounding nodes is required, distributing the variable with a weighting factor of 0.25 per node.

By consistently applying these interpolation rules, all physical parameters are evaluated at the exact mathematical center of their corresponding equations. This preserves the second-order accuracy of the central difference scheme and prevents numerical drift across the complex conical geometry. A schematic visualization of the staggered grid can be found in figure 3.2.

3.3. Boundary Conditions and Ficticious Nodes

3.3.1. Formulation of Boundary Conditions

The canopy is subjected to specific physical boundary conditions at its edges, which must be set in the numerical grid. For a typical canopy structure, two primary boundaries are defined: the edges at the support and the free edges at the other borders.

At the **clamped boundaries** (the fixed support), the physical constraint prevents any translation or rotation. In the numerical matrix, this is enforced by setting the displacement variables to zero. Additionally, the rotation of the meridian ϕ_x must be constrained, ensuring the shell remains perfectly perpendicular to the supporting wall.:

$$u_x = 0, \quad u_y = 0, \quad u_z = 0, \quad \varphi_x = 0 \tag{3.1}$$

At the **the hinged support**, the physical constraint prevents any translation, but allows the edge of the shell to rotate freely. In the numerical matrix, this is enforced by setting the displacement variables to zero, identical to the clamped condition. However, because the connection acts as a hinge, it cannot provide rotational stiffness or transfer bending moments to the supporting structure. This is mathematically enforced by setting the meridional bending moment to zero, rather than constraining the rotation ($\varphi_x = 0$):

$$u_x = 0, \quad u_y = 0, \quad u_z = 0, \quad m_{xx} = 0 \tag{3.2}$$

At the **free curved boundary**, the shell is completely unsupported, meaning it cannot transfer any forces or moments to an adjacent structure. This is mathematically enforced by setting the internal stress resultants at the final boundary nodes to zero:

$$n_{xx} = 0, \quad n_{xy} = 0, \quad m_{xx} = 0, \quad v_x = 0 \tag{3.3}$$

Along the **longitudinal free boundaries** (the side edges of the canopy), the shell is also completely unsupported. Because the normal vector to these side boundaries aligns with the circumferential y-direction, the physical absence of forces and moments must be defined with respect to the y-axis. Mathematically, this is enforced by setting the corresponding internal stress resultants at these edge nodes to zero:

$$n_{yy} = 0, \quad n_{yx} = 0, \quad m_{yy} = 0, \quad v_y = 0 \tag{3.4}$$

Note that the boundary conditions for the free edges are an approximation. A free edge normally requires the *effective* transverse shear force to be zero, which combines the transverse shear and the spatial derivative of the twisting moment: $v_y = \frac{\partial V_y}{\partial x}$ (P. Hoogenboom, 2024). However, in this FDM framework, this minimal value is neglected for simplicity. By applying these boundary equations at the absolute ends of the staggered grid, the global system of equations is closed and becomes uniquely solvable.

3.3.2. Boundary Treatment Using Fictitious Nodes

In the Finite Difference Method, the use of central difference approximations presents a mathematical challenge at the physical boundaries of the shell. Evaluating a spatial derivative, such as the meridional rotation ($\phi_x \approx -\partial u_z / \partial x$), directly at a boundary node requires data from adjacent nodes on both sides. However, at the extreme edges of the conical canopy, one of these required adjacent nodes falls outside the physical domain.

To resolve this issue while maintaining the second-order accuracy of the central difference scheme, the model employs *fictitious nodes*. These are added grid points located exactly one spatial step size outside the physical boundary.

A common intuition is that a zero-rotation boundary condition (e.g., a clamped edge where $\phi_x = 0$) could be contained by simply giving the boundary node itself a value of 0. However, this conflicts the value of the function (the physical displacement) with its gradient (the rotation). In a central difference scheme, the first spatial derivative at a physical boundary node i is calculated using its adjacent neighbors, for example:

$$\left(\frac{\partial u_z}{\partial x}\right)_i \approx \frac{u_{z,i+1} - u_{z,i-1}}{2\Delta x} \tag{3.5}$$

Crucially, the displacement of the boundary node itself ($u_{z,i}$) does not appear in this central difference equation. Therefore, explicitly setting the boundary node to zero ($u_{z,i} = 0$) successfully prevents physical translation, but it exerts no mathematical control over the slope passing through that node.

To enforce a horizontal slope ($\phi_x = 0$) at the boundary, the numerator of the central difference equation must evaluate to zero, which mathematically dictates that $u_{z,i-1} = u_{z,i+1}$. By governing the fictitious node ($i - 1$) to mirror the exact out-of-plane displacement of the first internal node ($i + 1$), the interpolated derivative directly at the boundary is forced to zero. This fictitious node technique allows the governing differential equations and central difference formulas to be applied uniformly across the entire numerical grid, accurately simulating a clamped support without altering the fundamental matrix structure. A visualization of the grid with fictitious nodes can be seen in figure 3.2

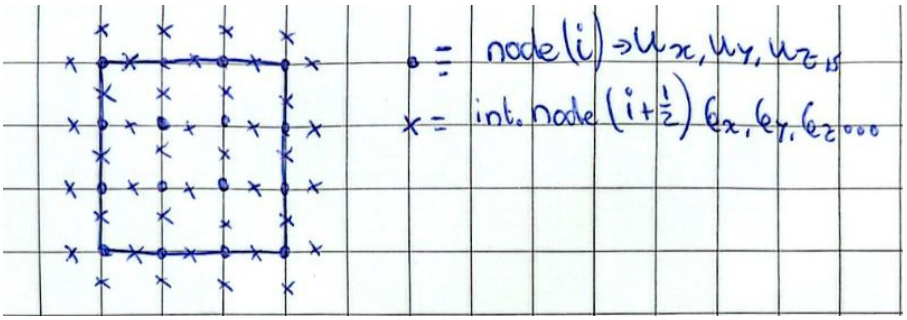


Figure 3.2: Schematic visualization the grid, the points outside the shell are eliminated by boundary conditions.

3.4. Test Scenarios and Load Cases

To validate the consistency of the FDM model, an analysis is done using multiple scenarios. Because the shell equations (formula 3, 4 and 5) remain unchanged, transitioning between these load cases and support conditions in the FDM model only requires modifying the specific boundary node equations and the external load vector. The following scenarios are evaluated:

1. **Scenario A: Single Clamped Edge with a Point Load.** In this baseline scenario, one edge of the canopy is fully clamped (representing a rigid wall connection), while a localized, downward point load is applied in the free corner ($x = L$ and $\theta = \pi$). In the FDM framework, this point is mathematically introduced by modifying the boundary condition at the corner node (e.g., equating n_{yy} to the applied concentrated load, rather than zero). The python code related to the boundary conditions related to this situation is provided in Appendix A.

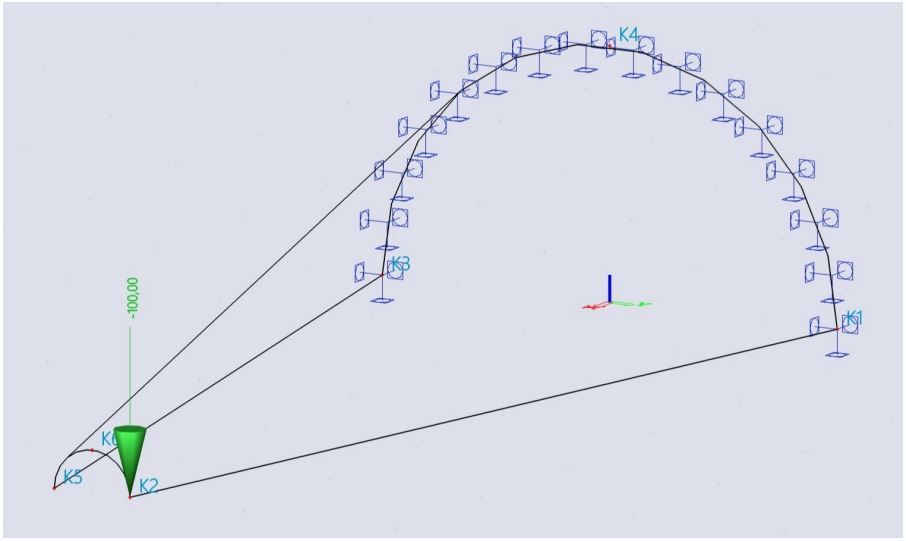


Figure 3.3: Schematic visualization of Scenario A: a point load ($F = 100kN$) applied on the corner of the canopy.

2. **Scenario B: Single Clamped Edge with a Distributed Surface Load.** The support conditions remain identical to Scenario A (a single clamped edge), but the concentrated point load is replaced by a uniformly distributed vertical surface load, representing self-weight or snow accumulation. The distributed load needs to be resolved into local components first, see figure 3.5. The python code related to the boundary conditions related to this situation is provided in Appendix A.2.

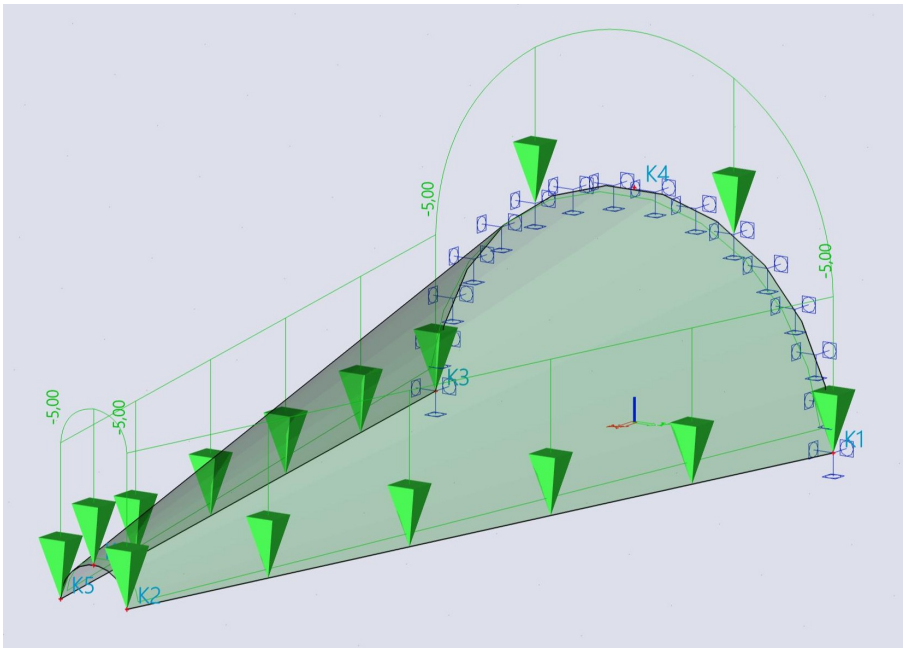


Figure 3.4: Schematic visualization of Scenario B: a surface load ($F = 5kN/m$) applied on the entire surface of the canopy.

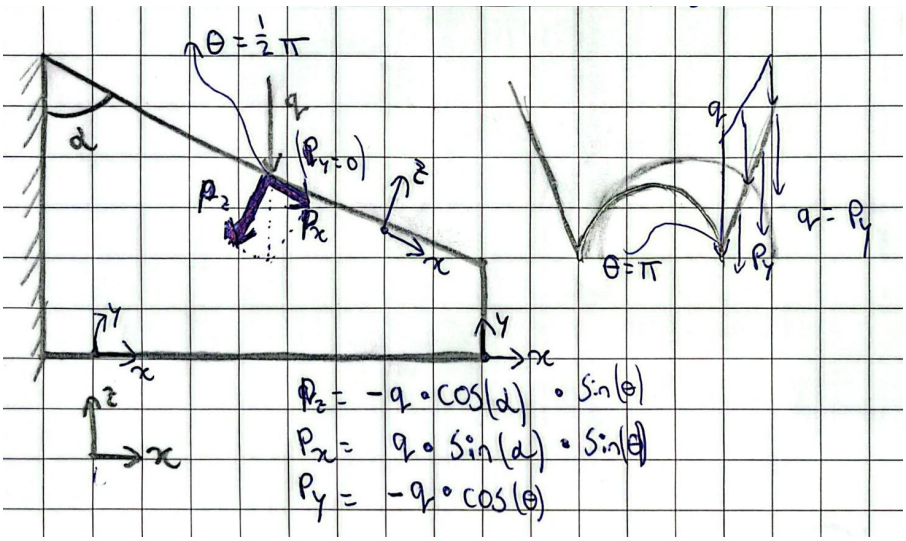


Figure 3.5: Schematic visualization of resolving the applied surface load q.

3. **Scenario C: Clamped-Hinged Edges under Distributed Surface Loading.** In this scenario, the shell is fully clamped at one boundary and hinged at the opposing boundary. This prevents any translation at both edges but allows rotation at the hinged side under a uniform surface load. The Python code defining these adjustments to the main Python code is provided in Appendix A.3.

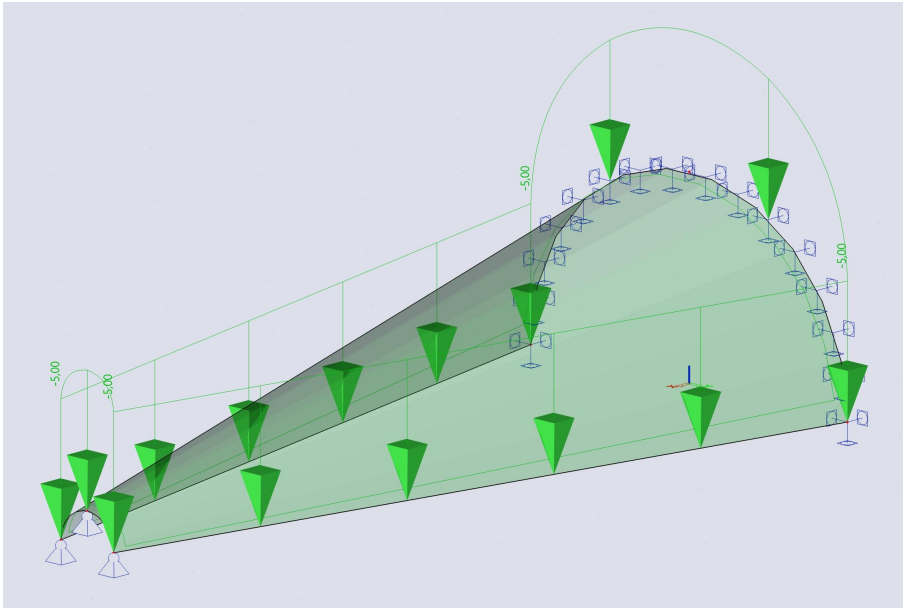


Figure 3.6: Schematic visualization of Scenario C: a surface load ($F = 5kN/m^2$), supported by two clamped edges.

4. **Scenario D: Asymmetrical Loading for Membrane Shear Inequality ($n_{xy} \neq n_{yx}$).**

   In this scenario, an asymmetrical surface load is applied to only one half edge of the canopy ($\theta \in [\frac{1}{2}\pi, \pi]$), combined with a single clamped edge identical to Scenario A and B. This specific scenario is designed to explicitly demonstrate that $n_{xy} \neq n_{yx}$ and maximize this inequality. The Python code defining these specific boundary conditions is provided in Appendix A.4.

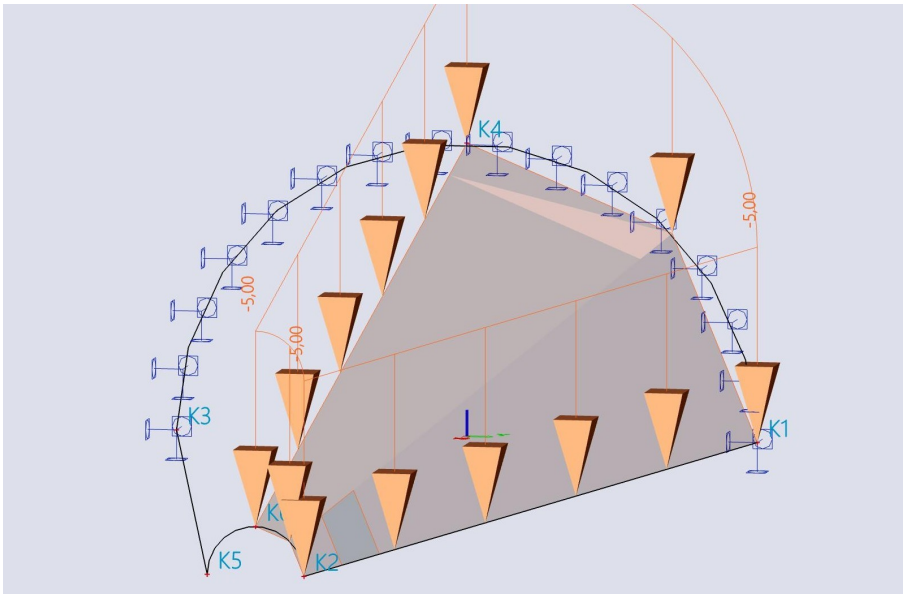


Figure 3.7: Schematic visualization of Scenario D: an asymmetric surface load ($F = 5kN/m^2$), supported by one clamped edge.

4

Results

This chapter presents the numerical results of the developed Finite Difference Method (FDM) Python model. To verify the accuracy and stability of the programmed FDM code, the outputs are compared against a Finite Element Method (FEM) model constructed in SCIA Engineer. The structural behavior of the conical canopy is analyzed across the four predefined load cases (Scenario A through D).

To evaluate the performance of the FDM matrix, the comparison focuses on structural variables governing shell mechanics. These include the out-of-plane normal displacement (u_z), the circumferential displacement (u_y), the internal twisting moment (m_{xy}), and the in-plane membrane forces (n_{xy} and n_{yx}).

The validation is performed by extracting the peak values of these parameters from both models for each scenario. Furthermore, when the peak value is a singularity another point will be evaluated, disregarding the singularity. Subsequently, the error margin between the FDM predictions and the SCIA baseline is calculated. This methodology not only shows the global accuracy of the FDM model, but also highlights local deviations caused by differences between the applied shell theories (e.g., concentrated load singularities and shear deformation effects).

4.1. Results Scenario A: Single Clamped Edge with Point Load

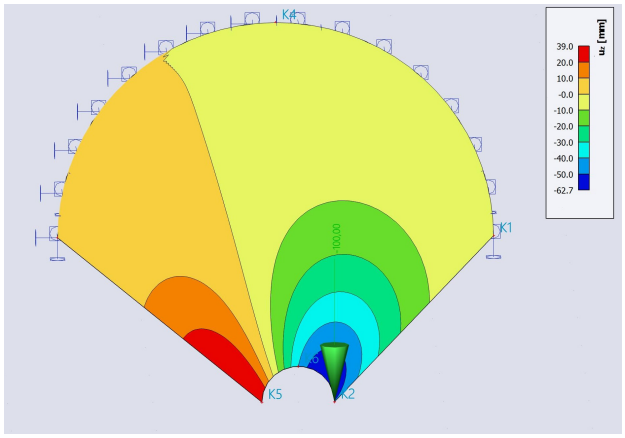
The results of the FEM and FDM models of scenario A are shown in figure 4.1. Visually, the contour plots do not show a corresponding global topology regarding the primary out-of-plane displacement (u_z). The FEM model identifies the maximum deformation at the free corner where the point load is applied, while the FDM model shows the maximum deformation more at the center of the free edge. However, both models do show the deflection gradually dissipating towards the clamped support. The topology of the circumferential displacement (u_y) is more corresponding between the models, with the largest displacement at the corner of the applied load.

A quantitative assessment the structural parameters shown in table 4.1 reveals significant numerical divergence, particularly concerning the circumferential displacement (u_y) and the out-of-plane displacement (u_z). The results highlight a fundamental difference between the two models under an asymmetric point load.

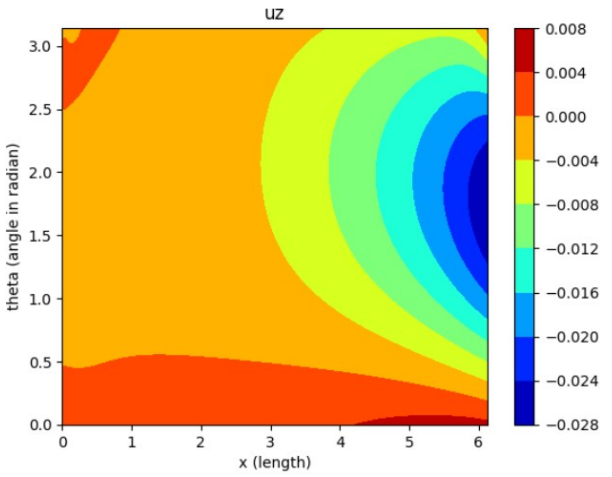
For the circumferential displacement (u_y), the FDM model exhibits peak values that are more than twice as low as the FEM baseline, failing to capture the expected transverse deformation pattern. Similarly, the FDM torsion moment (m_{xy}) displays a localized, stiff response confined to the loaded corner, entirely lacking the gradual radial distribution of forces present in the SCIA model.

Table 4.1: Comparison of peak absolute structural responses for Scenario A.

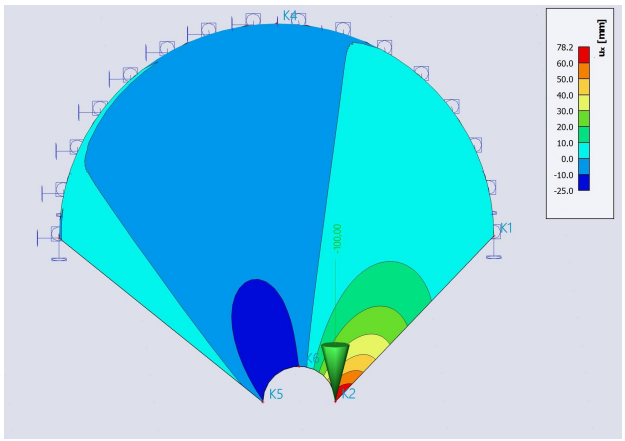
Parameter	SCIA (FEM)	Python (FDM)	Difference (%)
Max. Normal $\|u_z\|$	63.2 mm	28 mm	126%
Max. Circumferential $\|u_y\|$	78.2 mm	30 mm	161%
Max. Torsion Moment $\|m_{xy}\|$	25.1 kNm/m	32 kNm/m	22%



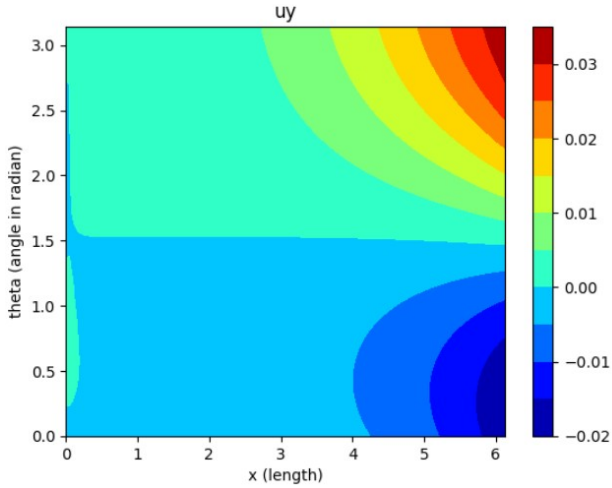
(a) SCIA (FEM): Normal displacement u_z



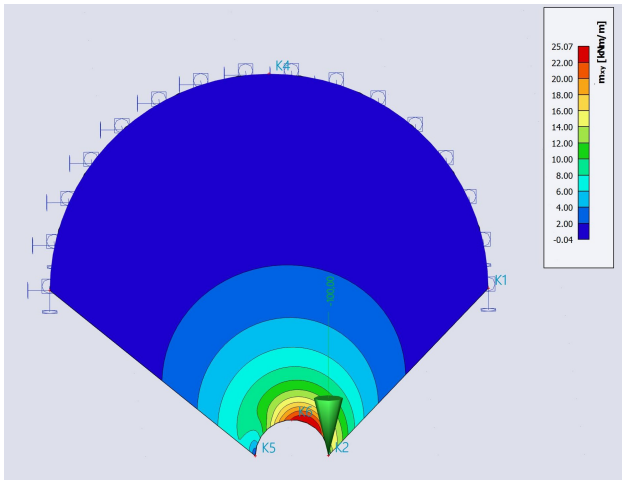
(b) Python (FDM): Normal displacement u_z



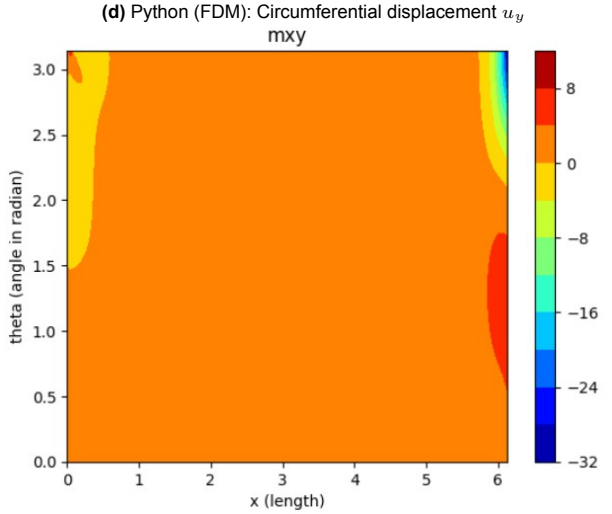
(c) SCIA (FEM): Circumferential displacement u_y



(d) Python (FDM): Circumferential displacement u_y



(e) SCIA (FEM): Torsion moment m_{xy}



(f) Python (FDM): Torsion moment m_{xy}

Figure 4.1: Left: 3D Isometric view of the FEM results. Right: Unrolled 2D contour plot ($x - \theta$ domain) of the FDM results.

4.2. Results Scenario B: Single Clamped Edge with Surface Load

Visual inspection of the contour plots for the uniform surface load (Scenario B) reveals the correlation between the FEM baseline and the developed FDM model. Unlike the singular load case in Scenario A, the distributed surface load is processed smoothly by the FDM matrix, resulting in highly comparable spatial gradients.

For the normal displacement (u_z, Figures 4.2a and 4.2b), both models correctly predict the maximum deflection at the center of the free edge ($\theta = \frac{1}{2}\pi$), with deformation bands dissipating towards the clamped support. Furthermore, the circumferential displacement (u_y, Figures 4.2c and 4.2d) exhibits a perfect anti-symmetric distribution across the longitudinal center axis in both models. This confirms that the FDM formulation correctly captures the geometric symmetries and transverse structural responses under a symmetric load.

The internal force distribution provides further validation of the model. The torsion moment (m_{xy}, Figures 4.2e and 4.2f) displays a distinct four-quadrant pattern, with positive and negative peak values situated near the constrained and free boundaries. The FDM model captures this complicated spatial distribution well.

Quantitatively, the peak values show a massive improvement compared to Scenario A. When converting the FDM output to the FEM baseline, the maximum displacements (u_z and u_y) and the internal torsion moments (m_{xy}) are within the exact same order of magnitude. The remaining numerical differences between the maximum values will be evaluated in the Discussion section.

Table 4.2: Comparison of peak absolute structural responses for Scenario B.

Parameter	SCIA (FEM)	Python (FDM)	Difference (%)
Max. Normal $\|u_z\|$	5.6 mm	5.3 mm	6%
Max. Circumferential $\|u_y\|$	5.6 mm	5.3 mm	6%
Max. Torsion Moment $\|m_{xy}\|$	0.09 kNm/m	0.11 kNm/m	22.2%

4.2.1. Grid optimization

Because the results get close to the FEM results, it is possible to do a grid optimization until the grid is most efficient. Table 4.3 shows the maximum values for the displacements (u_z and u_y) and the torsion moment (m_{xy}) using different grid sizes under the load of Scenario B. As the grid becomes finer, the results start to converge and stabilize. As the difference between a grid of 80x80 and a grid of 100x100 is close to 0, a grid of 80x80 is chosen for the rest of this scenario and the rest of the scenarios.

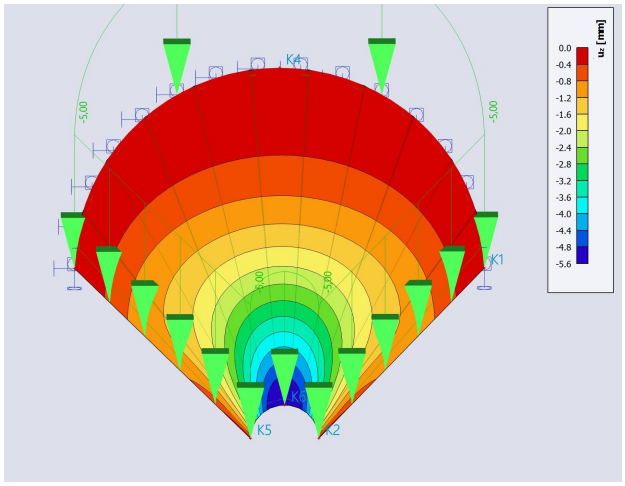
Table 4.3: Grid optimization FDM for scenario B.

Grid size (nn x mm)	u_z	u_y	m_{xy}
10 x 10	3.7 mm	4.5 mm	0.3 kNm/m
20 x 20	4.4 mm	4.7 mm	0.41 kNm/m
40 x 40	4.9 mm	5.0 mm	0.3 kNm/m
60 x 60	5.2 mm	5.2 mm	0.12 kNm/m
80 x 80	5.3 mm	5.3 mm	0.11 kNm/m
100 x 100	5.3 mm	5.3 mm	0.11 kNm/m

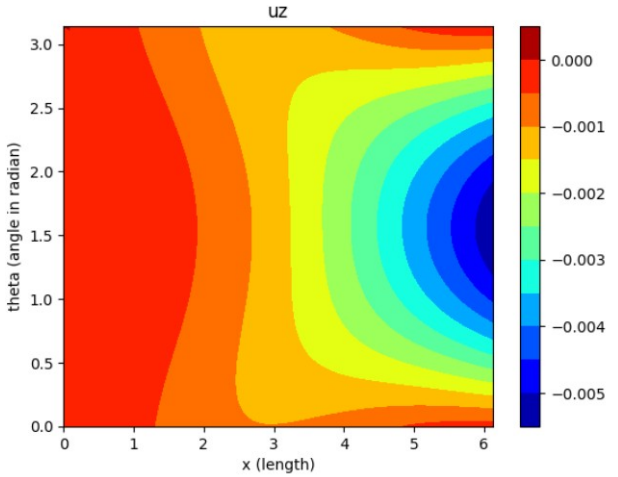
Table 4.4 shows the grid optimization for FEM for scenario B. The step from 0.05 m to 0.025 m is the first step where the values show no significant deviation, so the mesh with an element size of 0.05 m is chosen.

Table 4.4: Grid optimization FEM for scenario B.

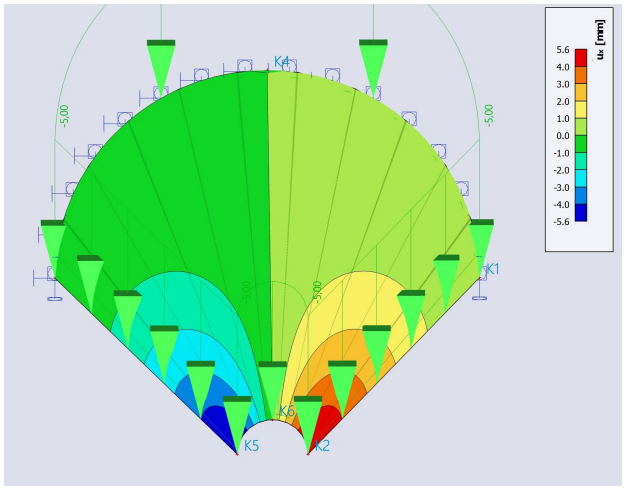
Element size	u_z	u_y	m_{xy}
0.5 m	5.6 mm	7.2 mm	0.2 kNm/m
0.4 m	5.6 mm	6.4 mm	0.12 kNm/m
0.3 m	5.6 mm	6.1 mm	0.10 kNm/m
0.2 m	5.6 mm	5.9 mm	0.09 kNm/m
0.1 m	5.6 mm	5.7 mm	0.09 kNm/m
0.05 m	5.6 mm	5.6 mm	0.09 kNm/m
0.025 m	5.6 mm	5.6 mm	0.09 kNm/m



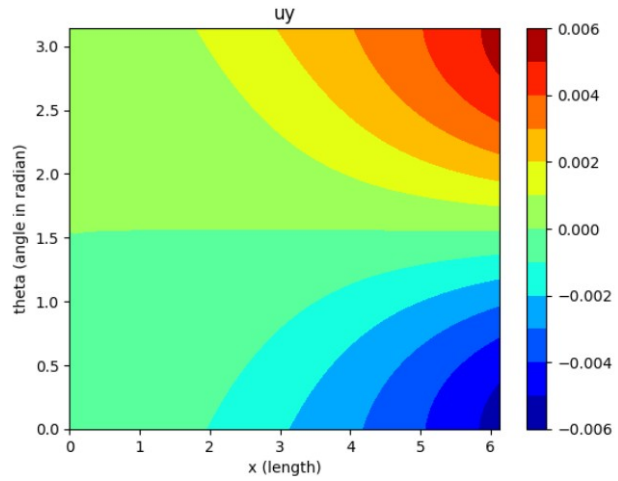
(a) SCIA (FEM): Normal displacement u_z



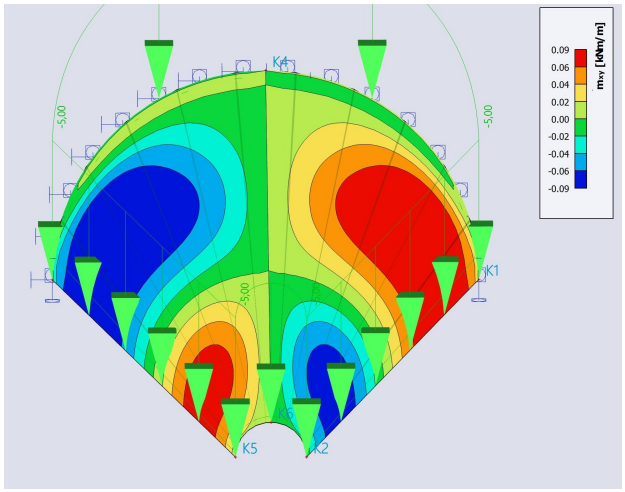
(b) Python (FDM): Normal displacement u_z



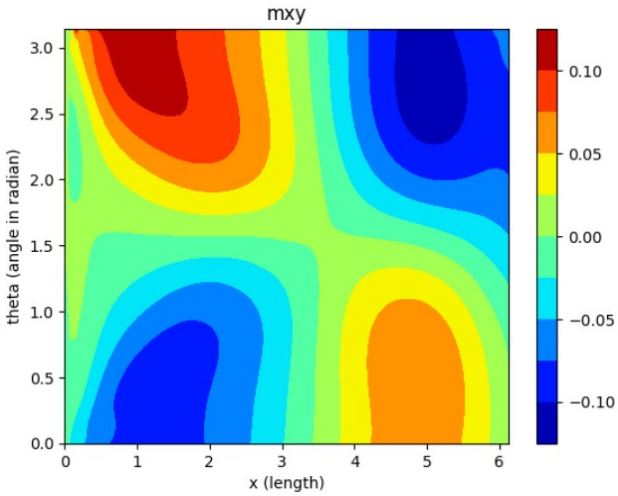
(c) SCIA (FEM): Circumferential displacement u_y



(d) Python (FDM): Circumferential displacement u_y



(e) SCIA (FEM): Torsion moment m_{xy}



(f) Python (FDM): Torsion moment m_{xy}

Figure 4.2: Left: 3D Isometric view of the FEM results. Right: Unrolled 2D contour plot ($x - \theta$ domain) of the FDM results.

4.3. Results Scenario C: Clamped-Hinged Edges under Distributed Surface Loading

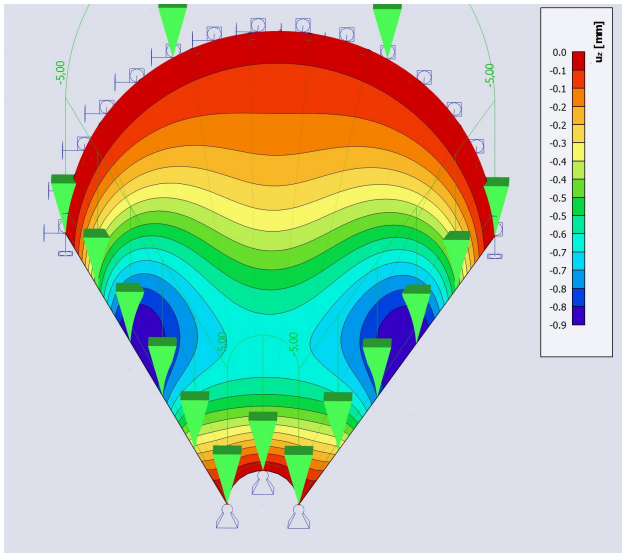
Visual inspection of the contour plots for the clamped-hinged configuration (Scenario C) reveals a structural response that is again more stable than the point load condition evaluated in Scenario A figure (figure 4.3). The normal displacement (u_z, Figures a and b) demonstrates a reasonable topological correlation between the FEM baseline and the FDM model. Both computational models successfully capture the symmetric global deflection pattern governed by the distributed surface load, with the maximum out-of-plane displacement correctly located near the center of the shell domain.

However, a critical evaluation of the kinematic fields exposes underlying numerical errors within the FDM model. While the max rotation ϕ_x is approximated reasonably well, the contour plot shows sharp, localized gradients near the right boundary corner. This indicates the presence of mathematical singularities in the FDM model.

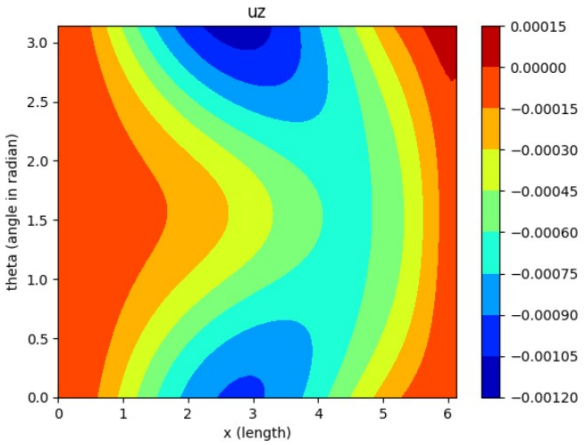
Furthermore, the inaccuracy of the FDM model in this scenario is explicitly visible in the evaluation of the internal forces (figure 4.3e and 4.3f) of m_{xy}. While the FEM baseline predicts a smooth, physically realistic stress distribution bounded between -0.9 kNm/m and $+0.9$ kNm/m, the FDM calculation is a lot less accurate, as the FDM output for m_{xy} explodes at the hinged support. Table B.1 shows that although u_z and ϕ_x seem to be in the same magnitude in both FEM and FDM models, this does not necessarily result in correctly evaluated internal stresses. For m_{xy} the singular values at the hinged support are disregarded.

Table 4.5: Comparison of peak absolute structural responses for Scenario C.

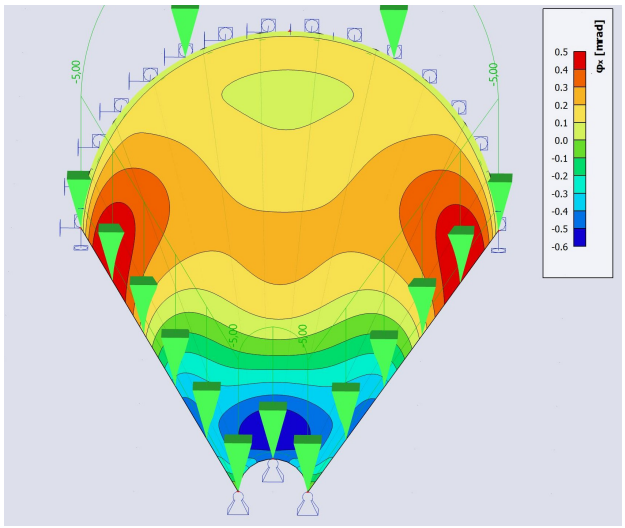
Parameter	SCIA (FEM)	Python (FDM)	Difference (%)
Max. Normal $\lvert u_z \rvert$	0.9 mm	1.1 mm	18%
Max. Bending $\lvert \phi_x \rvert$	0.6 mrad	0.73 mrad	17.8%
Torsion Moment $\lvert m_{xy} \rvert$	0.11 kNm/m	0.19 kNm/m	42.1%



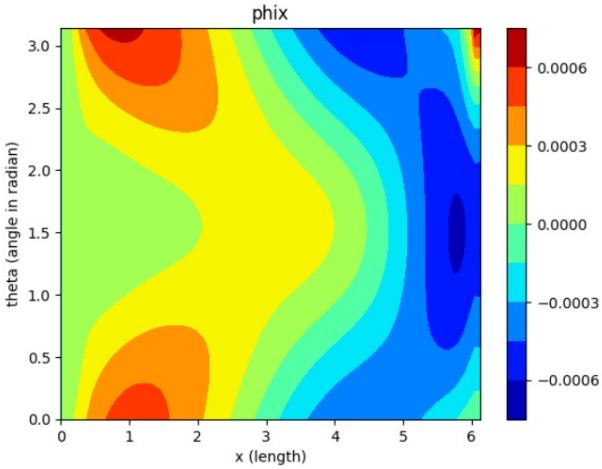
(a) SCIA (FEM): Normal displacement u_z



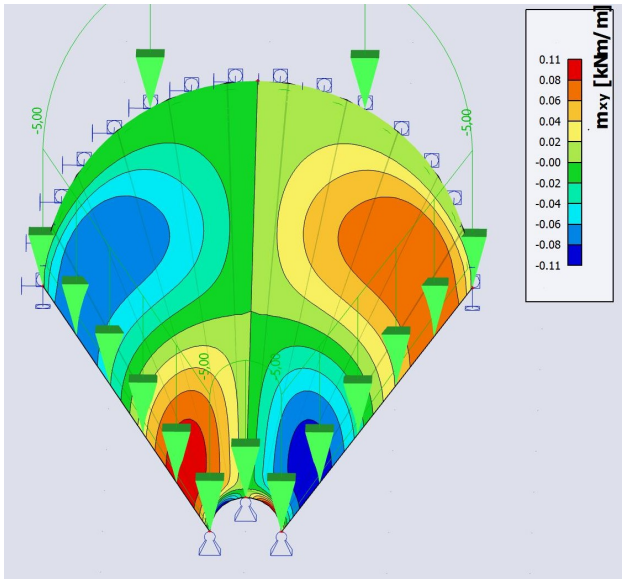
(b) Python (FDM): Normal displacement u_z



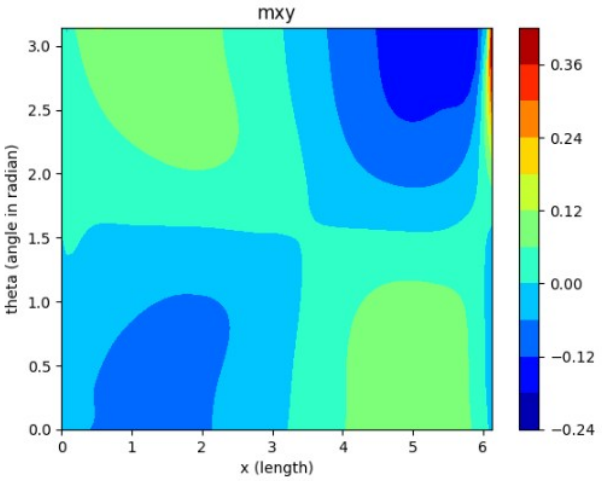
(c) SCIA (FEM): Circumferential displacement ϕ_x



(d) Python (FDM): Circumferential displacement ϕ_x



(e) SCIA (FEM): Torsion moment m_{xy}

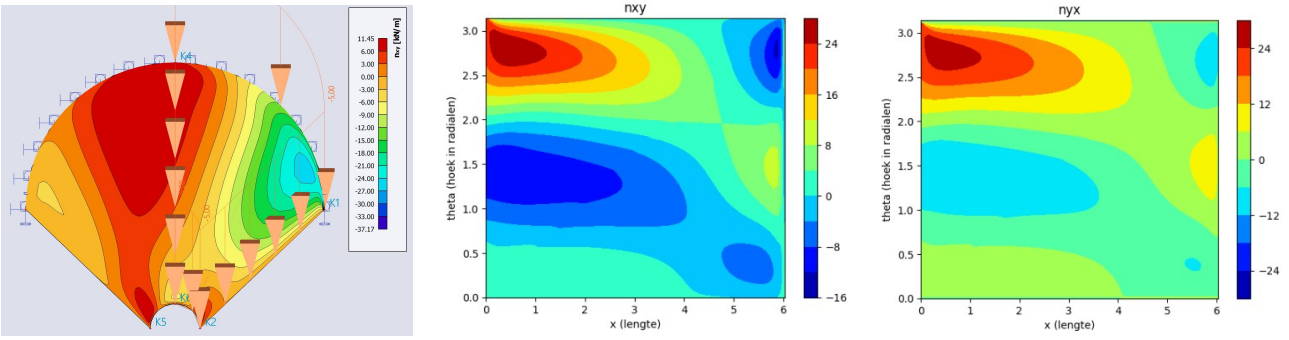


(f) Python (FDM): Torsion moment m_{xy}

Figure 4.3: Left: 3D Isometric view of the FEM results. Right: Unrolled 2D contour plot ($x - \theta$ domain) of the FDM results.

4.4. Results Scenario D: Evaluation of Membrane Shear Forces (n_{xy} and n_{yx})

As illustrated in Figure 4.4, the FDM plots for n_{xy} and n_{yx} show distinct spatial distributions under the uniform load, where the FEM model only has the n_{xy} plot.



(a) FEM baseline: Membrane shear **(b)** FDM: Membrane shear n_{xy} **(c)** FDM: Membrane shear n_{yx}

Figure 4.4: Comparison of the membrane shear forces under a uniform surface load. The FDM model calculates both n_{xy} and n_{yx}, where the FEM model only has the n_{xy} plot.

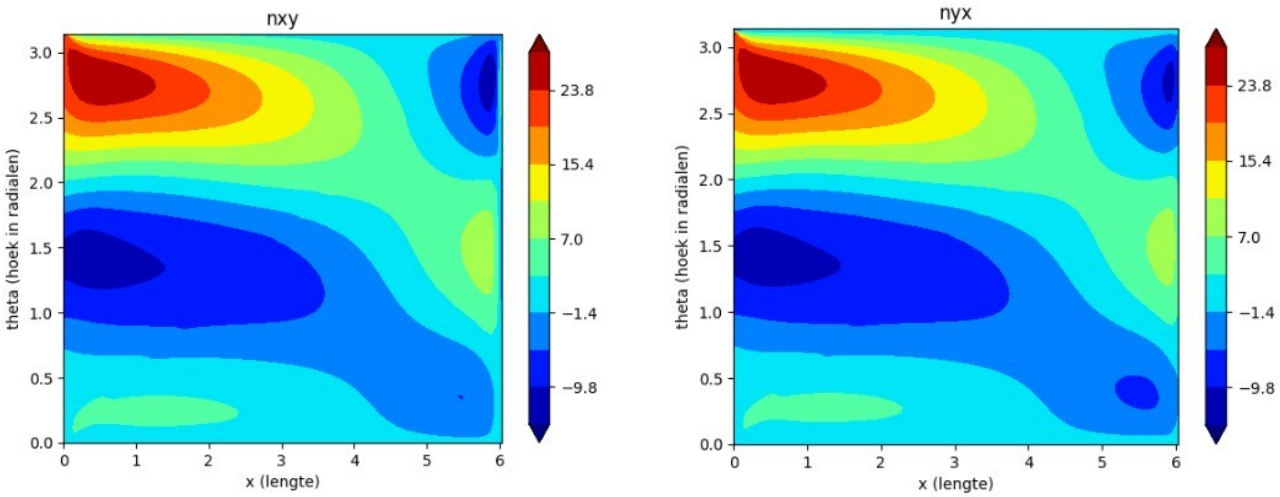
Table 4.6 presents the peak values on both the loaded and unloaded sides of the shell. The FEM results are included as a reference for the expected order of magnitude, with its local coordinate axis inverted (-1) due to the different direction of x-axis. The primary analysis focuses on the internal percentage difference between the two FDM shear components.

Table 4.6: Internal comparison of peak membrane shear forces (kN/m) within the FDM model under an asymmetric load. The difference % highlights the deviation of n_{yx} relative to n_{xy}.

Location	FEM Ref. (SCIA)	FDM n_{xy}	FDM n_{yx}	Internal Diff. FDM
Loaded side (Max)	+37.2	+28.0	+28.1	0.4%
Unloaded side (Min)	-11.5	-13.2	-28.1	112.9%

The data in Table 4.6 reveals a large internal difference within the FDM formulation depending on the location on the shell. Both n_{xy} and n_{yx} show similar peak values (+28.0 N/m and +28.1 kN/m, respectively), which align reasonably well with the expected order of magnitude provided by the FEM reference. The difference between the n_{xy} and n_{yx} values also show the expected difference between both shear forces. However, on the unloaded side, the FDM components diverge much more. The n_{xy} component correctly registers a much lower shear force (-13.2 kN/m), showing a realistic deformation of the rotating unloaded half, as validated by the FEM reference (-11.5 kN/m). In contrast, the n_{yx} component forces a peak of -28.1 kN/m, resulting in an internal deviation of 112.9% compared to n_{xy}. This confirms that for the FDM-model, n_{xy} and n_{yx} are calculated separately. However, when considering the legend on the right, for the nyx plot (Figure 4.4 c) the minimum value of -28.1 cannot be seen in the plot itself, which might a sign of a singularity.

In figure 4.5 the shear forces are plotted until a minimum of -14 kN/m and a maximum of 29 kN/m to cancel the singular, extreme values shown in the first plots. What stands out in the figure is that the plots now look more similar. In table 4.7 the results with the singularities disregarded are shown. The values of n_{xy} and n_{yx} now show a more realistic internal difference of 10.9%. The deformations of scenario D are shown in Appendix B.1. What is also remarkable is that the results between FEM and FDM deviate a lot more than in scenario B, while all conditions are the same except from the loading.



(a) FDM: Membrane shear n_{xy}

(b) FDM: Membrane shear n_{yx}

Figure 4.5: Comparison of the membrane shear forces under a uniform surface load. The plotted minimum is -12 kN/m and plotted maximum 28 kN/m.

Table 4.7: Internal comparison of peak membrane shear forces (kN/m) within the FDM model with singularities disregarded.

Location	FEM Ref. (SCIA)	FDM n_{xy}	FDM n_{yx}	Internal Diff. FDM
Loaded side (Max)	+37.2	+28.0	+28.1	0.4%
Unloaded side (Min)	-11.5	-13.2	-11.9	10.9%

5

Discussion

This chapter discusses the validity and limitations of the study's findings. The core of this research was focused on verifying the Finite Element Method with the Finite Difference Method based on the Sanders-Koiter equations. On the one hand, the FDM model used succesfully established the foundational geometric extensions (variable radius and curvature) and was able to approach FEM for a canopy with a symmetric surface load (scenario B). On the other hand, the model showed significant shortcomings when processing point loads and double supports, leading to unrealistic outcomes. This discussion analyzes the underlying causes of these deviations, evaluates the limitations of the current FDM model, and addresses the implications for future applications.

5.1. Interpretation of Results and Model Robustness

The robustness of the developed FDM model is dependent on the applied load and the type of support, where the FEM model seems to be more consistent. As demonstrated in Scenario B, the FDM model is capable of predicting internal forces and displacements when subjected to smooth, continuous surface loads. Under these symmetric, idealized conditions, the FDM matrix solves the coupling between membrane and bending stresses without numerical noise, but the results are not good enough to rely on entirely, with a difference of a minimal 6% when compared to FEM. Although this is a good approximation, the difference should be close 0. When looking closely to figure 4.2, it is notable that the plots of the FDM are not completely symmetric, which should be the case with symmetric loading in this situation. The FEM plots capture this behavior much better. This indicates that the FEM plots are more trustworthy than the FDM plots.

With only changing the load, the differences between FEM and FDM changes drastically. As in scenario B the FDM model does converge to an answer close to FEM (6% difference), the problem lies in how the Sanders-Koiter equations are programmed. It is likely that the remaining 6% difference is primarily a programming error rather than a physical or theoretical flaw. A mistake in the Python code could be the main cause for this difference, even though there is no sign of oscillations in the plots. Furthermore, as discussed the boundary conditions at the free edges are an approximation of the reality, as is assumed that $v_y = \frac{\partial V_y}{\partial x} = 0$ for simplicity. However, this will have a small impact, as v_y is close to 0.

Furthermore, the model's accuracy decreases when singularities or severe local gradients are introduced. In Scenario A, the application of a concentrated point load caused a massive divergence from the FEM baseline. As the rest of the code is the same as in situation B, the way the point load is programmed can be the main cause of this unrealistic result. In the asymmetric loading conditions (A and D), the torsion terms become more dominant in the formulas, making the result more dependent of these terms. As the results become less reliable when the torsion terms (m_{xy}) become more dominant, this suggests a programming error of these terms.

5.2. Data Reliability

The overall reliability of the data generated by the developed FDM model is conditional. Although the 6% deviation from the FEM baseline observed in Scenario B appears as an acceptable margin of error for a shell approximation, it is crucial to emphasize that this level of accuracy is limited to idealized, symmetric loading conditions.

When evaluating the model's response to more complicated situations such as concentrated point loads or asymmetric loading, the generated data shows even more divergence resulting in unrealistic local deformations. This results in that the overall reliability of the dataset is vulnerable outside of symmetric scenarios.

Consequently, while the FDM results can be considered sufficiently reliable for analyzing specific, symmetric load cases, the data output cannot currently be trusted for general structural analysis involving asymmetric loading. A comparison between FEM and FDM must therefore be within these symmetric limitations.

5.3. Limitations of the Current FDM Framework

Despite successfully validating the geometric extensions of the Sanders-Koiter theory, the current implementation of the Finite Difference framework is limited by several fundamental mathematical and numerical limitations.

Firstly, the algorithm lacks a natural mechanism to model high localized forces, such as colomn supports or connection points. In reality, the forces are always transferred over a contact area rather than a singular point. However, to represent a small column support over a large canopy, will require a relatively fine grid to calculate the stresses around this column correctly, which increases the computational time drastically. This FDM model uses a uniform grid for the canopy, where a FEM model could use local mesh refinement.

Secondly, the model is limited by its uniform spatial discretizations and strict geometric scope. Achieving sufficient accuracy to capture steep local gradients near edge disturbances requires a globally dense grid, which drastically increases computational overhead without providing additional value to the smooth interior regions. Furthermore, the current mathematical formulation is strictly confined to idealized conical geometries; the script cannot accommodate structural imperfections, asymmetrical cut-outs, or transitions to other shell topologies.

5.4. Usability in Practice

Based on the current findings, the developed FDM script is not yet suitable for general engineering practice. Real-world canopy structures are frequently subjected to non-uniform, asymmetric loading (e.g. wind loads) and discrete singular supports, for which this study has demonstrated the current FDM model is not able to analyze constructions as reliable as FEM does. Nevertheless, the model holds significant academic value. For pure theoretical research into continuous shell mechanics and the fundamental interactions of the Sanders-Koiter equations, the FDM framework offers a transparent, unfiltered view into the mathematical core of the structure that commercial FEM software often doesn't show.

6

Conclusion and Outlook

The objective of this research was focused on verifying the Finite Element Method with the Finite Difference Method using the Sanders-Koiter equations. While the two distinct methods aim to predict the same structural behavior, they both solve other equations. For this reason FDM should theoretically be a good method to compare with FEM. However, this study demonstrated that the current FDM framework is not yet capable of verifying the FEM model. The results of the asymmetric loading scenarios show that the FDM model's accuracy decreases drastically when the torsional moments become more dominant. The boundary conditions for the symmetric- and asymmetric loading scenarios were the same, which suggests that there is an underlying programming error in the FDM model. This results in that the FEM results will most likely be closest to reality, making the FDM method using the Sanders-Koiter equations unable to show FEM to be not accurate. However, the FDM model did succeed in showing that in the Sanders-Koiter equations the distinction between the membrane forces ($n_{xy} \neq n_{yx}$) is not neglected, with an internal difference of 10.9%. The following sub questions were answered to come to this conclusion:

6.1. Summary of Findings

The research objectives were met by addressing the three formulated sub-questions:

How does the geometry of a conical shell change the Sanders-Koiter equations compared to a standard cylinder?

The transition to a conical geometry changes the way the Sanders-Koiter equations are formulated. The inclusion of the angle with the horizontal (α) introduces a variable local radius $R(x)$ (after which $k_{yy} = -\cos(\alpha)/R(x)$) and activates the geodesic curvature ($k_y = -\sin(\alpha)/R(x)$). These terms were successfully implemented in the FDM model. These extra terms mathematically links the x-direction with the y direction, which can be seen in the formula for ϵ or the equilibrium equations.

To what extent can the FDM model accurately predict the deformations of a canopy under continuous and asymmetric load configurations when compared to FEM?

The validation against the FEM baseline resulted in mixed results, depending on the load case. Under symmetric, continuous surface loads (Scenario B), the FDM model produced global displacement patterns comparable to the FEM model, giving an error margin of 6% for the displacements. Moreover, the model demonstrated the distinction between the membrane shear forces ($n_{xy} \neq n_{yx}$), with a difference of 10.9% between the two forces. However, the validation exposed larger differences between FEM and FDM under localized and asymmetric loading. Under singular point loads (Scenario A) and situations with multiple supports (Scenario C), the FDM model differs significantly from the expected physical behavior with an error margin of at least 18%. Furthermore, the FEM framework demonstrates a significantly higher consistency across different loading cases, where the FDM model is volatile. This sensitivity of the FDM script to minor adjustments in the load cases strongly suggests that the FEM more accurately reflects the true physical behavior of the shell.

What is the numerical impact of local disturbances, such as singular point loads and different boundary conditions, on the accuracy of the FDM model?

Singularities can be disregarded when adjusting the color bar. They do not influence the numerical accuracy of the solution entirely. However, the evaluations in Scenario A (point load) and Scenario C (clamped-hinged

boundaries) showed the fragility of the used FDM script. In these situations the error are caused by how the discrete point load and boundary transitions are programmed. The current framework is not able to process such sudden, localized constraints within the grid.

What grid- and element size are needed for a reliable result for FDM and FEM respectively?

For the Finite Difference Method (FDM), a grid size of 80×80 nodes was found to be necessary to minimize truncation errors over the shell surface. Furthermore, the Finite Element Method (FEM) requires an average element size of 0.05 m to achieve full mesh convergence. At these specific resolutions, both numerical models reach a stable state where further refinement does not result in significant changes in the computed displacements or internal forces.

6.2. Future Work

Based on the numerical limitations encountered during the validation process, several areas for future research and model improvement are proposed:

- **Mathematical Treatment of Point Loads:** Currently, applying a concentrated load to a single point causes high singularities in the FDM model, as seen in Scenario A. To bridge the gap with FEM, future research should explore local mathematical ”smoothing” techniques. By spreading a singular point load over a small patch of surrounding grid nodes, the model can prevent extreme stress peaks and calculate the forces correctly.
- **Adaptive Grid Refinement:** The current model uses a uniform grid spacing across the entire canopy. A major improvement would be implementing an adaptive grid. This means the grid automatically becomes finer (higher resolution) near difficult areas, like edge disturbances and supports. This approach would optimize both the calculation speed and the accuracy of the model.
- **Implementation of Complex Geometries and Cut-outs:** As concluded, the current FDM grid requires a perfectly continuous shell. In architectural practice, canopies often feature cut-outs for skylights, pipes, or asymmetric edges. Future research could explore mapping techniques. This would allow the framework to break the canopy into multiple connected sub-grids, making it possible to calculate shells with holes or irregular boundaries without breaking the equations.

References

Billington, D. P. (1982). *Thin shell concrete structures* (2nd). McGraw-Hill. https://search.worldcat.org/search?q=bn:9780070052796

Calladine, C. R. (1983). *Theory of shell structures*. Cambridge University Press. https://www.google.nl/books/edition/Theory_of_Shell_Structures/REKJpbYCtu4C?hl=nl&gbpv=1&dq=isbn:9780521369459&printsec=frontcover

Chapra, S. C., & Canale, R. P. (2015). *Numerical methods for engineers* (7th). McGraw-Hill Education. https://www.google.nl/books/edition/Numerical_Methods_for_Engineers/SA1LPgAACAAJ?hl=nl

Hoogenboom, P. C. J. (2023). Python computational framework for mixed finite difference method analysis of shell structures [Unpublished base code provided for Bachelor thesis research, Delft University of Technology]. https://phoogenboom.nl/b17_code.txt

Hoogenboom, P. *Notes on shell structures*. Lecture notes, Delft University of Technology. Delft, The Netherlands, 2024. https://phoogenboom.nl/b17_book.pdf

Koiter, W. T. (1960). A consistent first approximation in the general theory of thin elastic shells. *Theory of thin elastic shells*, 12–33. https://books.google.nl/books/about/A_Consistent_First_Approximation_in_the.html?id=h5TZtgAACAAJ&redir_esc=y

LeVeque, R. J. (2007). *Finite difference methods for ordinary and partial differential equations: Steady-state and time-dependent problems*. SIAM. https://books.google.nl/books/about/Finite_Difference_Methods_for_Ordinary_a.html?id=lZysESfSWwgC&redir_esc=y

Noor, A. K., & Stephens, W. B. (1973). Mixed finite-difference scheme for free-vibration analysis of shells of revolution. *NASA Technical Note*, *7107*. https://ntrs.nasa.gov/api/citations/19730008181/downloads/19730008181.pdf

Sanders Jr, J. L. (1959). An improved first-approximation theory for thin shells. *NASA Technical Report*, *24*. https://archive.org/details/nasa_techdoc_19980227978

Timoshenko, S., & Woinowsky-Krieger, S. (1959). *Theory of plates and shells* (2nd). McGraw-Hill. https://search.worldcat.org/title/theory-of-plates-and-shells/oclc/1429956

Zienkiewicz, O. C., & Taylor, R. L. (2005). *The finite element method for solid and structural mechanics*. Elsevier. https://books.google.nl/books?id=VvpU3zssDOwC&printsec=frontcover&hl=nl#v=onepage&q&f=false

A

Source Code Example

A.1. Main Python Code

This scenario A models the reference situation of the conical shell with a fully fixed boundary.

```
# -*- coding: utf-8 -*-
"""Puntlast.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1WjI8yIvXr7YG1uASapEj8kaXQl31JVMO
"""

import numpy as np
import matplotlib.pyplot as plt
from scipy.sparse import lil_matrix
from scipy.sparse.linalg import spsolve

# --- 1. GEOMETRIE EN MATERIAAL ---
E  = 1e7    # kN/m2 Young's modulus
nu = 0.15   # Poisson's ratio


x_length = 6    # Cantilevered length
r_start = 1.5   # Radius at the support
r_end   = 0.25  # Radius at the free tip
l  = np.sqrt(x_length**2 + (r_start-r_end)**2) # length x-axis (slanted)
t  = 0.060  # m     Canopy thickness
F  = 100.0  # kN    Load

# Grid settings
mm = 125  # Grid points
nn = 125    # Grid points



Radius = np.linspace(r_start, r_end, mm)
alpha = np.arctan((r_start-r_end) / x_length)


k_yy = np.zeros(mm)
k_y  = np.zeros(mm)

for i in range(mm):
    if Radius[i] > 0.0001:
        #physical curvature (circumferential)
        k_yy[i] = -np.cos(alpha) / Radius[i]
        #Geodesic curvature (flaring effect)
        k_y[i]  = -np.sin(alpha) / Radius[i]


def Dx(k, g):
    h = l / (mm - 1)
    M[row, p[k] + j*m[k] + i]     = -g/h
    M[row, p[k] + j*m[k] + i + 1] =  g/h
```

```
    return

def Dy(k, g, i_loc):
    local_r = Radius[i_loc]
    h = np.pi * local_r / (nn - 1)
    M[row, p[k] + (j)*m[k] + i]     = -g/h
    M[row, p[k] + (j+1)*m[k] + i]   =  g/h
    return

# --- 3. VARIABELS
ux=0; uy=1; uz=2; epsilonxx=3; epsilonyy=4; gammaxy=5; phix=6; phiy=7; phiz=8
kappaxx=9; kappayy=10; rhoxy=11; nxx=12; nyy=13; nxy=14; nyx=15; mxx=16; myy=17; mxy=18; vx=19; vy
    =20

p=[0 for i in range(21)]
m=[0 for i in range(21)]
n=[0 for i in range(21)]

p[ux]=0;                                        m[ux]=mm;   n[ux]=nn
p[uy]=p[ux]+m[ux]*n[ux];                        m[uy]=mm;   n[uy]=nn
p[uz]=p[uy]+m[uy]*n[uy];                        m[uz]=mm;   n[uz]=nn
p[epsilonxx]=p[uz]+m[uz]*n[uz];                 m[epsilonxx]=mm-1; n[epsilonxx]=nn
p[epsilonyy]=p[epsilonxx]+m[epsilonxx]*n[epsilonxx]; m[epsilonyy]=mm; n[epsilonyy]=nn-1
p[gammaxy]=p[epsilonyy]+m[epsilonyy]*n[epsilonyy]; m[gammaxy]=mm-1; n[gammaxy]=nn-1
p[phix]=p[gammaxy]+m[gammaxy]*n[gammaxy];       m[phix]=mm+1; n[phix]=nn
p[phiy]=p[phix]+m[phix]*n[phix];                m[phiy]=mm;   n[phiy]=nn+1
p[phiz]=p[phiy]+m[phiy]*n[phiy];                m[phiz]=mm-1; n[phiz]=nn-1
p[kappaxx]=p[phiz]+m[phiz]*n[phiz];             m[kappaxx]=mm; n[kappaxx]=nn
p[kappayy]=p[kappaxx]+m[kappaxx]*n[kappaxx];    m[kappayy]=mm; n[kappayy]=nn
p[rhoxy]=p[kappayy]+m[kappayy]*n[kappayy];      m[rhoxy]=mm-1; n[rhoxy]=nn-1
p[nxx]=p[rhoxy]+m[rhoxy]*n[rhoxy];              m[nxx]=mm+1; n[nxx]=nn
p[nyy]=p[nxx]+m[nxx]*n[nxx];                    m[nyy]=mm;   n[nyy]=nn+1
p[nxy]=p[nyy]+m[nyy]*n[nyy];                    m[nxy]=mm;   n[nxy]=nn-1
p[nyx]=p[nxy]+m[nxy]*n[nxy];                    m[nyx]=mm-1; n[nyx]=nn+1
p[mxx]=p[nyx]+m[nyx]*n[nyx];                    m[mxx]=mm;   n[mxx]=nn
p[myy]=p[mxx]+m[mxx]*n[mxx];                    m[myy]=mm;   n[myy]=nn
p[mxy]=p[myy]+m[myy]*n[myy];                    m[mxy]=mm-1; n[mxy]=nn-1
p[vx]=p[mxy]+m[mxy]*n[mxy];                     m[vx]=mm+1;  n[vx]=nn
p[vy]=p[vx]+m[vx]*n[vx];                        m[vy]=mm;    n[vy]=nn+1
n_unknown = p[vy]+m[vy]*n[vy]

print(f'Start calculation for geometry: R_start={r_start}, R_end={r_end}')
print('Find memory.')
M=lil_matrix((n_unknown,n_unknown))
f=np.array([0.0 for i in range(n_unknown)])
u=np.array([0.0 for i in range(n_unknown)])

print('Add the Sanders-Koiter equations to the matrix.')
row=-1

for j in range(n[epsilonxx]): # epsilonxx
    for i in range(m[epsilonxx]):
        row=row+1
        M[row,p[epsilonxx]+j*m[epsilonxx]+i]=-1.0
        Dx(ux,1.0)

for j in range(n[epsilonyy]): # epsilonyy
    for i in range(m[epsilonyy]):
        row=row+1
        M[row,p[epsilonyy]+j*m[epsilonyy]+i]=-1.0
        Dy(uy,1.0, i)
        M[row,p[uz]+(j  )*m[uz]+i] = -0.5 * k_yy[i]
        M[row,p[uz]+(j+1)*m[uz]+i] = -0.5 * k_yy[i]
        M[row,p[ux]+(j  )*m[ux]+i] =  0.5 * k_y[i]
        M[row,p[ux]+(j+1)*m[ux]+i] =  0.5 * k_y[i]

for j in range(n[gammaxy]): # gammaxy
    for i in range(m[gammaxy]):
        row=row+1
        M[row,p[gammaxy]+j*m[gammaxy]+i]=-1.0
        Dy(ux,1.0, i)
        Dx(uy,1.0)
        M[row,p[uy]+(j  )*m[uy]+i] -= 0.5 * k_y[i]
        M[row,p[uy]+(j+1)*m[uy]+i] -= 0.5 * k_y[i]

for j in range(n[phix]): # phix
```

```
    for i in range(m[phix]-2):
        row=row+1
        M[row,p[phix]+j*m[phix]+i+1]=-1.0
        Dx(uz,-1.0)

for j in range(n[phiy]-2): # phiy
    for i in range(m[phiy]):
        row=row+1
        M[row,p[phiy]+(j+1)*m[phiy]+i]=-1.0
        Dy(uz,-1.0, i)
        M[row,p[uy]+(j  )*m[uy]+i] = -0.5 * k_yy[i]
        M[row,p[uy]+(j+1)*m[uy]+i] = -0.5 * k_yy[i]

for j in range(n[phiz]): # phiz
    for i in range(m[phiz]):
        row=row+1
        M[row,p[phiz]+j*m[phiz]+i]=-1.0
        Dy(ux,-0.5, i)
        Dx(uy, 0.5)
        M[row,p[uy]+(j  )*m[uy]+i] += 0.25 * k_y[i]
        M[row,p[uy]+(j+1)*m[uy]+i] += 0.25 * k_y[i]

for j in range(n[kappaxx]): # kappaxx
    for i in range(m[kappaxx]):
        row=row+1
        M[row,p[kappaxx]+j*m[kappaxx]+i]=-1.0
        Dx(phix,1.0)

for j in range(n[kappayy]): # kappayy
    for i in range(m[kappayy]):
        row=row+1
        M[row,p[kappayy]+j*m[kappayy]+i]=-1.0
        Dy(phiy,1.0, i)
        M[row,p[phix]+(j  )*m[phix]+i] = 0.5 * k_y[i]
        M[row,p[phix]+(j+1)*m[phix]+i] = 0.5 * k_y[i]



for j in range(n[rhoxy]): # rhoxy
    for i in range(m[rhoxy]):
        row=row+1
        M[row,p[rhoxy]+(j)*m[rhoxy]+i]=-1.0
        h_x = l / (mm - 1)
        h_y = np.pi * Radius[i] / (nn - 1)

        M[row, p[phix] + (j  ) * m[phix] + i] = -1.0 / h_y
        M[row, p[phix] + (j + 1) * m[phix] + i] =  1.0 / h_y
        M[row, p[phiy] + j * m[phiy] + i  ] = -1.0 / h_x
        M[row, p[phiy] + j * m[phiy] + i+1] =  1.0 / h_x

        M[row,p[phiy]+(j  )*m[phiy]+i] -= 0.5 * k_y[i]
        M[row,p[phiy]+(j+1)*m[phiy]+i] -= 0.5 * k_y[i]
        M[row,p[phiz]+(j  )*m[phiz]+i] = -1.0 * k_yy[i]

k=(E*t)/(1-nu**2)
for j in range(n[nxx]):
    for i in range(1,m[nxx]-1):
        row=row+1
        M[row,p[nxx      ]+(j  )*m[nxx      ]+i  ]=-1.0
        M[row,p[epsilonxx]+(j  )*m[epsilonxx]+i-1]=k
        if j==0:
            M[row,p[epsilonyy]+(j  )*m[epsilonyy]+i-1]=0.50*nu*k
            M[row,p[epsilonyy]+(j  )*m[epsilonyy]+i  ]=0.50*nu*k
        elif j==n[nxx]-1:
            M[row,p[epsilonyy]+(j-1)*m[epsilonyy]+i-1]=0.50*nu*k
            M[row,p[epsilonyy]+(j-1)*m[epsilonyy]+i  ]=0.50*nu*k
        else:
            M[row,p[epsilonyy]+(j-1)*m[epsilonyy]+i-1]=0.25*nu*k
            M[row,p[epsilonyy]+(j-1)*m[epsilonyy]+i  ]=0.25*nu*k
            M[row,p[epsilonyy]+(j  )*m[epsilonyy]+i-1]=0.25*nu*k
            M[row,p[epsilonyy]+(j  )*m[epsilonyy]+i  ]=0.25*nu*k

for j in range(1,n[nyy]-1):
    for i in range(m[nyy]):
        row=row+1
        M[row,p[nyy      ]+(j  )*m[nyy      ]+i  ]=-1.0
```

```
        M[row,p[epsilonyy]+(j-1)*m[epsilonyy]+i  ]=k
        if i==0:
            M[row,p[epsilonxx]+(j-1)*(m[epsilonxx]-1)+i  ]=0.50*nu*k
            M[row,p[epsilonxx]+(j  )*(m[epsilonxx]-1)+i  ]=0.50*nu*k
        elif i==m[nyy]-1:
            M[row,p[epsilonxx]+(j-1)*(m[epsilonxx]-1)+i-1]=0.50*nu*k
            M[row,p[epsilonxx]+(j  )*(m[epsilonxx]-1)+i-1]=0.50*nu*k
        else:
            M[row,p[epsilonxx]+(j-1)*(m[epsilonxx]-1)+i-1]=0.25*nu*k
            M[row,p[epsilonxx]+(j  )*(m[epsilonxx]-1)+i-1]=0.25*nu*k
            M[row,p[epsilonxx]+(j-1)*(m[epsilonxx]-1)+i  ]=0.25*nu*k
            M[row,p[epsilonxx]+(j  )*(m[epsilonxx]-1)+i  ]=0.25*nu*k

k=(E*t)/(2*(1+nu))
for j in range(n[gammaxy]):
    for i in range(m[gammaxy]):
        row=row+1
        M[row,p[nxy    ]+(j  )*m[nxy    ]+i]=-0.5
        M[row,p[nyx    ]+(j+1)*m[nyx    ]+i]=-0.5
        M[row,p[gammaxy]+(j  )*m[gammaxy]+i]=k

k=(E*t**3)/(12*(1-nu**2))
for j in range(n[mxx]):
    for i in range(m[mxx]):
        row=row+1
        M[row,p[mxx    ]+j*m[mxx    ]+i]=-1.0
        M[row,p[kappaxx]+j*m[kappaxx]+i]=k
        M[row,p[kappayy]+j*m[kappayy]+i]=nu*k

for j in range(n[myy]):
    for i in range(m[myy]):
        row=row+1
        M[row,p[myy    ]+j*m[myy    ]+i]=-1.0
        M[row,p[kappayy]+j*m[kappayy]+i]=k
        M[row,p[kappaxx]+j*m[kappaxx]+i]=nu*k

k=(E*t**3)/(24*(1+nu))
for i in range(n[mxy]*m[mxy]):
    row=row+1
    M[row,p[mxy  ]+i]=-1.0
    M[row,p[rhoxy]+i]=k




for j in range(n[vx]): # vx
    for i in range(1,m[vx]-1):
        row=row+1
        M[row,p[vx ]+j*m[vx ]+i ]=-1.0
        h_x=l/(mm-1)
        M[row,p[mxx]+j*m[mxx]+i-1]=-1.0/h_x
        M[row,p[mxx]+j*m[mxx]+i  ]= 1.0/h_x
        h_y = np.pi*Radius[i]/(nn-1)

        if j==0:
            M[row,p[mxy]+(j  )*m[mxy]+i-1]=-1.0/h_y
            M[row,p[mxy]+(j+1)*m[mxy]+i-1]= 1.0/h_y
        elif j==n[vx]-1:
            M[row,p[mxy]+(j-2)*m[mxy]+i-1]=-1.0/h_y
            M[row,p[mxy]+(j-1)*m[mxy]+i-1]= 1.0/h_y
        else:
            M[row,p[mxy]+(j-1)*m[mxy]+i-1]=-1.0/h_y
            M[row,p[mxy]+(j  )*m[mxy]+i-1]= 1.0/h_y

        if i > 0 and i < mm:
            M[row, p[mxx] + j*m[mxx] + i - 1] += 0.5 * k_y[i]
            M[row, p[mxx] + j*m[mxx] + i    ] += 0.5 * k_y[i]
            M[row, p[myy] + j*m[myy] + i - 1] -= 0.5 * k_y[i]
            M[row, p[myy] + j*m[myy] + i    ] -= 0.5 * k_y[i]



for j in range(1,n[vy]-1): # vy
    for i in range(m[vy]):
        row=row+1
        M[row,p[vy]+j*m[vy]+i]=-1.0
```

```
        h_y = np.pi*Radius[i]/(nn-1)
        M[row,p[myy]+(j-1)*m[myy]+i]=-1.0/h_y
        M[row,p[myy]+(j  )*m[myy]+i]= 1.0/h_y
        h_x= l / (mm-1)
        if i==0:
            M[row,p[mxy]+(j-1)*m[mxy]+i  ]=-1.0/h_x
            M[row,p[mxy]+(j-1)*m[mxy]+i+1]= 1.0/h_x
            M[row,p[mxy]+(j-1)*m[mxy]+i  ] += k_y[i]
            M[row,p[mxy]+(j-1)*m[mxy]+i+1] += k_y[i]
        elif i==m[vy]-1:
            M[row,p[mxy]+(j-1)*m[mxy]+i-2]=-1.0/h_x
            M[row,p[mxy]+(j-1)*m[mxy]+i-1]= 1.0/h_x
            M[row,p[mxy]+(j-1)*m[mxy]+i-2] += k_y[i]
            M[row,p[mxy]+(j-1)*m[mxy]+i-1] += k_y[i]
        else:
            M[row,p[mxy]+(j-1)*m[mxy]+i-1]=-1.0/h_x
            M[row,p[mxy]+(j-1)*m[mxy]+i  ]= 1.0/h_x
            M[row,p[mxy]+(j-1)*m[mxy]+i-1] += k_y[i]
            M[row,p[mxy]+(j-1)*m[mxy]+i  ] += k_y[i]



for j in range(n[mxy]): # nxy - nyx
    for i in range(m[mxy]):
        row=row+1
        M[row,p[mxy]+(j  )*m[mxy]+i]= -k_yy[i]
        M[row,p[nxy]+(j  )*m[nxy]+i]= -1.0
        M[row,p[nyx]+(j+1)*m[nyx]+i]= 1.0



for j in range(nn): # Eq_x: 0 = dnxx/dx + dnyx/dy + ky(nxx-nyy)
    for i in range(mm):
        row=row+1
        Dx(nxx,1.0)
        h_y = np.pi*Radius[i]/(nn-1)

        if i==0:
            M[row,p[nyx]+(j  )*m[nyx]+i  ]=-1.0/h_y
            M[row,p[nyx]+(j+1)*m[nyx]+i  ]= 1.0/h_y
        elif i==mm-1:
            M[row,p[nyx]+(j  )*m[nyx]+i-1]=-1.0/h_y
            M[row,p[nyx]+(j+1)*m[nyx]+i-1]= 1.0/h_y
        else:
            M[row,p[nyx]+(j  )*m[nyx]+i  ]=-0.5/h_y
            M[row,p[nyx]+(j+1)*m[nyx]+i  ]= 0.5/h_y
            M[row,p[nyx]+(j  )*m[nyx]+i-1]=-0.5/h_y
            M[row,p[nyx]+(j+1)*m[nyx]+i-1]= 0.5/h_y


        if i > 0 and i < mm-1:
            M[row, p[nxx] + (j  )*m[nxx] + i  ] += 0.5 * k_y[i]
            M[row, p[nxx] + (j  )*m[nxx] + i-1] += 0.5 * k_y[i]
            M[row, p[nyy] + (j  )*m[nyy] + i  ] -= 0.5 * k_y[i]
            M[row, p[nyy] + (j  )*m[nyy] + i-1] -= 0.5 * k_y[i]



for j in range(nn): # Eq_y: 0 = dnyy/dy + dnxy/dx + ky(nxy+nyx) - kyy*vy
    for i in range(1,mm):
        row=row+1
        Dy(nyy,1.0, i)
        h_x=l/(mm-1)
        if j==0:
            M[row,p[nxy]+(j  )*m[nxy]+i-1]=-1.0/h_x
            M[row,p[nxy]+(j  )*m[nxy]+i  ]= 1.0/h_x
        elif j==nn-1:
            M[row,p[nxy]+(j-1)*m[nxy]+i-1]=-1.0/h_x
            M[row,p[nxy]+(j-1)*m[nxy]+i  ]= 1.0/h_x
        else:
            M[row,p[nxy]+(j  )*m[nxy]+i-1]=-0.5/h_x
            M[row,p[nxy]+(j  )*m[nxy]+i  ]= 0.5/h_x
            M[row,p[nxy]+(j-1)*m[nxy]+i-1]=-0.5/h_x
            M[row,p[nxy]+(j-1)*m[nxy]+i  ]= 0.5/h_x


```

```
        if j > 0 and j < nn-1:
            M[row,p[nxy]+(j  )*m[nxy]+i  ] += 0.5 * k_y[i]
            M[row,p[nxy]+(j  )*m[nxy]+i-1] += 0.5 * k_y[i]
            M[row,p[nyx]+(j  )*m[nyx]+i  ] += 0.5 * k_y[i]
            M[row,p[nyx]+(j+1)*m[nyx]+i  ] += 0.5 * k_y[i]
        M[row,p[vy]+(j  )*m[vy]+i] -= 0.5 * k_yy[i]
        M[row,p[vy]+(j+1)*m[vy]+i] -= 0.5 * k_yy[i]



for j in range(nn): # Eq_z: 0 = kyy*nyy + dvx/dx + dvy/dy + ky*vx
    for i in range(mm):
        row=row+1
        M[row,p[nyy]+(j  )*m[nyy]+i] += 0.5 * k_yy[i]
        M[row,p[nyy]+(j+1)*m[nyy]+i] += 0.5 * k_yy[i]
        Dx(vx, 1.0)
        Dy(vy, 1.0, i)
        if i > 0 and i < mm-1:
            M[row,p[vx]+(j  )*m[vx]+i  ] += 0.5 * k_y[i]
            M[row,p[vx]+(j  )*m[vx]+i-1] += 0.5 * k_y[i]

print('Insert the boundary conditions.')
for j in range(nn): # Fixed edge
    row=row+1
    M[row,p[ux  ]+j*m[ux  ]+0]=1.0
    row=row+1
    M[row,p[uy  ]+j*m[uy  ]+0]=1.0
    row=row+1
    M[row,p[uz  ]+j*m[uz  ]+0]=1.0
    row=row+1
    M[row,p[phix]+j*m[phix]+0]=0.5
    M[row,p[phix]+j*m[phix]+1]=0.5

for j in range(nn): # Free edge
    row=row+1
    M[row,p[nxx]+(j+1)*m[nxx]-1]=0.5
    M[row,p[nxx]+(j+1)*m[nxx]-2]=0.5
    row=row+1
    M[row,p[mxx]+(j+1)*m[mxx]-1]=1.0
    row=row+1
    M[row,p[vx ]+(j+1)*m[vx ]-1]=0.5
    M[row,p[vx ]+(j+1)*m[vx ]-2]=0.5

    i=m[mxy]-1
    Dy(mxy,1.0, i)
for j in range(n[mxy]):
    row=row+1
    M[row,p[nxy]+(j+1)*m[nxy]-1]=0.5
    M[row,p[nxy]+(j+1)*m[nxy]-2]=0.5
    i_end = mm-1
    M[row,p[mxy]+(j+1)*m[mxy]-1]= k_yy[i_end]

for i in range(mm): # Close straight edge
    row=row+1
    M[row,p[nyy]+0*m[nyy]+i]=0.5
    M[row,p[nyy]+1*m[nyy]+i]=0.5
    row=row+1
    M[row,p[myy]+0*m[myy]+i]=1.0
    row=row+1
    M[row,p[vy ]+0*m[vy ]+i]=0.5
    M[row,p[vy ]+1*m[vy ]+i]=0.5
    j=0
    Dx(mxy,1.0)
for i in range(m[nyx]):
    row=row+1
    M[row,p[nyx]+0*m[nyx]+i]=0.5
    M[row,p[nyx]+1*m[nyx]+i]=0.5


for i in range(mm): # Far straight edge
    row=row+1
    M[row,p[nyy]+(n[nyy]-1)*m[nyy]+i]=0.5
    M[row,p[nyy]+(n[nyy]-2)*m[nyy]+i]=0.5
    if i==(mm-1):
      f[row]=F*2.0*(mm-1)/l
    row=row+1
```

```
    M[row,p[myy]+(n[myy]-1)*m[myy]+i]=1.0
    row=row+1
    M[row,p[vy ]+(n[vy ]-1)*m[vy ]+i]=0.5
    M[row,p[vy ]+(n[vy ]-2)*m[vy ]+i]=0.5
    j=nn
    Dx(mxy,1.0)
for i in range(m[nyx]):
    row=row+1
    M[row,p[nyx]+(n[nyx]-1)*m[nyx]+i]=0.5
    M[row,p[nyx]+(n[nyx]-2)*m[nyx]+i]=0.5

n_equations=row+1
print('Check: Number of unknown is ', n_unknown,'. Number of equations is ', n_equations,'.')

print('Solve the matrix.')
M=M.tocsr()
u=spsolve(M,f)
del M
del f

print('Plot results.')
title=['ux','uy','uz','epsilonxx','epsilonyy','gammaxy','phix','phiy','phiz','kappaxx','kappayy','
    rhoxy','nxx','nyy','nxy','nyx','mxx','myy','mxy','vx','vy']
for k in range(21):
    X= [i/(m[k]-1)*l for i in range(m[k])]
    Y= [j/(n[k]-1)*np.pi for j in range(n[k])]

    Z=[[u[p[k]+j*m[k]+i] for i in range(m[k])] for j in range(n[k])]
    plt.figure()
    cp=plt.contourf(X,Y,Z,10,cmap=plt.cm.jet)
    plt.colorbar(cp)
    plt.title(title[k])
    plt.xlabel('x (length)')
    plt.ylabel('theta (angle in radian)')
    plt.show()

print('Done.')
```

A.2. Scenario B adjustments

This scenario B models the situation of the conical shell with a full surface load of $F = 5kN/m^2$

```

q_vlak = 5.0


for j in range(nn): # Eq_x: 0 = dnxx/dx + dnyx/dy + ky(nxx-nyy) + px
    for i in range(mm):
        row=row+1
        Dx(nxx,1.0)
        h_y = np.pi*Radius[i]/(nn-1)


        if i==0:
            M[row,p[nyx]+(j  )*m[nyx]+i  ]=-1.0/h_y
            M[row,p[nyx]+(j+1)*m[nyx]+i  ]= 1.0/h_y
        elif i==mm-1:
            M[row,p[nyx]+(j  )*m[nyx]+i-1]=-1.0/h_y
            M[row,p[nyx]+(j+1)*m[nyx]+i-1]= 1.0/h_y
        else:
            M[row,p[nyx]+(j  )*m[nyx]+i  ]=-0.5/h_y
            M[row,p[nyx]+(j+1)*m[nyx]+i  ]= 0.5/h_y
            M[row,p[nyx]+(j  )*m[nyx]+i-1]=-0.5/h_y
            M[row,p[nyx]+(j+1)*m[nyx]+i-1]= 0.5/h_y


        if i > 0 and i < mm-1:
            M[row, p[nxx] + (j  )*m[nxx] + i  ] += 0.5 * k_y[i]
            M[row, p[nxx] + (j  )*m[nxx] + i-1] += 0.5 * k_y[i]
            M[row, p[nyy] + (j  )*m[nyy] + i  ] -= 0.5 * k_y[i]
            M[row, p[nyy] + (j  )*m[nyy] + i-1] -= 0.5 * k_y[i]


        theta = j * np.pi / (nn - 1)
```

```
        px_local = -q_vlak * np.sin(alpha) * np.sin(theta)
        f[row] = -px_local
        # ------------------------------------------

for j in range(nn): # Eq_y: 0 = dnyy/dy + dnxy/dx + ky(nxy+nyx) - kyy*vy + py
    for i in range(1,mm):
        row=row+1
        Dy(nyy,1.0, i)
        h_x=l/(mm-1)
        if j==0:
            M[row,p[nxy]+(j  )*m[nxy]+i-1]=-1.0/h_x
            M[row,p[nxy]+(j  )*m[nxy]+i  ]= 1.0/h_x
        elif j==nn-1:
            M[row,p[nxy]+(j-1)*m[nxy]+i-1]=-1.0/h_x
            M[row,p[nxy]+(j-1)*m[nxy]+i  ]= 1.0/h_x
        else:
            M[row,p[nxy]+(j  )*m[nxy]+i-1]=-0.5/h_x
            M[row,p[nxy]+(j  )*m[nxy]+i  ]= 0.5/h_x
            M[row,p[nxy]+(j-1)*m[nxy]+i-1]=-0.5/h_x
            M[row,p[nxy]+(j-1)*m[nxy]+i  ]= 0.5/h_x


        if j > 0 and j < nn-1:
            M[row,p[nxy]+(j  )*m[nxy]+i  ] += 0.5 * k_y[i]
            M[row,p[nxy]+(j  )*m[nxy]+i-1] += 0.5 * k_y[i]
            M[row,p[nyx]+(j  )*m[nyx]+i  ] += 0.5 * k_y[i]
            M[row,p[nyx]+(j+1)*m[nyx]+i  ] += 0.5 * k_y[i]
        M[row,p[vy]+(j  )*m[vy]+i] -= 0.5 * k_yy[i]
        M[row,p[vy]+(j+1)*m[vy]+i] -= 0.5 * k_yy[i]

        theta = j * np.pi / (nn - 1)
        py_local = -q_vlak * np.cos(theta)
        f[row] = -py_local
        # ------------------------------------------------------------


for j in range(nn): # Eq_z: 0 = kyy*nyy + dvx/dx + dvy/dy + ky*vx + pz
    for i in range(mm):
        row=row+1
        M[row,p[nyy]+(j  )*m[nyy]+i] += 0.5 * k_yy[i]
        M[row,p[nyy]+(j+1)*m[nyy]+i] += 0.5 * k_yy[i]
        Dx(vx, 1.0)
        Dy(vy, 1.0, i)

        if i > 0 and i < mm-1:
            M[row,p[vx]+(j  )*m[vx]+i  ] += 0.5 * k_y[i]
            M[row,p[vx]+(j  )*m[vx]+i-1] += 0.5 * k_y[i]

        theta = j * np.pi / (nn - 1)
        pz_local = -q_vlak * np.cos(alpha) * np.sin(theta)
        f[row] = -pz_local
```

A.3. Scenario C adjustments

This scenario C analyzes the influence of a pinned support at the other edge of the structure.

```

print('Insert the boundary conditions.')
for j in range(nn): # Fixed edge
    row=row+1
    M[row,p[ux  ]+j*m[ux  ]+0]=1.0
    row=row+1
    M[row,p[uy  ]+j*m[uy  ]+0]=1.0
    row=row+1
    M[row,p[uz  ]+j*m[uz  ]+0]=1.0
    row=row+1
    M[row,p[phix]+j*m[phix]+0]=0.5
    M[row,p[phix]+j*m[phix]+1]=0.5



for j in range(nn): # Hinged edge
    row=row+1
```

```
    M[row,p[ux  ]+j*m[ux  ]+ (mm-1)]=1.0
    row=row+1
    M[row,p[uz  ]+j*m[uz  ]+(mm-1)]=1.0
    row=row+1
    M[row,p[mxx ]+j*m[mxx ]+(mm-1)]=1.0

for j in range(n[mxy]):
    row=row+1
    M[row,p[uy  ]+j*m[uy  ]+(mm-1)]=1.0


for i in range(mm): # Close straight edge
    row=row+1
    M[row,p[nyy]+0*m[nyy]+i]=0.5
    M[row,p[nyy]+1*m[nyy]+i]=0.5
    row=row+1
    M[row,p[myy]+0*m[myy]+i]=1.0
    row=row+1
    M[row,p[vy ]+0*m[vy ]+i]=0.5
    M[row,p[vy ]+1*m[vy ]+i]=0.5
    j=0
    Dx(mxy,1.0)
for i in range(m[nyx]):
    row=row+1
    M[row,p[nyx]+0*m[nyx]+i]=0.5
    M[row,p[nyx]+1*m[nyx]+i]=0.5


for i in range(mm): # Far straight edge
    row=row+1
    M[row,p[nyy]+(n[nyy]-1)*m[nyy]+i]=0.5
    M[row,p[nyy]+(n[nyy]-2)*m[nyy]+i]=0.5
    row=row+1
    M[row,p[myy]+(n[myy]-1)*m[myy]+i]=1.0
    row=row+1
    M[row,p[vy ]+(n[vy ]-1)*m[vy ]+i]=0.5
    M[row,p[vy ]+(n[vy ]-2)*m[vy ]+i]=0.5
    j=n[mxy]
    Dx(mxy,1.0)
for i in range(m[nyx]):
    row=row+1
    M[row,p[nyx]+(n[nyx]-1)*m[nyx]+i]=0.5
    M[row,p[nyx]+(n[nyx]-2)*m[nyx]+i]=0.5
```

A.4. Scenario D adjustments

This model introduces an asymmetrical load on the shell surface with one clamped edge.

```


q_vlak = 5.0


for j in range(nn): # Eq_x
    for i in range(mm):
        row=row+1
        Dx(nxx,1.0)
        h_y = np.pi*Radius[i]/(nn-1)


        if i==0:
            M[row,p[nyx]+(j  )*m[nyx]+i  ]=-1.0/h_y
            M[row,p[nyx]+(j+1)*m[nyx]+i  ]= 1.0/h_y
        elif i==mm-1:
            M[row,p[nyx]+(j  )*m[nyx]+i-1]=-1.0/h_y
            M[row,p[nyx]+(j+1)*m[nyx]+i-1]= 1.0/h_y
        else:
            M[row,p[nyx]+(j  )*m[nyx]+i  ]=-0.5/h_y
            M[row,p[nyx]+(j+1)*m[nyx]+i  ]= 0.5/h_y
            M[row,p[nyx]+(j  )*m[nyx]+i-1]=-0.5/h_y
            M[row,p[nyx]+(j+1)*m[nyx]+i-1]= 0.5/h_y

        # SK termen voor Eq_x
```

```
        if i > 0 and i < mm-1:
            M[row, p[nxx] + (j  )*m[nxx] + i  ] += 0.5 * k_y[i]
            M[row, p[nxx] + (j  )*m[nxx] + i-1] += 0.5 * k_y[i]
            M[row, p[nyy] + (j  )*m[nyy] + i  ] -= 0.5 * k_y[i]
            M[row, p[nyy] + (j  )*m[nyy] + i-1] -= 0.5 * k_y[i]

        theta = j * np.pi / (nn - 1)
        if theta >= 0.5 * np.pi:
            px_local = -q_vlak * np.sin(alpha) * np.sin(theta)
        else:
            px_local = 0.0

        f[row] = -px_local



for j in range(nn):
    for i in range(1,mm):
        row=row+1
        Dy(nyy,1.0, i)
        h_x=1/(mm-1)
        if j==0:
            M[row,p[nxy]+(j  )*m[nxy]+i-1]=-1.0/h_x
            M[row,p[nxy]+(j  )*m[nxy]+i  ]= 1.0/h_x
        elif j==nn-1:
            M[row,p[nxy]+(j-1)*m[nxy]+i-1]=-1.0/h_x
            M[row,p[nxy]+(j-1)*m[nxy]+i  ]= 1.0/h_x
        else:
            M[row,p[nxy]+(j  )*m[nxy]+i-1]=-0.5/h_x
            M[row,p[nxy]+(j  )*m[nxy]+i  ]= 0.5/h_x
            M[row,p[nxy]+(j-1)*m[nxy]+i-1]=-0.5/h_x
            M[row,p[nxy]+(j-1)*m[nxy]+i  ]= 0.5/h_x

        #Eq_y
        if j > 0 and j < nn-1:
            M[row,p[nxy]+(j  )*m[nxy]+i  ] += 0.5 * k_y[i]
            M[row,p[nxy]+(j  )*m[nxy]+i-1] += 0.5 * k_y[i]
            M[row,p[nyx]+(j  )*m[nyx]+i  ] += 0.5 * k_y[i]
            M[row,p[nyx]+(j+1)*m[nyx]+i  ] += 0.5 * k_y[i]
        M[row,p[vy]+(j  )*m[vy]+i] -= 0.5 * k_yy[i]
        M[row,p[vy]+(j+1)*m[vy]+i] -= 0.5 * k_yy[i]


        theta = j * np.pi / (nn - 1)
        if theta >= 0.5 * np.pi:
            py_local = -q_vlak * np.cos(theta)
        else:
            py_local = 0.0

        f[row] = -py_local
        # ------------------------------------------------------------


for j in range(nn): # Eq_z
    for i in range(mm):
        row=row+1
        M[row,p[nyy]+(j  )*m[nyy]+i] += 0.5 * k_yy[i]
        M[row,p[nyy]+(j+1)*m[nyy]+i] += 0.5 * k_yy[i]
        Dx(vx, 1.0)
        Dy(vy, 1.0, i)

        if i > 0 and i < mm-1:
            M[row,p[vx]+(j  )*m[vx]+i  ] += 0.5 * k_y[i]
            M[row,p[vx]+(j  )*m[vx]+i-1] += 0.5 * k_y[i]


        theta = j * np.pi / (nn - 1)
        if theta >= 0.5 * np.pi:
            pz_local = -q_vlak * np.cos(alpha) * np.sin(theta)
        else:
            pz_local = 0.0

        f[row] = -pz_local
```

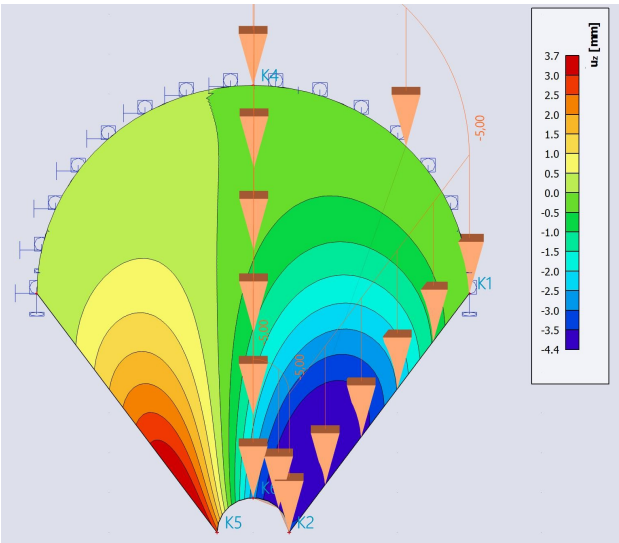
B

Contour plots scenario D

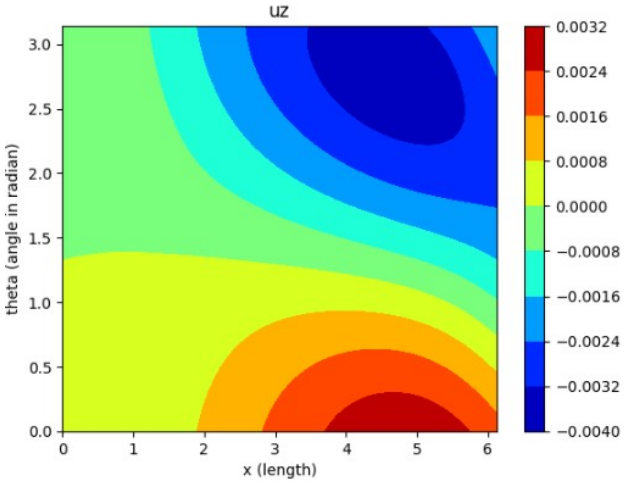
This appendix shows the numerical results for the displacements (u_z and u_y) and the torsion moment (m_{xy}). For the normal displacement u_z and the circumferential displacement u_y, the absolute maximum values are included in the table. For the twisting moment m_{xy} the evaluation was conducted at the coordinate $x = 5$ and $\theta = 0.7$ rad. The analysis of the FDM model showed that the numerical difference between the in-plane shear forces n_{xy} and n_{yx} is largest at this exact point.

Table B.1: Comparison of peak absolute structural responses for Scenario D.

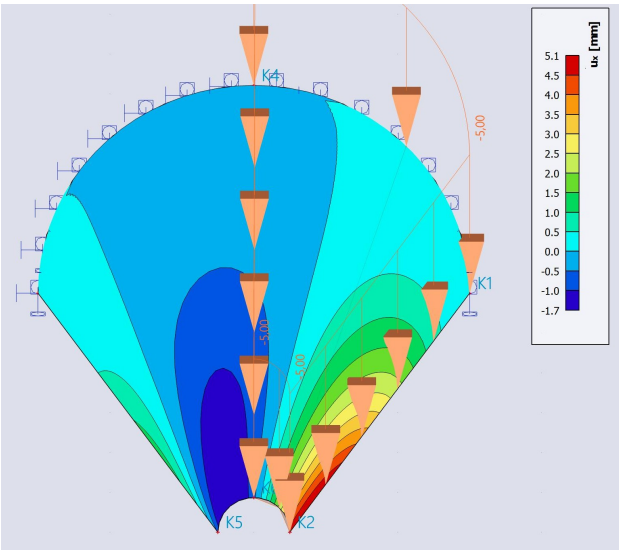
Parameter	**SCIA (FEM)**	**Python (FDM)**	**Difference (%)**
Max. Normal $\lvert u_z \rvert$	4.4 mm	3.7 mm	18.9%
Max. Circumferential $\lvert u_y \rvert$	5.1 mm	4.1 mm	24%
Torsion Moment $\lvert m_{xy} \rvert$	0.42 kNm/m	0.32 kNm/m	31.3%



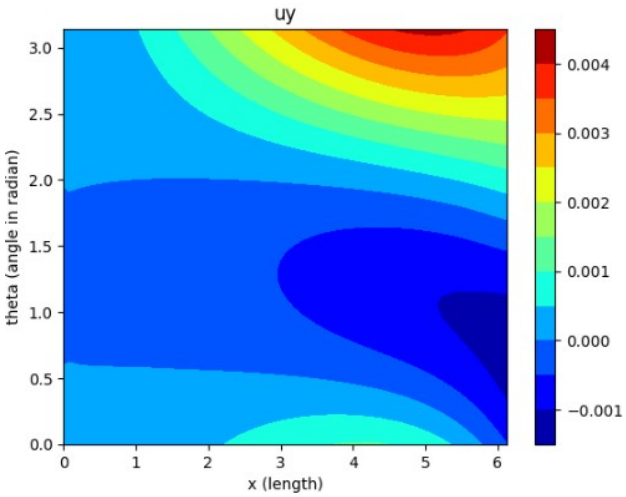
(a) SCIA (FEM): Normal displacement u_z



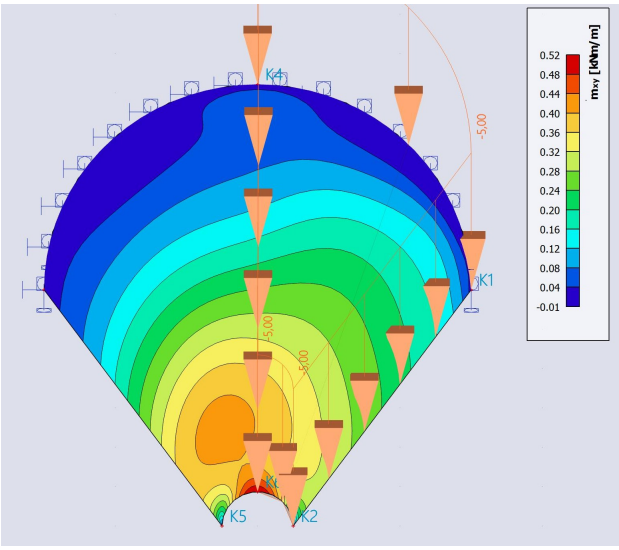
(b) Python (FDM): Normal displacement u_z



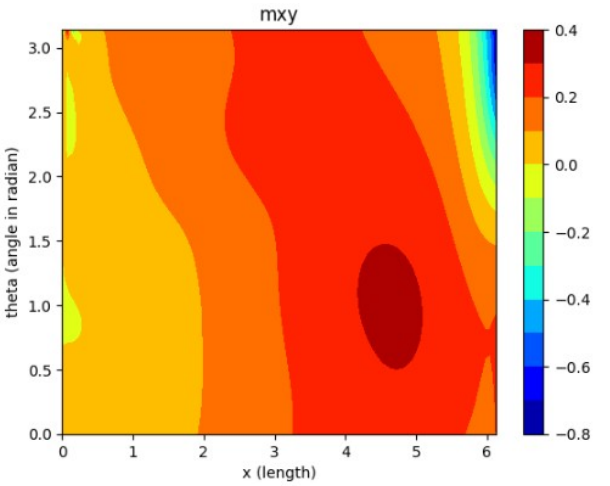
(c) SCIA (FEM): Circumferential displacement u_y



(d) Python (FDM): Circumferential displacement u_y



(e) SCIA (FEM): Torsion moment m_{xy}



(f) Python (FDM): Torsion moment m_{xy}

Figure B.1: Left: 3D Isometric view of the FEM results. Right: Unrolled 2D contour plot ($x - \theta$ domain) of the FDM results.