Quantifying The Influence Of Membrane Forces, Curvature, And Imperfections On The Nonlinear Buckling Load Of Thin-Shells

Version of December 8th, 2017

Erik J. Giesen Loo

Quantifying The Influence Of Membrane Forces, Curvature, And Imperfections On The Nonlinear Buckling Load Of Thin-Shells

by

Erik Johannes Giesen Loo
Student number: 4625064

Additional Master Thesis

Submitted in partial fulfillment of the
requirements for the degree of

Master Of Science

in

Civil Engineering

at

Delft University of Technology

Supervisors:

Dr. ir. P.C.J. Hoogenboom
Dr. ir. drs. C.R. Braam

Department of Structural Mechanics
Faculty of Civil Engineering and Geoscience
Delft University of Technology
Delft, The Netherlands

To Maximilian E. Ororbia

QUANTIFYING THE INFLUENCE OF MEMBRANE FORCES, CURVATURES, AND IMPERFECTIONS ON THE NONLINEAR BUCKLING LOAD OF THIN-SHELLS

Author:	Erik Johannes Giesen Loo
Student I.D.:	4625064
Email	E.J.Giesen@student.tudelt.nl

ABSTRACT

Shells tend to be thin because their curvature enables them to carry distributed loads as membrane forces. The property of thinness stems from shells' capacity to store membrane strain energy without much deformation. As a result, buckling failures often govern the design of shell structures. These buckling failures usually start locally, at a location where a combination of curvature and membrane forces is met. Moreover, shells tend to be imperfection-sensitive structures, that is, real-life shells (with initial geometric imperfections) usually cannot resist loads as high as the theoretically predicted critical buckling load. Advanced finite element analyses can accurately predict these so-called nonlinear buckling loads but require significant time and computation effort. On the other hand, current design equations are simple yet highly inaccurate and often penalize strength significantly. This treatise caters a Python script that executes nonlinear finite element analyses (using ANSYS Mechanical APDL) to generate a database of the nonlinear buckling loads of shell portions with varying membrane forces, curvatures and magnitudes of initial geometric imperfections. The aim, beyond the scope of this treatise, is to perform a parametric regression on said database to device design equation(s) that accurately predict the nonlinear buckling load of linear-elastic shell structures with initial geometric imperfections based merely on the linear elastic results of a geometrically perfect shell model.

Supervisors:

Dr. ir. P.C.J. Hoogenboom
Dr. ir. drs. C.R. Braam

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1. Thesis Statement

Shells tend to be thin because their curvature enables them to carry distributed loads as membrane forces. As a result, buckling failure often governs the design of shell structures. Unlike frames, shell buckling starts locally, at a location where a combination of curvature and membrane forces is met. Shell analysis is difficult because the curvature is changed by the initial geometric imperfections and the membrane forces. Additionally, the membrane forces are strongly influenced by the changing curvature.

While shell buckling can be predicted accurately using advanced finite element analyses, current design equations are not accurate. For instance, the influence of lateral curvature, lateral normal force and initial geometric imperfections are generally unknown. This lack of knowledge means these equations penalize strength significantly. More accurate design equations based on a parametric study would alleviate this.

1.2. Research Purpose

The objective of this treatise is to lay the foundation for subsequent research by providing a Python script capable of executing a batch of finite element analyses (using ANSYS Mechanical APDL) that determine the nonlinear buckling loads of shell portions with varying curvatures, membrane forces and initial geometric imperfections. This script would generate a database from which design equation(s) can be derived to predict the nonlinear buckling load of a shell portion based merely on the original curvature and linear-elastic membrane forces of a geometrically perfect shell model. This task falls beyond the scope of the current treatise. The nonlinear buckling loads are obtained by performing geometrically nonlinear analyses with initial geometric imperfections (GNIA). Physical nonlinearities are not accounted for.

1.3. Thesis Overview

This treatise is divided in 5 chapters which, save for this brief introduction, are outlined here.

Chapter 2 summarizes the background knowledge. This encompasses the linear elastic theory, finite element method implementation, shell buckling and postbuckling theory, and finite element solution techniques.

Chapter 3 presents the methodology, paying special attention to the ANSYS Mechanical APDL and Python code provided in the appendices.

Chapter 4 accommodates an analytical solution for the linear elastic membrane forces of an externally pressurized thin-shell torus, its buckling modes, as well as sample results from the Python script.

Chapter 5 evaluates potential shortcomings in the code and suggests a course of action for continuing the research upon the foundations laid herein, especially the generation of the database and parametric regression.

2. BACKGROUND

2.1. Shell Linear Elastic Theory

In this section, the coordinates and curvatures are defined and the Lamé parameters are introduced. The shell membrane theory is covered briefly. Shells can be described as plates with a curved middle surface. This curvature enables shells to carry out-of-plane pressure loads as membrane forces instead of bending moments (Blaauwendraad & Hoefakker, 2013, pp.1-2; Voyiadjis & Woelke, 2008, p. 1). The latter are restricted to so-called edge disturbances found at concentrated loads, edges and discontinuities. (Voyiadjis & Woelke, 2008, p. 1; Hoogenboom, 2017, p. 65).

Assuming the existence of an orthogonal parametrization $\mathrm{r}(u,v)=\lfloor \bar{x}(u,v) \quad \bar{y}(u,v) \quad \bar{z}(u,v)\rfloor^{\mathrm{T}}$, three coordinate systems can be defined: global, local and curvilinear (Hoogenboom, 2017, p.29). The global coordinate system $(\bar{x},\bar{y},\bar{z})$ describes the shell geometry. The curvilinear and local systems are based on $\mathrm{r}(u,v)$. The curvilinear *x* and *y* axes follow the *u* and *v* parameter lines, i.e., $\mathrm{r}(u,v)$ with increasing *u* and *v*, respectively. The local *x* and *y* axes are tangential to their curvilinear counterparts. Both local and curvilinear systems' *z* axis are orthogonal to the *x* and *y* axes, and point in the same direction. The local coordinate system is used to define curvatures, loading, internal force resultants and displacements.

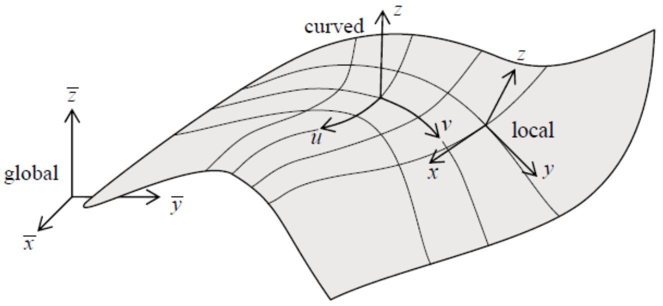


Figure 2.1. Shell coordinates (Hoogenboom, 2017, p. 29)

Several definitions of curvature are considered. The shell surface curvature is described by two normal section curvatures, k_{xx} and k_{yy}, and a surface twist, k_{xy}. The normal section curvatures are the inverse of the radii of the circles tangential to the shell on the local *x*-*z* and *y*-*z* planes. They are found using Eq. (2.1) and (2.2); the twist is given by Eq. (2.3) (Hoogenboom, 2017, p. 19):

$$k_{xx} = \partial^2 z / \partial x^2 \ , \tag{2.1}$$

$$k_{yy} = \partial^2 z / \partial y^2 \tag{2.2}$$

and

$$k_{xy} = \partial^2 z / \partial x \partial y \ . \tag{2.3}$$

The shell curvatures form a tensor which can be transformed to yield the principal curvatures; or Eq. (2.4) can be used (Hoogenboom, 2017, p. 21):

$$k_{1,2} = \frac{1}{2}\left(k_{xx} + k_{yy}\right) \pm \sqrt{\frac{1}{4}\left(k_{xx} - k_{yy}\right)^2 + k_{xy}{}^2}\,. \tag{2.4}$$

The curvature tensor has two invariants, namely the Gauss curvature, k_G, and mean curvature, k_m (Hoogenboom, 2017, pp. 23-24):

$$k_G = k_1 k_2 = k_{xx}k_{yy} - k_{xy}^2 \tag{2.5}$$

and

$$k_m = \frac{1}{2}\left(k_1 + k_2\right) = \frac{1}{2}\left(k_{xx} + k_{yy}\right). \tag{2.6}$$

Additionally, the k_x and k_y in-plane curvatures are defined as the inverse of the r_y and r_x radii of the circles in the plane normal to the local z-axis tangential to u and v (Hoogenboom, 2017, p. 35):

$$k_x = \frac{1}{\alpha_x}\frac{\partial \alpha_x}{\partial y} \tag{2.7}$$

and

$$k_y = \frac{1}{\alpha_y}\frac{\partial \alpha_y}{\partial x}\,. \tag{2.8}$$

The Lamé parameters, α_x and α_y, map the ratio of change in x with respect to a change in u and a change in y with respect to a change in v (Hoogenboom, 2017, pp. 30-31), i.e.,

$$\begin{Bmatrix} dx \\ dy \end{Bmatrix} = \begin{bmatrix} \alpha_x & 0 \\ 0 & \alpha_y \end{bmatrix} \begin{Bmatrix} du \\ dv \end{Bmatrix}. \tag{2.9}$$

If the shell is parametrized, the Lamé parameters can be used derive k_{xx}, k_{yy} and k_{xy}. Figure 2.2 shows the normal section curvature and in-plane curvatures.

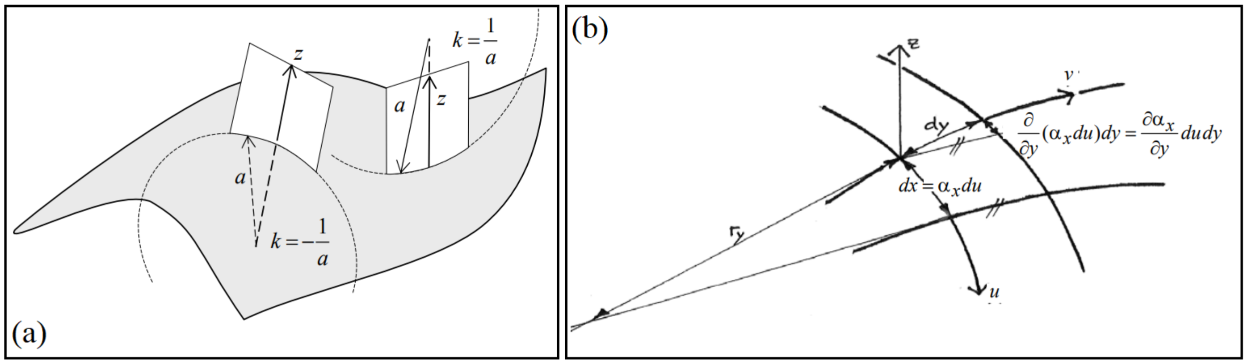

$k = \frac{1}{a}$
z
a
z
a
$k = -\frac{1}{a}$
(a)
(b)
z
v
dy
$\frac{\partial}{\partial y}(\alpha_x du)dy = \frac{\partial \alpha_x}{\partial y} du dy$
$dx = \alpha_x du$
r_y
u

Figure 2.2. (a) Normal section curvature (Hoogenboom, 2017, p.19), and (b) In-plane curvature (Hoogenboom, 2017, p.35)

The geometry of a shell is described by its thickness and surface curvature. Thus, shells can be categorized based on their radius-to-thickness ratio (Blaauwendraad & Hoefakker, 2014, p.3). Table 2.1 shows such a classification, with suitable theories (Hoogenboom, 2017, pp. 13-14).

Table 2.1. Shell types and corresponding theory based on radius *a* and thickness *t*

Type of shell	**Slenderness**	**Theory**
Very thick shell	$a/t < 5$	Solid elements (i.e., not a shell)
Thick shell	$5 < a/t < 30$	Mindlin-Reissner (includes shear deformations)
Thin-shell	$30 < a/t < 4000$	Sanders-Koiter (membrane forces and moments)
Membrane	$4000 < a/t$	Shell membrane (only membrane forces)

As seen in Table 2.1, the analysis of thin shells involves two distinct theories: the shell membrane theory, which does not include bending and shear; and the Sanders-Koiter theory, which includes bending deformations and shear stresses but not shear deformations (Voyiadjis & Woelke, 2008, p. 2). The positive internal force resultants from the latter theory are shown in Fig. 2.3.

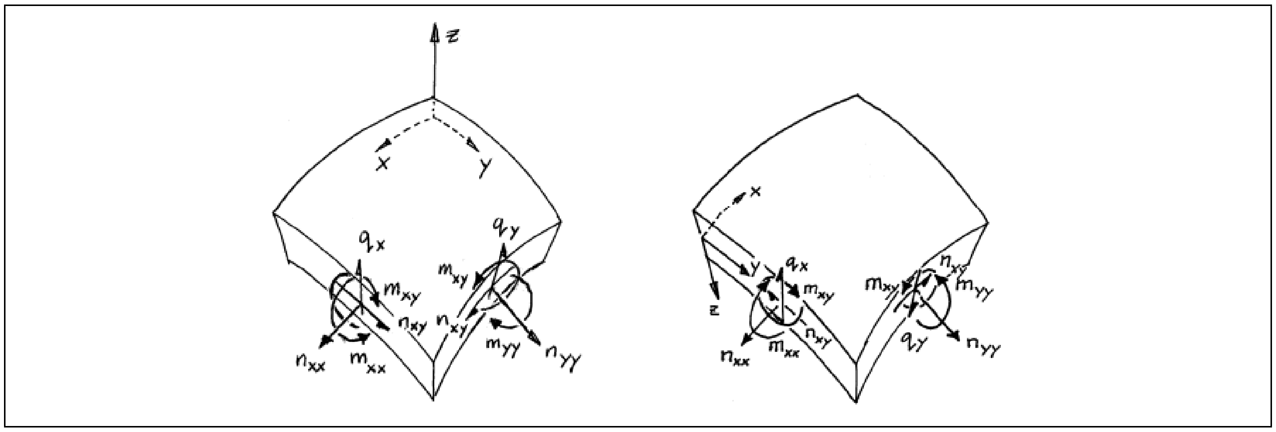


Figure 2.3. Positive internal forces (Hoogenboom, 2017, p.13)

Even though the Sanders-Koiter equations offer a more faithful representation of the internal forces of thin shell structures, membrane stresses are more important for practical purposes (Voyiadjis & Woelke, 2008, p.2). The shell membrane equations are also simpler to solve analytically. Such a solution is provided in section 4.1 to verify the finite element method results. This solution required only the shell membrane equilibrium equations shown next (Hoogenboom, 2017, p.35-36; see also Blaauwendraad & Hoefakker, 2013, p. 27):

$$\partial n_{xx}/\partial x + \partial n_{xy}/\partial y + k_y(n_{xx} - n_{yy}) + 2k_x n_{xy} + p_x = 0, \tag{2.10}$$

$$\partial n_{yy}/\partial y + \partial n_{xy}/\partial x + k_x(n_{yy} - n_{xx}) + 2k_y n_{xy} + p_y = 0 \tag{2.11}$$

and

$$k_{xx} n_{xx} + 2k_{xy} n_{xy} + k_{yy} n_{yy} + p_z = 0. \tag{2.12}$$

The reader is referred to Hoogenboom (2017), Blaauwendraad & Hoefakker (2013), and Voyiadjis & Woelke (2008) for a comprehensive coverage of the linear elastic theories and equations.

2.1.1. Finite Element Implementation

The three most common shell finite elements are: flat shell elements, elements based on the Sanders-Koiter equations and reduced solid elements (Hoogenboom, 2017, p. 65). Flat shell elements combine plane stress with plate bending and drilling degrees of freedom (Hoogenboom 2017, p.65; Blaauwendraad & Hoefakker, 2013, p. 285). Their main disadvantage is that, by being flat, a fine mesh is required to preserve the curvature of the shell model (Chen, 2014, p. 12).

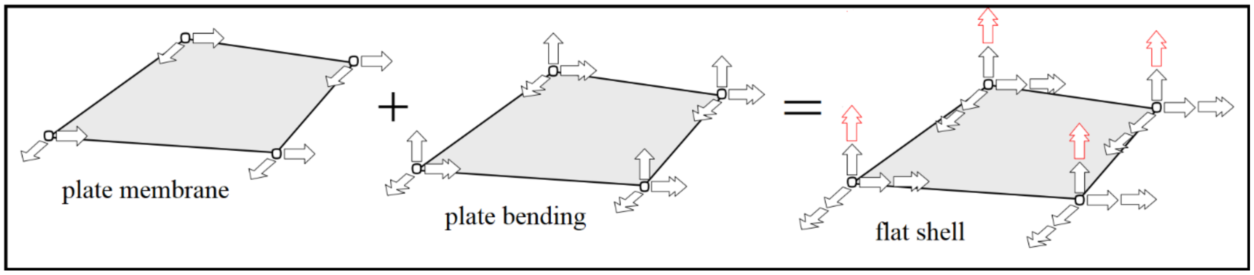


Figure 2.4. Flat shell element (Hoogenboom, 2017, p. 65)

Shell elements based on the Sanders-Koiter equations, such as the semi-loof element (Fig. 2.5), can be very accurate but are often difficult to implement in finite element software (Hoogenboom, 2017, p.66).

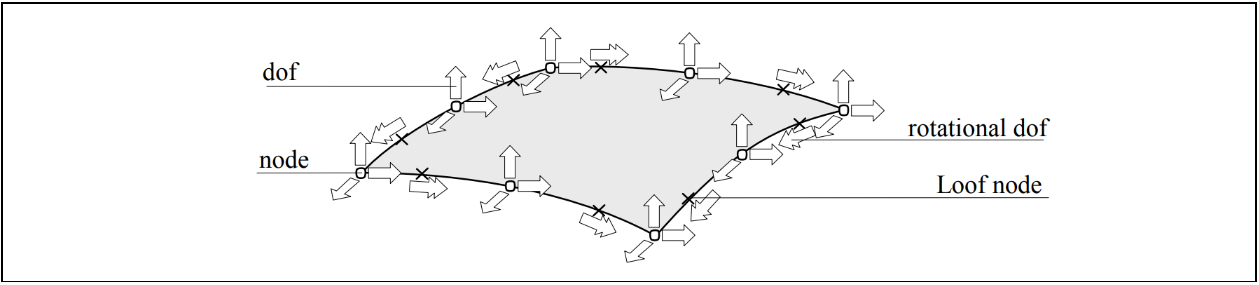


Figure 2.5. Semi-loof element (Hoogenboom, 2017, p. 66)

The most common element type is the reduced solid element (Fig. 2.6) in which degrees of freedom are combined and the constitutive equations are simplified (Hoogenboom, 2017, p.66; Blaauwendraad & Hoefakker, 2013, p. 286). 8-noded quadrilaterals can be curved, which reduces the need for fine meshes. The script provided in App. B uses this shell element to generate the shell models.

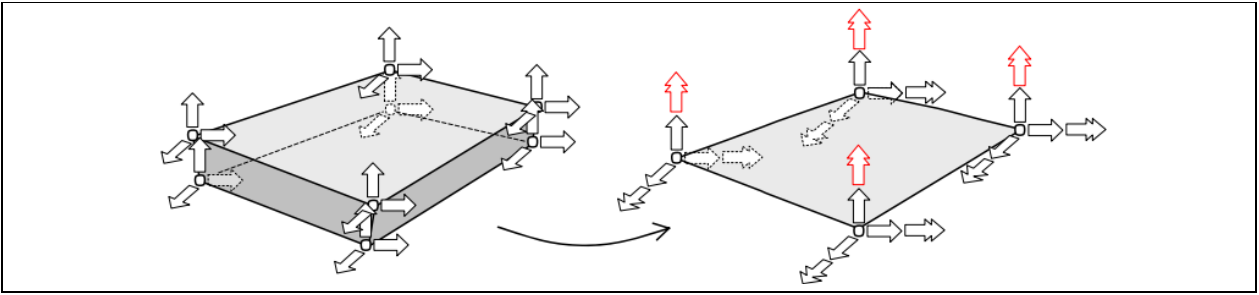


Figure 2.6. Reduced solid element (Hoogenboom, 2017, p. 66)

2.2. Shell Buckling Theory

Shells tend to be thin because their curvature enables them to carry distributed loads as membrane forces. The property of thinness stems from shells' capacity to store membrane strain energy without much deformation. Yet, if this energy is converted into bending energy, shells may become statically unstable and fail dramatically (Bushnell, 1981, p.1187).

2.2.1. Static Instability

Static instability, loosely termed buckling, is the condition when a structural member or system exhibits a loss in its load-carrying capacity (Ziemian, 2010, p. 12). Buckling may be divided into two categories: 1) bifurcation of equilibrium (Fig. 2.7, point B) and 2) collapse at the limit load without prior bifurcation (point A). Bifurcation is exemplified by a sudden change in the load-carrying path, e.g., from axial (or membrane) forces to bending moments, and corresponding deformations. Columns, plates and cylindrical shells experience this type of instability. Shallow arches and spherical caps are examples of the second type of instability, also termed nonlinear buckling or "snap-through" (Ziemian, 2010, p.12; Bushnell, 1981, p. 1183-1187). However, given initial geometric imperfections, even arches and spherical caps are prone to fail in an asymmetric mode due to bifurcation prior to their limit load, i.e., curve 0-B-D in Fig. 2.7 (Ziemian, 2010, p. 22; Hutchinson & Koiter, 1970, p. 1354; Bushnell, 1981, p. 1187).

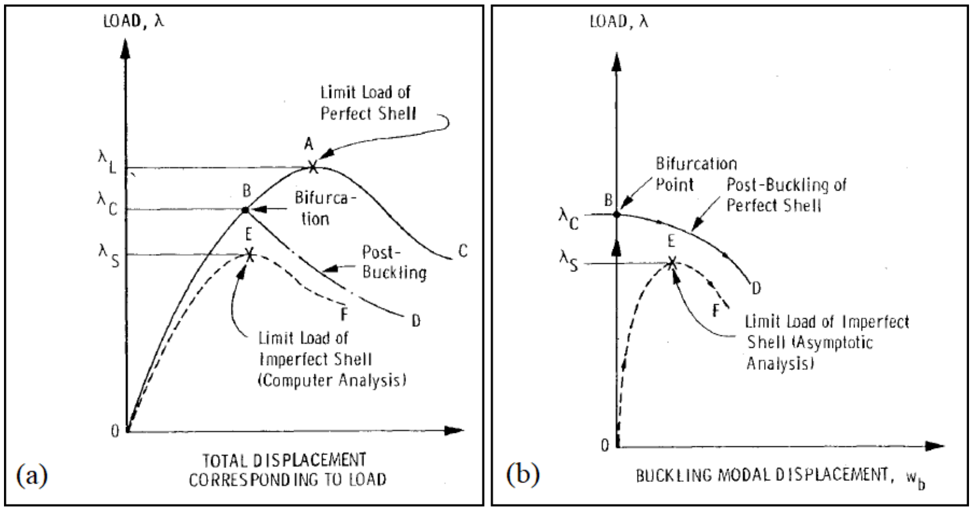


Figure 2.7. Load-deflection curves showing limit and bifurcation points: (a) General nonlinear analysis, and (b) Asymptotic analysis (Bushnell, 1981, p. 1187)

The loads observed in Fig. 2.7 are expressed as a multiplier λ to some reference load. λ_C is the critical buckling load ratio at the bifurcation point. λ_L, or limit load ratio, is the maximum load that can be achieved without prior bifurcation. λ_S is the maximum load that can be achieved by a structure with initial geometric imperfections before static instability is reached (Hutchinson & Koiter, 1970, p. 1354). Chapter 1 refers to λ_S as the nonlinear buckling load because – just like for the limit load ratio – λ_S is obtained by means of a geometrically nonlinear analysis (GNA). However, estimating λ_S requires explicit modeling of the initial geometric imperfections in the finite element model. An analysis that includes such imperfections is referred to as a geometrically nonlinear analysis with initial geometric imperfections (GNIA).

2.2.2. Bifurcation Buckling

The critical buckling load for discretized systems is the lowest eigenvalue from Eq. (2.22). It can also be obtained analytically for elementary shell types by solving a reduced eighth-order differential equation (Blaauwendraad & Hoefakker, 2013, p. 81). Equation (2.13) shows the theoretical buckling membrane force of an axially loaded thin-shell cylinder (Bushnell, 1981, p. 1189):

$$n_C = -\left[3(1-\nu^2)\right]^{-1/2} \frac{Et^2}{a} \cong -0.6 \frac{Et^2}{a} . \tag{2.13}$$

$\left[3(1-\nu^2)\right]^{-1/2}$ is approximated as 0.6 for realistic values of ν. Equation (2.13) is also valid for axially loaded hyperboloids and for externally pressurized closed cylinders, spherical shells, domes, and hyperbolic paraboloids (Hoogenboom, 2017, p. 111). The fact Eq. (2.13) makes no reference to the number of waves found in the buckling pattern helps to explain this wide range of applicability (Bushnell, 1981, p. 1190). This quality has further repercussions seen in subsection 2.2.4 which describes methods that estimate λ_S based on asymptotic analyses that rest on the theoretical foundations established by Koiter and use the postbuckling behavior as starting point (Hutchinson & Koiter, 1970, p. 1354).

2.2.3. Postbuckling Behavior

While relatively simple to solve, neither Eq. (2.22) nor Eq. (2.13) yield information on the postbuckling behavior (Hutchinson & Koiter, 1970, p. 1354), and thus also not about the stability of a structure after bifurcation (Bushnell, 1981, p. 1193). A general understanding of the postbuckling behavior of shells can be obtained by considering the simple model in Fig. 2.8, like the one proposed by Cox (1940, p. 231).

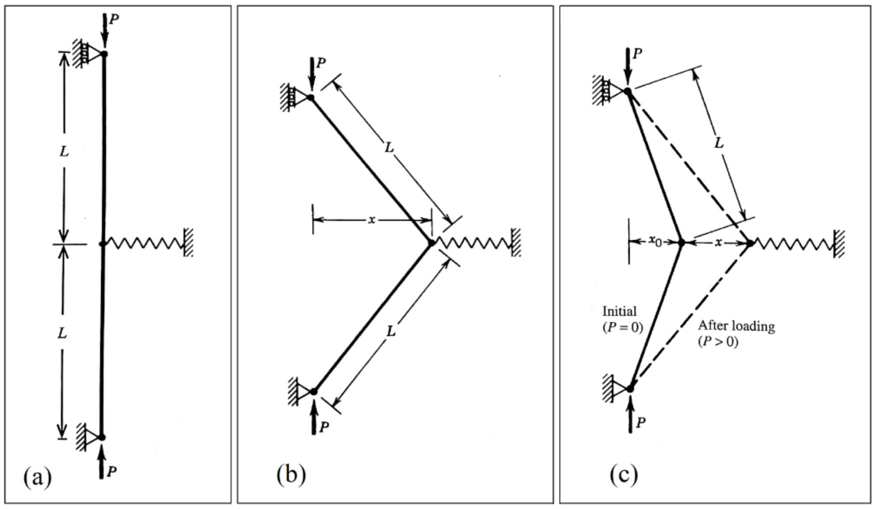


Figure 2.8. Bifurcation buckling of initially perfect model: (a) prior to buckling, (b) postbuckling; and (c) initially imperfect model. Adapted from (Ziemian, 2010, pp. 13-18)

The model consists of two rigid bars hinged to one another and supported laterally by a nonlinear elastic spring (or similar). The spring possesses an arch-like stiffness, which can be approximated by a cubic polynomial. Cox (1940, p. 231) and Koiter (1970, p. 2) provide a solution of the form

$$P = P_C\left[1 - a\cdot\varepsilon + b\cdot\varepsilon^2\right], \tag{2.14}$$

where P_C is the critical buckling load, a and b are related to the spring coefficients, and $\varepsilon = x/L$. In his dissertation, Koiter (1970, pp. 71-117) used perturbation analysis to generalize the theory of stability for elastic systems under conservative loading. He arrived at Eq. (2.15), which demonstrates an asymptotically exact relation between λ (the postbuckling load ratio) and λ_C near the bifurcation point (Hutchinson & Koiter, 1970, p. 1355; Bushnell, 1981, p. 1193):

$$\lambda = \lambda_C\left[1 + a_s\cdot\xi + b_s\cdot\xi^2\right]. \tag{2.15}$$

For shells, ξ represents the post-buckling deflection δ as a multiplier to the thickness, i.e., $\xi = \delta/t$. Equation (2.15) can be visualized in Fig. 2.9. Koiter (1970, pp. 71-117) identified three postbuckling behaviors based on the parameters a_s and b_s: stable, unstable and asymmetric.

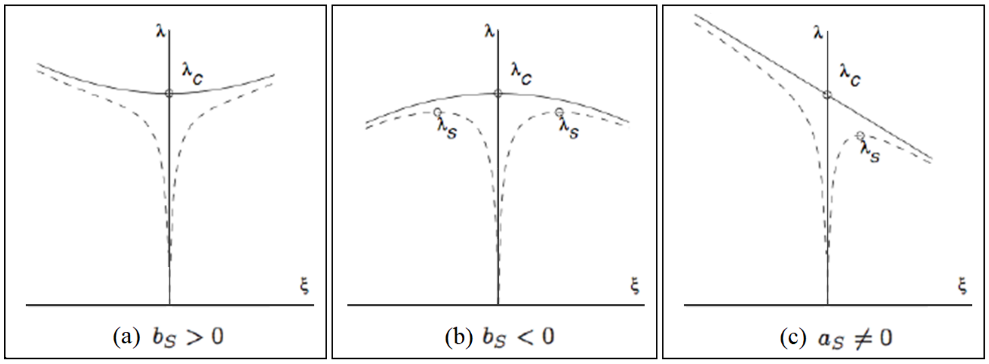


Figure 2.9. Postbuckling: (a) stable, (b) unstable, and (c) asymmetric (Bažant & Cedolin, 1991, p.470)

Equation (2.15) agrees with buckling theories for different structural elements. Figure 2.10 shows how typical structural elements fall within these categories.

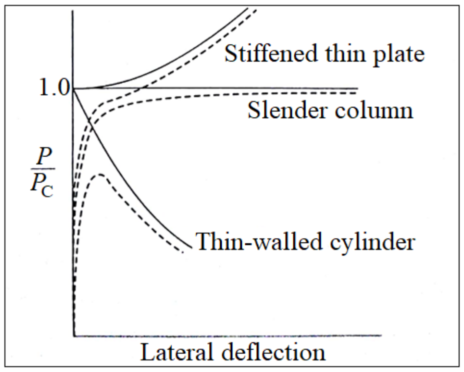


Figure 2.10. Elastic postbuckling curves for compressed elements (Ziemian, 2010, p. 8)

Koiter (1970, pp. 119-149) also noted that small initial geometric imperfections such as in Fig. 2.8(c) have a marked effect on systems with unstable postbuckling curves (dashed curves on Fig. 2.9 and 2.10). These imperfections cause those systems to fail at loads below the critical load (Ziemian, 2010, p. 19, Budiansky & Hutchinson, 1966, p. 1506). Such systems are referred to as imperfection-sensitive.

2.2.4. Imperfection Sensitivity

The classical example of imperfection sensitivity is the axially loaded thin-shell cylinder. Figure 2.11 shows the test results of 172 axially loaded thin-shell cylinders compared to the critical load predicted by Eq. (2.13). n_S has values as low as one sixth of n_C. Kármán and Tsien (1941, p. 303-312) attributed this discrepancies to the highly unstable postbuckling regimes seen in both cylindrical and spherical shells.

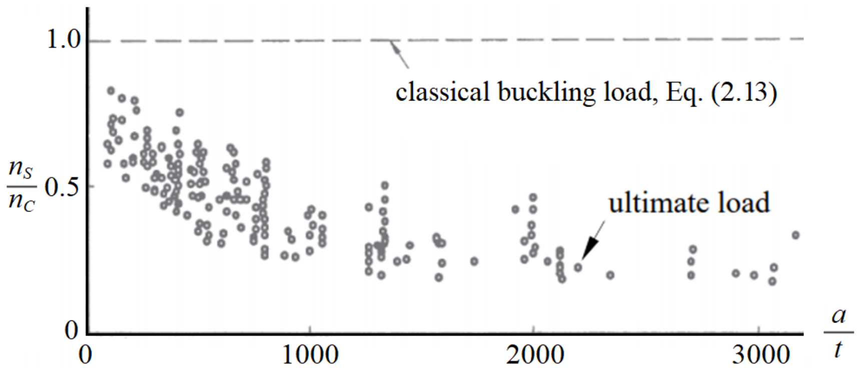


Figure 2.11. Test results of axially loaded cylinders (from Weingarten, Morgan & Seide, 1965)

Around the same time, Koiter found that asymptotically exact estimates of λ_S can be obtained by including the first-order effects of small initial geometric imperfections in the shape of the critical buckling mode (Hutchinson & Koiter, 1970, pp. 1355-1356; Bushnell, 1981, p. 1193). If the magnitude of the initial geometric imperfection is denoted $\bar{\delta}$, then for $a_s = 0$ and $b_s < 0$, λ_S can be estimated by

$$\lambda_S \cong \lambda_C\left[1-3(-b_S/4)^{1/3}(\rho\bar{\delta})^{2/3}\right], \tag{2.16}$$

where ρ is a constant that depends on the imperfection shape. On the other hand, for a postbuckling curve with $a_s \neq 0$ and $b_s = 0$, λ_S is estimated using

$$\lambda_S \cong \lambda_C\left[1-2(-\rho a_S\bar{\delta})^{1/2}\right]. \tag{2.17}$$

In both cases, small values of $\bar{\delta}$ have a sizeable effect on λ_S (Hutchinson & Koiter, 1970, p. 1356) which further substantiates the claim by Kármán and Tsien (1941, p. 303-312).

2.2.4.1. Externally pressurized thin-shell cylinder

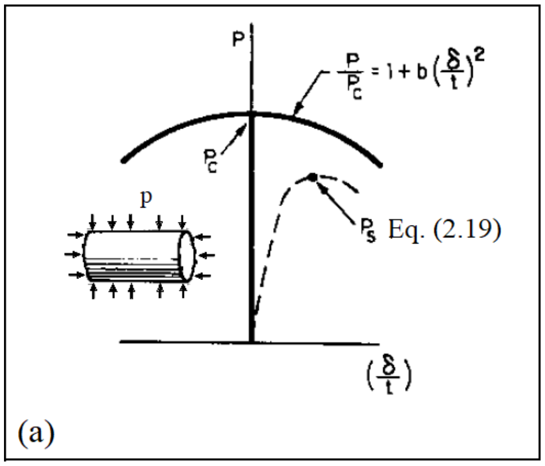
The externally pressurized thin-shell cylinder studied by Budiansky and Amazigo (1969, p. 223-235) is an illustrative example of Koiter's theory. The solid curve on Fig. 2.12 (a) represents the pressure-deflection relation of the perfect structure given by

$$p = p_C\left[1+b\cdot\left(\frac{\delta}{t}\right)^2\right]. \tag{2.18}$$

where p is the postbuckling pressure, p_C is the critical buckling pressure and δ is the normal to surface buckling displacement amplitude. In turn, the solid curves in Fig. 2.12 (b) represent the asymptotic relationship between p_S and $\bar{\delta}$ given by Eq. (2.19), which is the same form as (2.16):

$$\left(1-\frac{p_S}{p_C}\right)^{3/2}=\frac{3\sqrt{3}}{2}(-b)^{1/2}\left|\frac{\bar{\delta}}{t}\right|\left(\frac{P_S}{P_C}\right). \tag{2.19}$$

However, Eq. (2.16) and (2.17) cannot be applied to the axially loaded thin-shell cylinder nor the externally pressurized spherical shell studied by Kármán and Tsien in 1941 (pp. 303-312) and 1939 (pp. 43-50) due to the multiplicity of buckling modes associated with λ_C (Budiansky & Hutchinson, 1966, p. 1506; Hutchinson & Koiter, 1970, pp. 1358-1360; Bushnell, 1981, p. 1189). The reader may recall that Eq. (2.13) – applicable to both cylinders and spheres – makes no mention of a buckling pattern, thus hinting at the fact these shells are susceptible to several mode-based geometric imperfections (Bushnell, 1981, p. 1190).



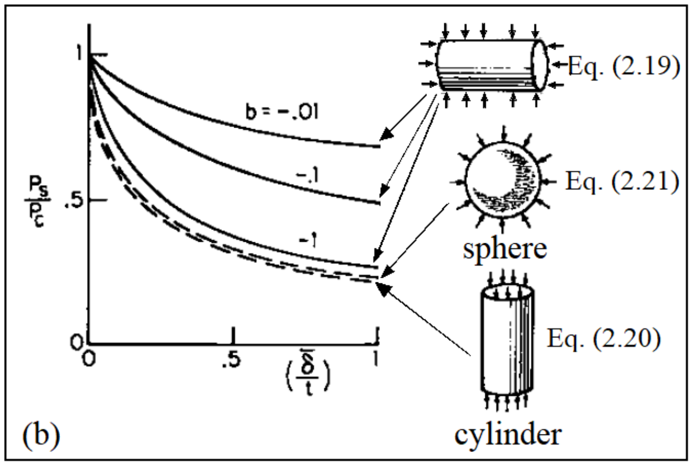


Figure 2.12. (a) Postbuckling and imperfection sensitivity of externally pressurized cylinder, and (b) Imperfection sensitivity of various shells (modified from Budiansky & Hutchinson, 1966, p. 1506)

2.2.4.2. Axially loaded thin-shell cylinder and externally pressurized spherical shell

Even with this limitation, it is possible to give a close estimate of P_S for the axially loaded thin-shell cylinder with the classical theory by using an imperfection in the shape of the axisymmetric buckling mode (Koiter, 1970, pp. 289-290, Hutchinson & Koiter, 1970, p. 1359):

$$\left(1-\frac{P_S}{P_C}\right)^{2}=\frac{3c}{2}\left|\frac{\bar{\delta}}{t}\right|\left(\frac{P_S}{P_C}\right) \tag{2.20}$$

or

$$\frac{P_S}{P_C}=1+\frac{3c}{4}\frac{\bar{\delta}}{t}-\sqrt{\frac{3c}{4}\frac{\bar{\delta}}{t}\left(2+\frac{3c}{4}\frac{\bar{\delta}}{t}\right)},$$

where $c=\sqrt{3(1-\nu^2)}$. Koiter (1970, pp. 289-290) also found that the cylinder's length and boundary conditions play a negligible role. Similarly, Hutchinson (1967a, p. 52) developed an equation for a shallow section of an externally pressurized spherical shell, taking in consideration the interaction between buckling modes. The highest reduction in pressure was observed for two operative buckling modes with one such mode having a zero wave-number associated with either the *x*- or *y*-coordinate:

$$\left(1-\frac{p_S}{p_C}\right)^2=\frac{27\sqrt{3}c}{32}\left|\frac{\bar{\delta}}{t}\right|\left(\frac{p_S}{p_C}\right) \tag{2.21}$$

or

$$\frac{p_S}{p_C}=1+\frac{27\sqrt{3}c}{64}\frac{\bar{\delta}}{t}-\sqrt{\frac{27c}{1024}\frac{\bar{\delta}}{t}\left(32\sqrt{3}+\frac{81c}{4}\frac{\bar{\delta}}{t}\right)}.$$

Equations (2.20) and (2.21) are plotted in Fig. 2.12. Similar imperfection sensitivity studies were done on axially compressed oval thin-shell cylinders (Hutchinson, 1968), externally pressurized thin-shell spheroids (Danielson, 1969) and externally pressurized thin-shell toroidal segments (Hutchinson, 1967b). It appears that imperfection sensitivity disappears for toroidal segments of sufficiently large negative Gaussian curvature (Budiansky & Hutchinson, 1966, p. 1507).

2.2.5. FEM Solution Techniques

Two FEM procedures are described, each with its own strengths and pitfalls: linear buckling analysis (LBA) and geometrically nonlinear analysis (GNA). LBA is an eigenvalue analysis based on Eq. (2.22) (ANSYS Inc., 2009, p. 1008; McGuire, Gallagher & Ziemian, 2000, p.235):

$$\left(\mathrm{K}+\lambda_{\mathrm{i}}\mathrm{K}_{\mathrm{g}}\right)\Psi_{\mathrm{i}}=0, \tag{2.22}$$

where K is the (linear elastic) stiffness matrix, K_g is the geometric stiffness matrix computed for a reference load, λ_i is an eigenvalue (buckling load factor) and Ψ_i is a corresponding eigenvector (buckling mode). LBA assumes negligible deflections prior to bifurcation of the loading path (McGuire, Gallagher & Ziemian, 2000, p. 218). The lowest eigenvector is referred to as the critical buckling load, that is, λ_C.

The assumption of negligible displacements seldom holds: the transition to an alternate load path is usually gradual due to deflections which may be enhanced or even triggered by the presence of initial geometric imperfections. GNA accounts for these deformations by updating the geometry and satisfying equilibrium on the deformed geometry (McGuire, Gallagher & Ziemian, 2000, p. 219).

GNA commonly tracks the equilibrium path via an incremental-iterative scheme: equilibrium is established to prescribed tolerances by means of iterations at each load increment (McGuire, Gallagher & Ziemian, 2000, p. 236-237). The reader is referred to (Borst & Crisfield, 2012) for a comprehensive treatment of such schemes. The script provided in App. H stipulates that analyses of this sort be executed using an arc-length controlled Newton-Raphson method.

While GNA can yield a good approximation of λ_L, it fails to capture the effect of initial geometric imperfections. GNA with explicitly modelled initial geometric imperfections is referred to as a GNIA. Such analyses are typically required for cases in which the initial geometric imperfections play a crucial role in triggering the nonlinear behavior and account for a significant reduction from λ_C to λ_S.

2.2.6. Initial Imperfections

Chen (2014, pp. 66-85) proposes four approaches to adding imperfections. The first approach is to update the geometry by rescaling the k^{th} buckling mode. This is achieved via

$$\delta^{\text{Imp}}(u,v)=\frac{\bar{\delta}}{\delta_{\max}^{k}}\delta^{k}(u,v), \tag{2.23}$$

where $\delta^{\text{Imp}}(u,v)$ is the imperfection, $\bar{\delta}$ is the prescribed magnitude, $\delta^{k}(u,v)$ is the deflection of the k^{th} buckling mode and $\delta_{\max}^{k}$ is the absolute maximum of δ^{k}. The second approach suggested by Chen (2014, p. 67) is to apply a uniform combination of n buckling modes based on

$$\delta^{\text{Imp}}(u,v)=\frac{\bar{\delta}}{\max\sum_{k=1}^{n}\delta^{k}(u_i,v_j)}\sum_{k=1}^{n}\delta^{k}(u,v). \tag{2.24}$$

The denominator equals the maximum deflection of the sum of the n modes for all possible (u_i, v_j). Chen further suggests using

$$\delta^{\text{Imp}}(u,v)=\frac{\bar{\delta}}{\delta_{\max}^{\text{rand}}}\delta^{\text{rand}}(u,v) \tag{2.25}$$

instead of Eq. (2.24), because the contribution of each buckling mode is randomized. The buckling modes are combined to yield δ^{rand} using

$$\delta^{\text{rand}}(u,v)=\sum_{k=1}^{n}A^{k}\delta^{k}(u,v), \tag{2.26}$$

where

$$A^{k}=\text{rand}(0,1); k\in[1,n]. \tag{2.27}$$

The remaining two approaches suggested by Chen are using random noise imperfections, a drawback of which is its mesh dependency; and imperfection patterns based on sinusoidal waves. Only Eq. (2.23) is implemented in the code in App. H. Nevertheless, using Eq. (2.24) or Eq. (2.25) could yield remarkable results as the first mode may not govern as shown by the axially-loaded thin-shell cylinder and the externally pressurized spherical shell (Budiansky & Hutchinson, 1966, p. 1506; Hutchinson & Koiter, 1970, p. 1358). Chen (2014, p.85) also observed that for structures with closely spaced buckling loads the imperfection sensitivity tends to be significant and is often controlled by a combination of buckling modes.

3. METHODOLOGY

3.1. General Procedure

The goal of the current treatise is to cater a recipe to conveniently generate a comprehensive database composed of the parameters and results shown in Fig. 3.1. The database should later be employed to fit (some of) the parameters and membrane forces into equation(s) that can accurately predict λ_S.

E	t	v	k_{xx}	k_{yy}	k_{xy}	$\bar{\delta}$	n_{xx}	n_{yy}	n_{xy}	λ_C	λ_S
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Figure 3.1. Table of parameters and results to be produced

This database is to be produced using ANSYS® Mechanical APDL Release 18.1 (ANSYS Inc., 2017), hereinafter referred to as MAPDL (Mechanical ANSYS Parametric Design Language), and Python. The procedure has two main components: a MAPDL macro that executes a GNIA and a Python script that runs the macro multiple times while varying the parameters.

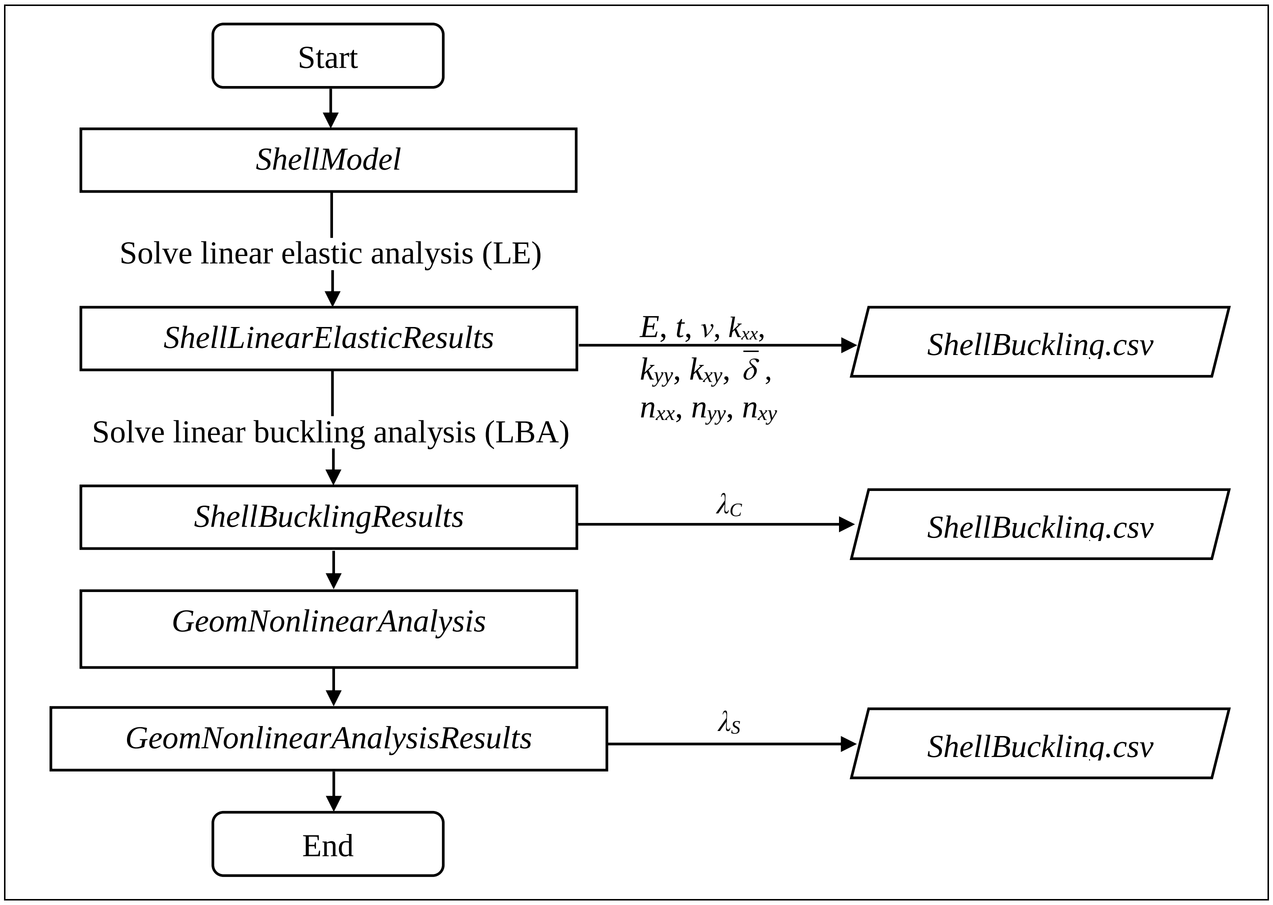

Start
ShellModel
Solve linear elastic analysis (LE)
ShellLinearElasticResults
E, t, v, k_{xx}, k_{yy}, k_{xy}, $\bar{\delta}$, n_{xx}, n_{yy}, n_{xy}
ShellBuckling.csv
Solve linear buckling analysis (LBA)
ShellBucklingResults
λ_C
ShellBuckling.csv
GeomNonlinearAnalysis
GeomNonlinearAnalysisResults
λ_S
ShellBuckling.csv
End

Figure 3.2. *ShellBuckling* flow diagram

The main MAPDL macro is named *ShellBuckling*. Its subroutines mirror MAPDL's workflow: model creation, performing a linear elastic analysis (LE) to obtain the membrane forces, a linear buckling analysis (LBA) to obtain the mode-based initial geometric imperfections and λ_C, and a geometrically nonlinear analysis (GNA) to obtain λ_S. The parameters and results of each analysis are appended, not overwritten, onto a single line in a comma-separated-values file: *ShellBuckling.csv*. Figure 3.2 shows the flow diagram for *ShellBuckling*. Text in boxes is delegated to subroutines, distilled in sections 3.2 through 3.5. The Python script (section 3.6) provides the input parameters, runs MAPDL and waits for the output.

ShellBuckling and all its subroutines are written in a single library file, named *ShellBucklingLibrary.mac*. *ShellBuckling* and the subroutines are included in the appendices, starting with *ShellBuckling* in App. A. To reassemble the library, copy the content of each appendix onto a *.mac* file titled *ShellBucklingLibrary*. The Python script is in App. K. Alternatively, the MATLAB (The MathWorks, Inc., 2017) script in App. L can be used in lieu of the Python one.

3.2. Model Creation

The idea is to model a portion of a shell subject to combinations of membrane forces and curvatures that will lead to buckling. This shell portion needs to be extended to minimize the effect of the boundary conditions on the buckled shape and load. The proposed models are a toroidal shell segment (Fig. 3.3) and a complete torus. The complete torus model is discussed amply in section 3.7, with the reasons for not choosing is as the main geometry in section 4.1.

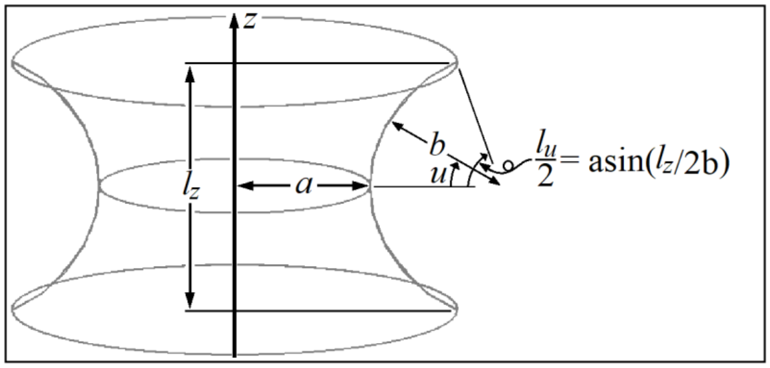


Figure 3.3. Model geometry

The toroidal shell segment is parametrized by equations that model the inner walls of a torus; hence, it is easy to change the principal curvatures. To diminish the effects of the boundary conditions, the external pressure is only applied to the model area between 0.25 to 0.75 times its height. The top ring is restrained against displacement in the global $\bar{x}$ and $\bar{y}$ directions while the bottom ring is restrained against displacement in all directions (see subsection 3.2.3).

The model uses reduced-solid 8-noded quadrilateral shell elements (SHELL281) of thickness *t*. The material is linear-elastic with modulus *E* and Poisson's ratio *ν*. The element type is defined with **ET**, the thickness with **R** and the material properties with **MP**. All commands (in bold) are from (ANSYS Inc., 2009). The code snippet in Fig. 3.4 defines the element type and material properties. Note that the Poisson's ratio is represented by w to avoid confusion between the *v*-parameter and the Greek letter *ν*.

```
! Create element type: shell
!--------------------------------------------------------------------
ET,1,SHELL281                         ! element type: 8 node quadrilateral
R,1,t,t,t,t, , ,                      ! element thickness

! Create material properties
!--------------------------------------------------------------------
MP,EX,1,E                             ! Elastic modulus (linear elastic material)
MP,PRXY,1,w                           ! Poisson's ratio (linear elastic material)
```

Figure 3.4. Element type and material properties code snippet (from *ShellModel*)

The toroidal shell segment is parametrized by

$$\mathrm{r}(u,v)=\begin{Bmatrix}\bar{x}(u,v)\\ \bar{y}(u,v)\\ \bar{z}(u)\end{Bmatrix}=\begin{Bmatrix}(a+(1-\cos(u))b)\cos(v)\\ (a+(1-\cos(u))b)\sin(v)\\ b\sin(u)\end{Bmatrix}. \tag{3.1}$$

where v goes from 0 to 2π and u goes from $-l_u/2$ to $l_u/2$. l_u, in turn, is calculated using

$$l_u = 2\sin^{-1}(l_z/2/b). \tag{3.2}$$

The length l_z is the total height of the model (i.e., from - $l_z/2$ to $l_z/2$). Subsections 3.2.1 and 3.2.2 illustrate the creation of nodes (**N**) and elements (**E**), respectively. Subsection 3.2.3 summarizes the enforcement of fixities (**D**) and imposition of surface loads (**SFE**). When creating the model, a series of nodes following the u-parameter from $-l_u$ to l_u will be referred to as a nodal column and a series of nodes going along the v-parameter from 0 to n_v will be called a nodal row. The same nomenclature will be applied to elements. *ShellModel*, whose snippets are shown throughout section 3.2, is in App. B.

3.2.1. Nodes

Nodal counters, used exclusively for node creation, are denoted n_i and n_j. Nodes are created to accommodate n_u rows by n_v columns of 8-noded quadrilateral shell elements. $2n_u + 1$ nodal rows and $2n_v$ nodal columns are required to accommodate all elements. The '$2n_v$ +1' nodal column (at $v = 2\pi$) is omitted because it coincides with the first one (at $v = 0$).

A nodal column is created in a loop with n_i going from $-n_u$ to $+n_u$. This loop is nested inside another loop with n_j going from 0 to $2n_v - 1$ to create nodal columns along the v-parameter. The loop counters are chosen to fit the required number of nodal rows and columns and to ease calculation of the u and v parameters. The nodes are at equally spaced intervals of $l_u/2n_u$ along u and π/n_v along v. For each node, u and v are obtained from n_i and n_j using

$$u = n_i(l_u/2n_u) \tag{3.3}$$

and

$$v = n_j(l_v/2n_n). \tag{3.4}$$

Equation (3.1) then yields $\bar{x}$, $\bar{y}$ and $\bar{z}$, and the node is created via **N**. The variables t_x and t_y are introduced that, when added together, act like a Boolean indicating when a node should or should not be created. If and only if $t_x + t_y$ is less than 1, a node is created. Thus, nodes are omitted if n_i and n_j are simultaneously even. Figure 3.5 shows the code snippet in charge of node creation. Figure 3.6 shows the node creation pattern (for $a > 0$, $b > 0$, $n_u = 4$ and $n_v = 12$). The red lines follow the direction of node creation, stating at N1, in both the u-v plane and global coordinates.

```
! Create Shell nodes
!----------------------------------------------------------------------
! ty and tx are dummy variables that, added together, act like a Boolean indicating when
! a node should be created. If ty + tx is less than 1, then a node is created.
ty=1
*DO,nj,0,2*nv-1       ! Nodal columns going from nj = 0 to 2*nv-1, i.e., (2*nv) columns
   ty=-ty
   tx=1
   v = nj*lv/nv/2                         ! Parameter v (U-V plane)
   *DO,ni,-nu,nu      ! Nodal rows going from ni = -nu to nu, i.e., (2*nu) rows
      tx=-tx
      *IF,tx+ty,LT,1,THEN                 ! if tx+ty = 2 omit node creation
         u = ni*lu/nu/2                   ! Parameter u (U-V plane)
         x = (aa+(1-COS(u))*bb)*COS(v)    ! x-coordinate x = x(u,v)
         y = (aa+(1-COS(u))*bb)*SIN(v)    ! y-coordinate y = y(u,v)
         z = bb*SIN(u)                    ! z-coordinate z = z(u)
         N,,x,y,z,,,                      ! Create node
      *ENDIF
   *ENDDO
*ENDDO
```

Figure 3.5. Node creation code snippet (from *ShellModel*)

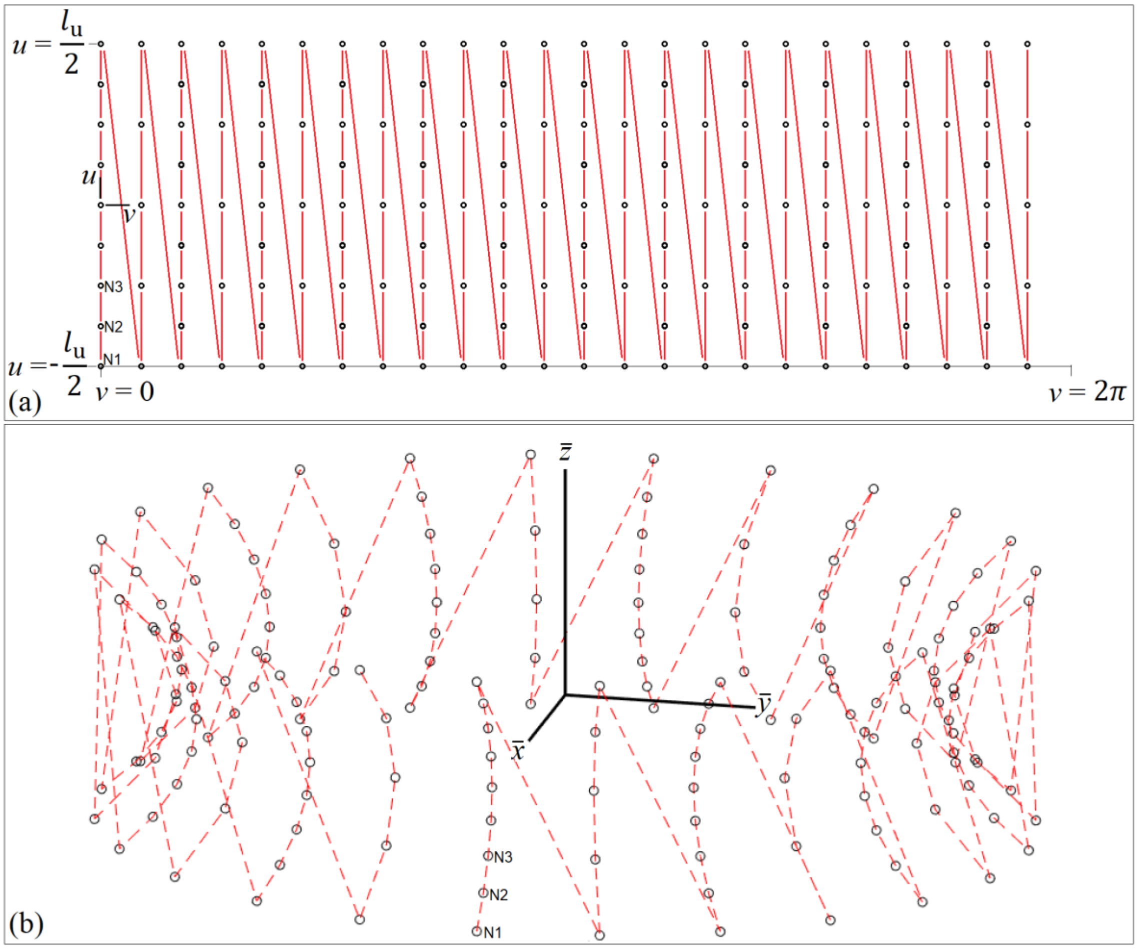


Figure 3.6. Node creation: (a) in the *u*-*v* plane, and (b) in global coordinates

3.2.2. Elements

Element rows and columns are denoted *i* and *j* respectively. An element column is created using a loop with *i* going from 1 to n_u. This loop is nested within another loop with *j* going from 1 to n_v that creates n_v element columns along the *v*-parameter. An element is created using **E** whose arguments are the numbers identifying the surrounding 8 nodes. Equations (3.5), (3.6) and (3.7) identify the numbers of the bottom three nodes of an element, from left to right, based on *i* and *j*:

$$k_1 = 1 + 2(i-1) + (j-1)(3n_u + 2)\,, \qquad (3.5)$$

$$k_2 = i + (3j-1)n_u + 2j - 1 \qquad (3.6)$$

and

$$k_3 = 1 + 2(i-1) + j(3n_u + 2)\,. \qquad (3.7)$$

These equations hold for all elements except those in the last column ($j = n_v$) because, as noted in subsection 3.2.1, the last nodal column is omitted. Instead, those nodes are identified using $k_1(j = 1)$, i.e., the first nodal column. Figure 3.7 shows the two cases that arise because of omitting this nodal column: (a) $j < n_v$ and (b) $j = n_v$. Additionally, the arrows represent the direction of the arguments in **E**, starting with red, i.e., **E**, k3, k3+2, k1+2, k1, k3+1, k2+1, k1+1, k2.

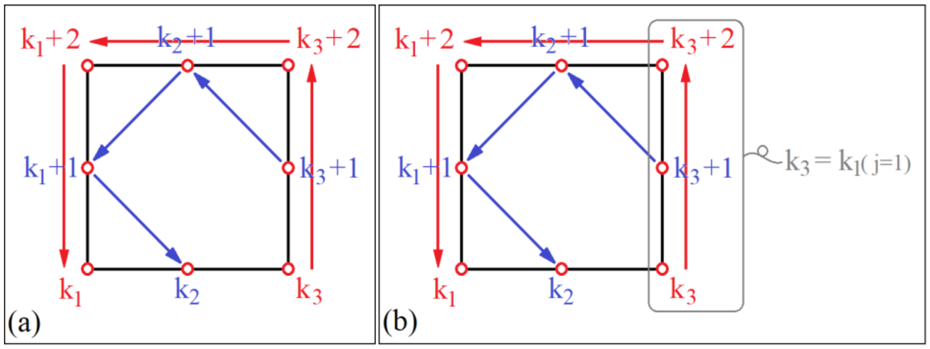


Figure 3.7. Two cases in element creation: (a) $j < n_v$, and (b) $j = n_v$

Figure 3.8 shows the element creation code snippet. Figure 3.9 shows the element creation pattern starting with E1 for a shell model with $a > 0$, $b > 0$, $n_u = 4$ and $n_v = 12$.

```
! Create Shell elements
!---------------------------------------------------------------------
SHPP,OFF                                   ! no warning aspect ratio
*DO,j,1,nv                                 ! j-th element column (along v-axis)
    *DO,i,1,nu                             ! i-th element row (along u-axis)
      k1 = 1 + 2*(i-1) + (j-1)*(3*nu+2)
      k2 = i + 2*j + (3*j - 1)*nu - 1
      *IF,j,LT,nv,THEN                     ! j < nv
         k3 = 1 + 2*(i-1) + j*(3*nu+2)
      *ELSE                                ! j = nv
         k3 = 1 + 2*(i-1)
      *ENDIF
      E,k3,k3+2,k1+2,k1,k3+1,k2+1,k1+1,k2
   *ENDDO
*ENDDO
```

Figure 3.8. Element creation code snippet (from *ShellModel*)

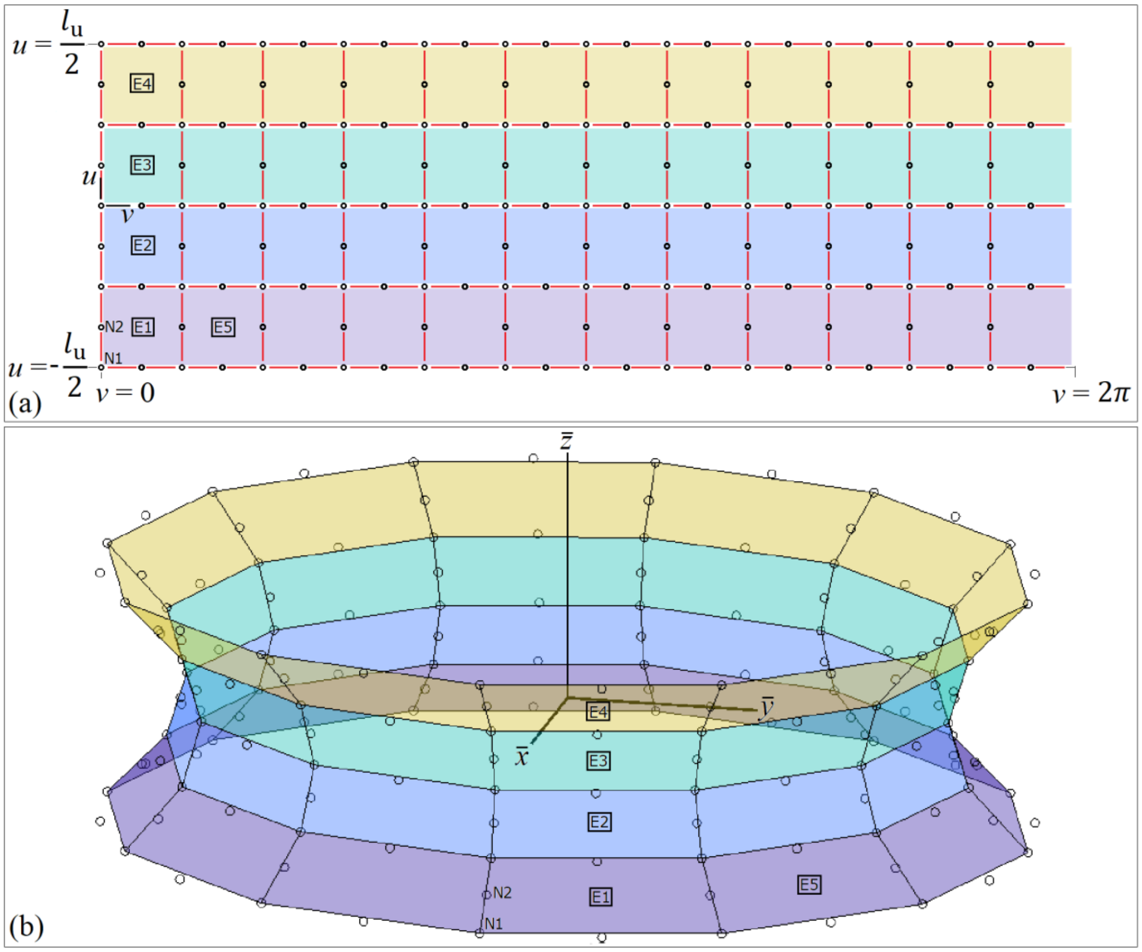


Figure 3.9. Element creation: (a) in the u-v plane, and (b) in global coordinates

3.2.3. Boundary Conditions

Fixities (Dirichlet boundary conditions) are enforced with **D**. The pressure load (Neumann boundary condition) is applied on the elements using **SFE**. The top ring nodes are restrained against displacement in the global $\bar{x}$ and $\bar{y}$ directions while the bottom ring nodes are fully pinned. Equation (3.5) is used to select the bottom nodes by setting $i = 1$, i.e., $k_1(i = 1, j)$. In turn, the top nodes are selected using $k_1(i = n_u, j) + 2$.

Uniform pressure is applied externally between $-\frac{1}{4}l_u$ to $\frac{1}{4}l_u$, which maps to roughly half the height. Elements are selected along the jth column with $i_{\text{bot}} \le i \le i_{\text{top}}$, where

$$i_{\text{top}} = j \cdot n_u - NINT(\alpha \cdot n_u) \tag{3.8}$$

and

$$i_{\text{bot}} = 1 + (j-1)n_u + NINT(\alpha \cdot n_u)\,. \tag{3.9}$$

The NINT operation returns the closest integer. α (=¼) is half the proportion of the unloaded area. Figure 3.10 shows the code snippet which enforces the boundary conditions on the model.

```
! Create Dirichlet boundary conditions
!--------------------------------------------------------------------
! Top and bottom rings
*DO,j,1,nv                                  ! j-th element column (along v-axis)
   n_bot = 1 + (j-1)*(3*nu+2)               ! n_bot = k1(i = 1,j)
   n_top = 3 + 2*(nu-1) + (j-1)*(3*nu+2)   ! n_top = k1(i = nu,j) + 2
   D,n_top,UX,0,,,,UY,
   D,n_bot,UX,0,,,,UY,UZ
*ENDDO

! Create Neumann boundary conditions
!--------------------------------------------------------------------
ESEL,S,ELEM,,nu/2
*DO,j,1,nv                                  ! j-th element column (along v-axis)
   i_bot = 1+(j-1)*nu+NINT(alpha*nu)       ! i_bot-th element row
   i_top = j*nu-NINT(alpha*nu)             ! i_top-th element row
   ESEL,A,ELEM,,i_bot,i_top,1              ! Append elements from i_bot to i_top in steps of 1
*ENDDO
SFE, ALL, 1, PRES, 0, p,,,                  ! Add uniform pressure on all selected elements
ALLSEL                                      ! Reselect all elements
```

Figure 3.10. Fixities and load code snippet (from *ShellModel*)

Figure 3.11 shows a test model using n_u = 40, n_v = 120, a = 2000 mm and b = 5000 mm.

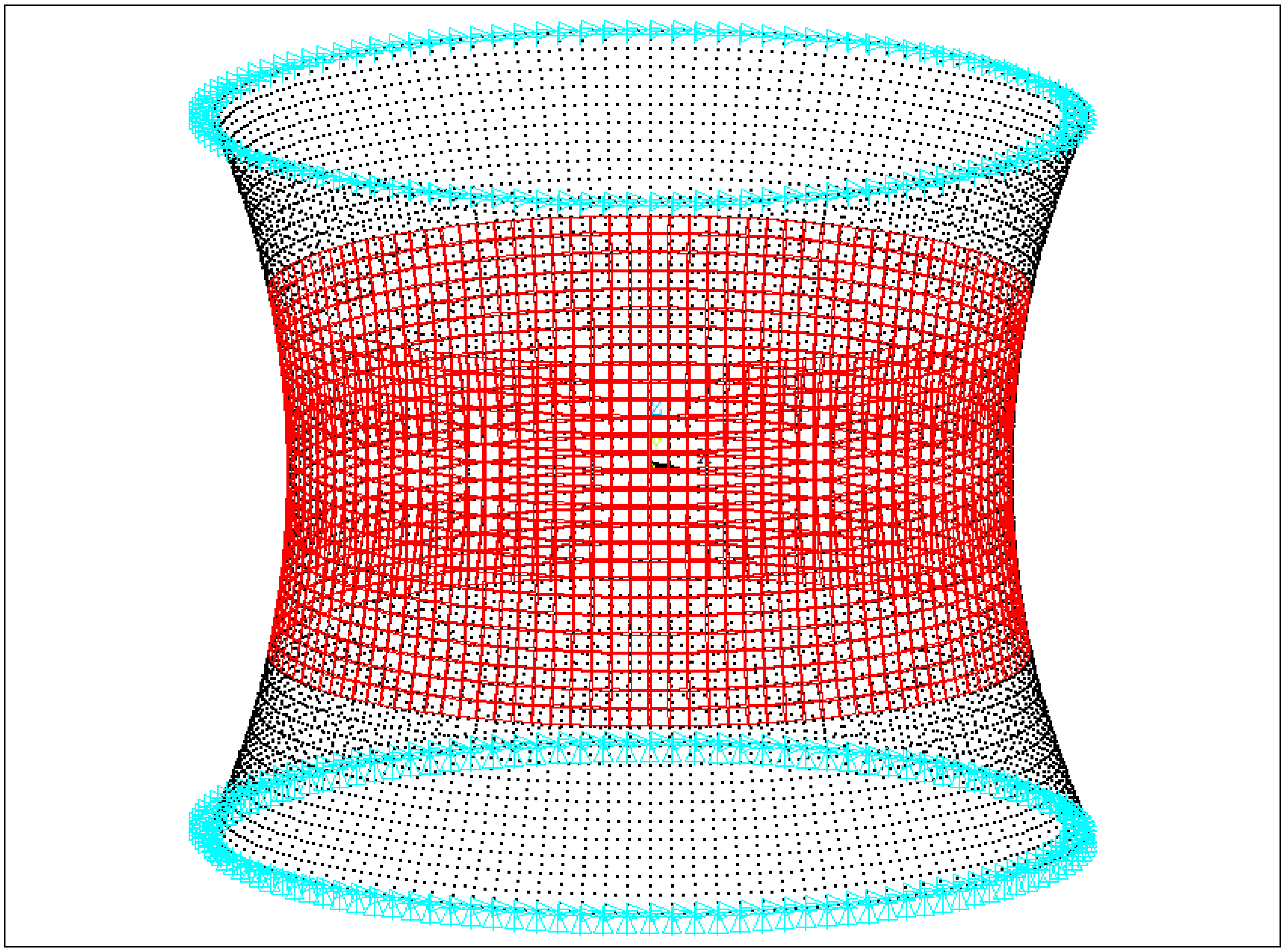


Figure 3.11. Fixities and load on a test shell

3.3. Linear Elastic Analysis (LE)

MAPDL's solution procedure is preceded by **/SOLU**. The LE analysis is specified with **ANTYPE, STATIC** (ANSYS, 2009, p. 977). Additionally, **PSTRES, ON** (prestress effects) is required after **ANTYPE** to save the stress state for the LBA (ANSYS Inc., 2009, p.1007). Figure 3.12 shows the code snippet – within *ShellBuckling* – in charge of the LE analysis.

```
! Linear Elastic Analysis (Find stresses for buckling analysis)
!-------------------------------------------------------------------
/SOLU
ANTYPE, STATIC                        ! Linear elastic analysis
PSTRES, ON                            ! Prestress effects to be included in buckling analysis
SOLVE
FINISH
*USE,ShellLinearElasticResults        ! Membrane forces
```

Figure 3.12. Linear elastic analysis (LE) code snippet (from *ShellBuckling*)

After the LE analysis, *ShellLinearElasticResults* (App. C) is run. Element-specific result variables such as membrane forces, moments and shears are defined in MAPDL's **/POST1** post-processor using **ETABLE**. Then, ***GET** stores said variables into parameters that are operated upon and later written to *ShellBuckling.csv*. The membrane forces in the loaded area are averaged to $\bar{n}_{xx}$, $\bar{n}_{yy}$ and $\bar{n}_{xy}$. (written as nx_avg, ny_avg, and nxy_avg in the code). Membrane forces at mid-height are stored as n_{xx}, n_{yy} and n_{xy} (nx, ny, and nxy). Figure 3.13 shows the code that extracts the element membrane forces and writes them to *ShellBuckling.csv*.

```
/POST1
ESEL,S,ELEM,,1,nu,1                   ! Select elements from 1 to nu in increments of 1
ETABLE,nxx,SMISC,1                    ! Extract shell nxx membrane force
ETABLE,nyy,SMISC,2                    ! Extract shell nyy membrane force
ETABLE,nxy,SMISC,3                    ! Extract shell nxy membrane force
ETABLE,mxx,SMISC,4                    ! Extract shell mxx moment
ETABLE,myy,SMISC,5                    ! Extract shell myy moment
ETABLE,mxy,SMISC,6                    ! Extract shell mxy moment
ETABLE,qx ,SMISC,7                    ! Extract shell vx shear force
ETABLE,qy ,SMISC,8                    ! Extract shell vy shear force
nx_avg = 0                            ! Re-set nx_avg to 0
ny_avg = 0                            ! Re-set ny_avg to 0
nxy_avg = 0                           ! Re-set nxy_avg to 0
i1 = NINT(alpha*nu)                   ! element number 1
i2 = NINT((1-alpha)*nu)               ! element number 2
*DO,nele,i1,i2
   *GET,nx,ETAB,1,ELEM,nele        $  nx_avg = nx_avg + nx
   *GET,ny,ETAB,2,ELEM,nele        $  ny_avg = ny_avg + ny
   *GET,nxy,ETAB,3,ELEM,nele       $  nxy_avg = nxy_avg + nxy
*ENDDO
nx_avg = nx_avg/(i2-i1+1)             ! nx_avg = Sum(nx)/(i2-i1+1)
ny_avg = ny_avg/(i2-i1+1)             ! ny_avg = Sum(ny)/(i2-i1+1)
nxy_avg = nxy_avg/(i2-i1+1)           ! nxy_avg = Sum(nxy)/(i2-i1+1)
*GET,nx,ETAB,1,ELEM,NINT(nu/2)
*GET,ny,ETAB,2,ELEM,NINT(nu/2)
*GET,nxy,ETAB,3,ELEM,NINT(nu/2)
*CFOPEN,ShellBuckling,csv,,APPEND
*VWRITE,E,t,w,aa,bb,lv,nv,lz,nu
(F10.0,',',F10.3,',',F10.3,',',F10.0,',',F10.0,',',F10.3,',',F10.0,',',F10.0,',',F10.0,',',$)
*VWRITE,delta,nx,ny,nxy,nx_avg,ny_avg,nxy_avg
(F10.3,',',F12.4,',',F12.4,',',F12.4,',',F12.4,',',F12.4,',',F12.4,',',$)
*CFCLOS ! The $ sign suppreses \n (new line command)
```

Figure 3.13. ETABLE results code snippet (from *ShellLinearElasticResults*)

Subsequently, the internal forces of all elements along the first element column ($j = 1$) are output to *Shell.txt*, which is overwritten at every analysis (Fig. 3.14).

```
/OUTPUT,Shell,txt
PRETAB,nxx,nyy,nxy
PRETAB,qx,qy
PRETAB,mxx,myy,mxy
/OUT
```

Figure 3.14. ETABLE results code snippet 2 (from *ShellLinearElasticResults*)

Shell.txt may be processed with Matplotlib (Hunter, 2007) to check the correctness of the LE solution by plotting graphs like Fig. 3.15 ($p = -0.1$MPa, $E = 10000$MPa, $t = 5$mm, $\nu = 0.3$, $a = 2000$mm, $b = 5000$mm, $l_z = 4000$mm, $n_u = 40$ and $n_v = 120$).

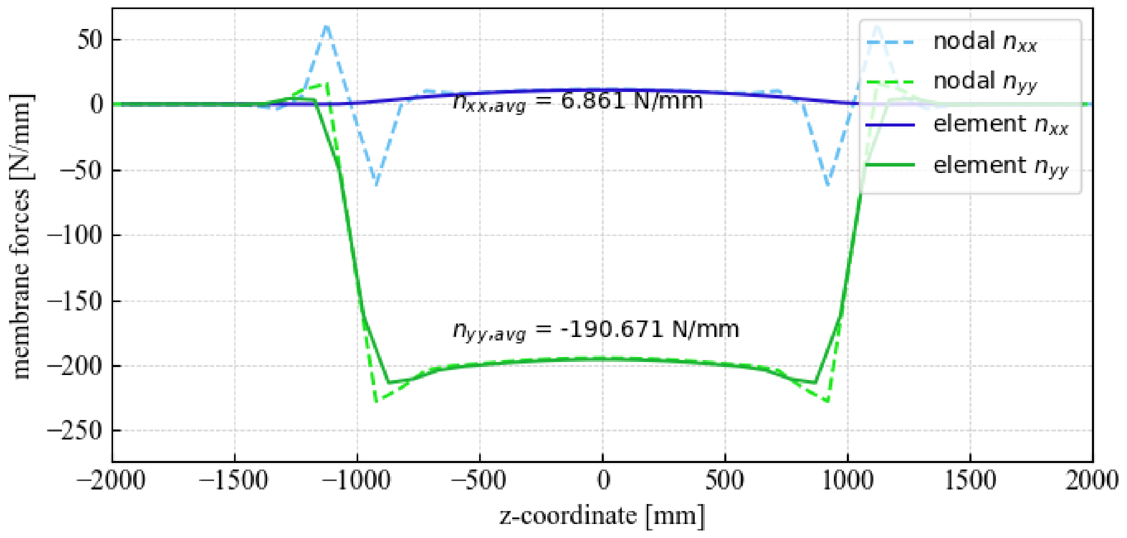


Figure 3.15. Element membrane forces along the z-coordinate of a test shell

Additionally, a subroutine named *NodalMembraneForces* (App. D) may be used to extract nodal stresses in global coordinates, transform them to local coordinates using *TransformationMatrix* (App. J), and into membrane forces with Eq. (3.10) and (3.11):

$$n_{\alpha\beta} = \sigma_{\alpha\beta,mid} \cdot t \tag{3.10}$$

and

$$q_\alpha = \frac{2}{3}\sigma_{\alpha z,mid} \cdot t\,. \tag{3.11}$$

α and β represent x or y. *NodalMembraneForces* is not essential and may be left out, provided the command that calls it is commented out. Nodal and element results are nonetheless compared in Fig. 3.15, 4.2, 4.3 and 4.4. The reader may note there is a discrepancy between the so-called nodal membrane forces and element forces. This discussion is deferred to chapter 4.

3.4. Linear Buckling Analysis (LBA)

The LBA is solved using the Block Lanczos method. ANSYS Inc. (2009, p.1008) recommends requesting a few additional modes than needed to enhance the accuracy of the final solution. Figure 3.16. shows the code snippet – within *ShellBuckling* – in charge of performing the LBA.

```
! Linear Buckling Analysis (Find buckling modes and buckling loads)
!----------------------------------------------------------------------
/SOLU
ANTYPE, BUCKLE                       ! Linear buckling analysis
BUCOPT, LANB, 5,0,,CENTER            ! Block Lanczos method, 5 buckling modes
SOLVE
FINISH
*USE,ShellBucklingResults            ! Buckling mode & lambdaC
```

Figure 3.16. Linear buckling analysis (LBA) code snippet (from *ShellBuckling*)

After the LBA, *ShellBucklingResults* (App. E) is run. It plots the first two buckling modes by executing *PlotBuckingModes* (App. F). Fig. 3.17 is based on same parameters as Fig. 3.15.

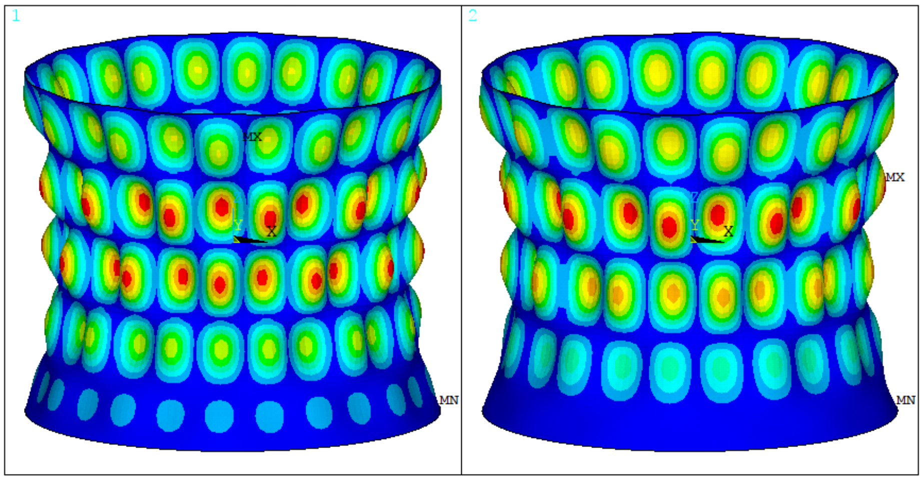


Figure 3.17. First and second buckling modes of a test shell

The buckling load, λ_C, is extracted using the code in Fig. 3.18.

```
/POST1
buckling_mode = 1
*USE,PlotBucklingModes               ! Plots 2 buckling modes (may be left commented out)
SUBSET,1,buckling_mode,FACT,,,,      ! Load buckling mode
*GET,lambdaC,ACTIVE,0,SET,FREQ       ! Get the buckling load factor lambdaC
*CFOPEN,ShellBuckling,csv,,APPEND
*VWRITE,lambdaC
(F11.5,',',$)
*CFCLOS
```

Figure 3.18. Get buckling load code snippet (from *ShellBucklingResults*)

Initial imperfections for the GNIA are based on Eq. (2.22). The code uses a similar nomenclature: $\overline{\delta}$ (delta) is the prescribed imperfection amplitude and $\delta_{\max}^{k}$ (uz_max) is the maximum deflection. The latter is obtained by looping through all the nodes and storing the largest $|u_z|$ (Fig. 3.19). The deflections are transformed to local coordinates using *TransformationMatrix* (App. J).

```
!Loop to read deflection of node next to ith element
uz_max = 0
defl_max = 0
*DO,j,1,nv                                      ! j-th element column (along v)
   *DO,i,1,nu+1                                 ! i-th element row (along u)
      nnode = 1 + 2*(i-1) + (j-1)*(3*nu+2)      ! nnode = k1(i,j)
      *GET,u_x,NODE,nnode,U,X                   ! Extract u_x from nnode
      *GET,u_y,NODE,nnode,U,Y                   ! Extract u_y from nnode
      *GET,u_z,NODE,nnode,U,Z                   ! Extract u_z from nnode
      defl = SQRT(u_x*u_x+u_y*u_y+u_z*u_z)      ! Absolute deflection
      u = (i-1)*(lu/nu)-lu/2                    ! u-parameter
      v = (j-1)*(lv/nv)                         ! v-parameter
      *USE,TransformationMatrix                 ! Assemble Gamma matrix
      *VEC,Defl_global,D,ALLOC,3,,,             ! Allocate space for Defl_global
      *SET,Defl_global(1),u_x,u_y,u_z           ! Let Defl_global = [u_x;u_y;u_z]
      *MULT, Gamma, , Defl_global, , Defl_local ! Defl_local = Gamma*Defl_global
      u_z = Defl_local(3)                       ! local z-displacement
      *IF,ABS(u_z),GT,uz_max,THEN
         uz_max = ABS(u_z)                      ! uz_max = max(u_z)
         defl_max = ABS(defl)                   ! defl_max = max(defl)
         nnode_max = 1 + 2*(i-1) + (j-1)*(3*nu+2)  ! node with highest deflection
      *ENDIF
   *ENDDO
*ENDDO
```

Figure 3.19. Get maximum deflection code snippet (from *ShellBucklingResults*)

The optional *NodalDisplacements* subroutine (App. G) may be run to output the buckling mode nodal displacements to *ShellLBA.csv*, which can be processed in MATLAB to yield, for instance, Fig. 3.20 based on the test shell from Fig. 3.15. u_z is an order of magnitude greater than u_x and u_y, as would be expected.

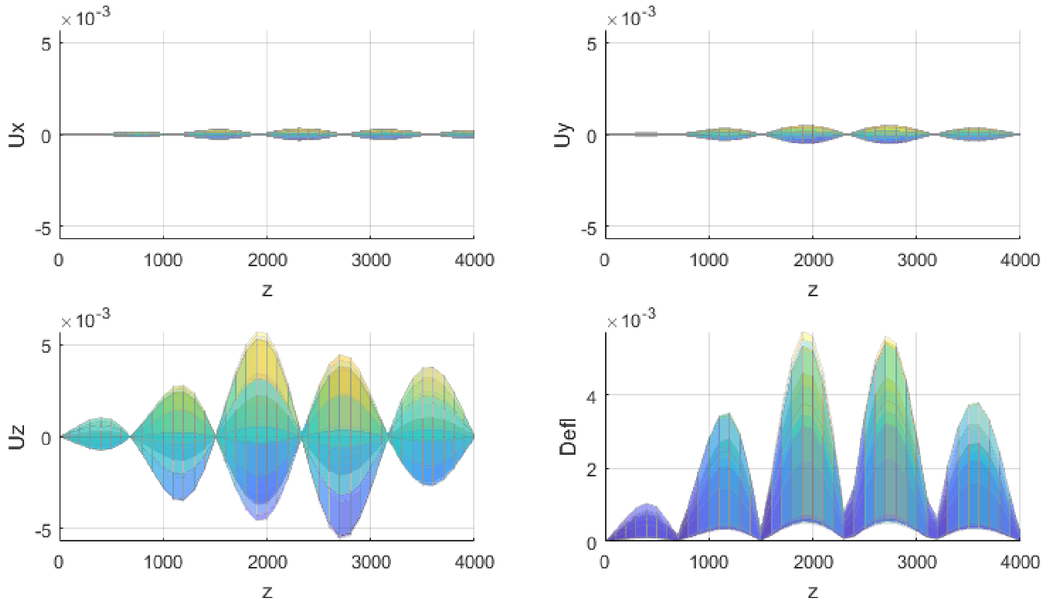


Figure 3.20. Buckling mode local coordinate displacements of a test shell

3.5. Geometrically Nonlinear Analysis with Initial Geometric Imperfections (GNIA)

GeomNonlinearAnalysis (App. H) updates the geometry using a buckling mode and $\delta_{\max}^{k}$ (uz_max) from section 3.4. The geometry is updated with **UPGEOM**, whose arguments are the factor (delta/uz_max), load step, buckling mode, file and file extension (Fig. 3.21). The solution is then set up and executed (Fig. 3.22).

```
! Update the geometry
!----------------------------------------------------------------------
/PREP7
FACTOR = delta/uz_max                         ! Factor for UPGEOM
UPGEOM,FACTOR,1,buckling_mode,'file','rst',  ! Add imperfections
/RESET   $  /ERASE   $     /REPLOT           ! Replot
FINISH
```

Figure 3.21. Update the geometry code snippet (from *GeomNonlinearAnalysis*)

```
/SOLU
! Set analysis type: GNA
!----------------------------------------------------------------------
NCNV,0                              ! Do not terminate program if not-converged
NERR,,,-1                           ! Do not terminate analysis if not-converged
ANTYPE, STATIC                      ! Static analysis
NLGEOM, ON                          ! Nonlinear geometry
TIME, 1                             ! Time at the end of load step

! Set nonlinear controls / solution technique
!----------------------------------------------------------------------
ARCLEN,ON                           ! Arclength ON
nsubstep = 200                      ! # of substeps. Keep as par b/c it is used later in *VEC
NSUBST,nsubstep,,                   ! Number of substeps
NEQIT,50,                           ! Max. number of iterations
CNVTOL,STAT                         ! Convergence tolerance (default)

! Set output controls
!----------------------------------------------------------------------
RESCONTROL,DEFINE,ALL,1,            ! Write new files at every substep
OUTRES,NSOL,ALL,,,,                 ! Write nodal results at every substep
OUTRES,ESOL,ALL,,,,                 ! Write element results at every substep

! Solve
!----------------------------------------------------------------------
SOLVE
FINISH
```

Figure 3.22. Geometrically nonlinear analysis code snippet (from *GeomNonlinearAnalysis*)

MAPDL uses **TIME** as the counter for both dynamic and nonlinear static analysis. In a non-proportional analysis, time acts as a counter for indexing each load step. For single load step analysis, time equals λ. Assuming λ_S does not exceed 1, *GeomNonlinearAnalysisResults* (App. I) retrieves λ_S with the time-dependent post-processor: **/POST26**. A loop runs through the time variable and stores its maximum value.

```
lambdaS = 0
*DO,k,1,nn
   *IF,time(k),GT,lambdaS,AND,time(k),LT,1,THEN
      lambdaS = time(k)
   *ENDIF
*ENDDO
```

Figure 3.23. Get GNIA load factor code snippet (from *GeomNonlinearAnalysisResults*)

Since time defaults to 1 at the end of each analysis, the loop ensures this value does not get saved. Prior to extracting λ_S, the time variable is stored in an array parameter using **VGET**. Likewise, the u_x, u_y and u_z deflections of the node with highest initial imperfection are defined as variables and stored into homonymous vector parameters. **VOPER** operates element-wise on the u_x, u_y and u_z vector parameters to yield $\delta = \sqrt{u_x^2 + u_y^2 + u_z^2}$. δ (defl) and λ (time) of every analysis are appended to *ShellGNIA.csv*. Varying the number of elements on the test shell from Fig. 3.15, force-displacement curves like in Fig. 3.24 can be obtained. Such plots verify the efficacy of Eq. (3.12) and (3.13) from section 3.6 in dictating enough elements. Another salient feature is that λ_S exceeds λ_C as expected for a negatively curved toroidal segment.

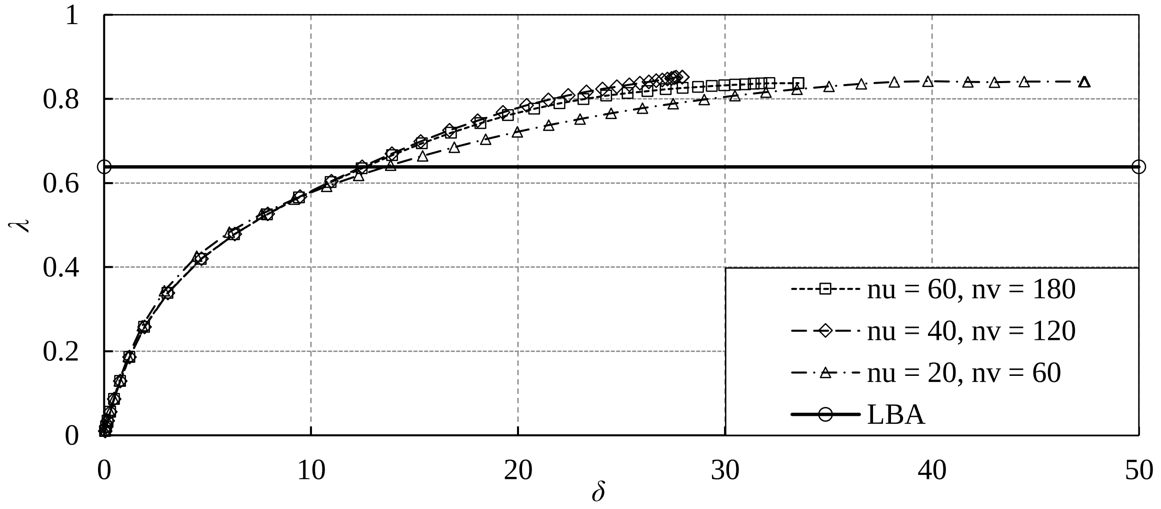


Figure 3.24. Force-displacement plot of a test shell

3.6. Python Script

The Python script consists of a function (*RunAPDL* in App. K) embedded in loops that iterate over set of input parameters. For *RunAPDL* to work properly, the main output files (*ShellBuckling.csv* and *ShellGNIA.csv*) must be closed and a valid ANSYS license must be reachable. Also, *RunAPDL* has a built-in piece of code that ensures no lock file prevents MAPDL from running.

RunAPDL executes three tasks: 1) write an input file with all the necessary parameters, 2) invoke MAPDL and 3) extract the critical buckling to update the input pressure for the following analysis. When writing the input file, a sufficient number of elements must be prescribed. The choice rests on the predicted number of buckling waves. The length of half a wave is approximately $\beta \cdot \sqrt{|a| \cdot t}$. Based on the number of waves in Fig. 3.17, β is estimated to be between 5.2 and 8. Thus, for a shell segment of radius a and $l_z = 2a$, setting

$$n_u = 2 \cdot \mathrm{NINT}\left(|a| \big/ \sqrt{|a| \cdot t}\right) \tag{3.12}$$

and

$$n_v = 3n_u \tag{3.13}$$

will ensure the presence of at least 5 to 8 elements per semi-wave in both x and y directions. For some given input parameters, the input file generated by *RunAPDL* may end up looking as in Fig. 3.25.

```
/NERR,200,10000,,OFF,0
pi = acos(-1)
E = 100000      ! N/mm2 Young's modulus
t = 6.00        ! mm thickness
w = 0.30        ! Poisson's ratio
p =    -0.278483    ! N/mm2 external pressure
aa = 1100.00    ! mm horizontal radius at u = 0
bb = 3000.00    ! mm vertical radius
lz = 4000.00    ! mm model height
lu = 2*asin(lz/2/bb) !mm
lv = 2*pi       ! model perimeter at u = 0
nu = 2*NINT(ABS(aa)/SQRT(ABS(aa)*t))
nv = 3*nu       ! number of elements along v-axis
alpha = 0.25    ! ratio of lu that is not loaded by p
delta = 5.00*t ! prescribed imperfection magnitude
*ULIB,ShellBucklingLibrary,mac
*USE,ShellBuckling,pi,E,t,w,p,aa,bb,lz,lu,nu,nv,alpha,delta
/CLEAR
```

Figure 3.25. *ShellBucklingInput.inp*

Python then invokes MAPDL and passes *ShellBucklingInput.inp* via the subprocess module. Once an analysis finishes, λ_C is read from the last line of *ShellBuckling.csv* and p is updated for the next analysis per

$$p = 1.2 \cdot \lambda_C \cdot p\,,$$

where 1.2 is arbitrarily chosen to be large enough that λ_S is reached before all the load is applied (or else the analysis will simply terminate at $\lambda = 1$) and small enough so that there are enough load steps leading up to λ_S. When crafting loops for RunAPDL, some forethought is in order:

1) Avoid choosing too small values of any variable, especially a and t to avoid large n_u and n_v.
2) Negative values of a generate models with positive k_G. E.g., – 2000 mm < a < – 5000 mm.
3) Positive values of a generate models with negative k_G. E.g., 2000 mm < a < 5000 mm.
4) 2b should be larger than l_z. E.g., if l_z is 4000 mm, choose 3000 mm < b < 10000 mm.
5) The shell should be thin, i.e., 30 < a/t < 4000. If a is 2000 mm, choose 1 mm < t < 10 mm.
6) Choose $\bar{\delta}$ according to exponentially decreasing λ_S with increasing $\bar{\delta}$, e.g., [0.01t, 0.1t, 0.5t, 1t, 2t].
7) The change in variable size should be gradual for $p = 1.2 \cdot \lambda_C \cdot p$ to work properly.
8) The initial value of p should be negative to apply the pressure from the outside inwards.

Points 2) and 3) are illustrated by Fig. 3.26.

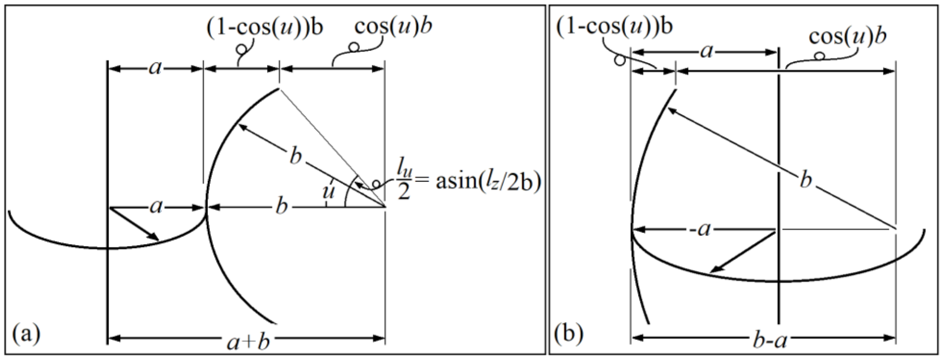


Figure 3.26. (a) Model with negative Gausian curvature (b) model with positive Gaussian curvature

Figure 3.27 shows a possible assembly of loops. This example generates negatively curved shells.

```
w = 0.3         # Poisson’s ratio
p = -0.1        # Initial external pressure (note: negative)
lz = 4000       # Model height (Can also be varied)
alpha = 0.25    # Half of unloaded area
modulus = [30000,60000,100000,200000] # MPa
thickness = np.linspace(5,10,6)       # mm
a = np.linspace(1000,2000,11)         # mm
b = np.linspace(3000,6000,16)         # mm
delta_init = [0.01,0.1,0.5,1,2]       # as a ratio of t

for E in modulus:
    for t in thickness:
        for aa in a:
            for bb in b:
                for delta in delta_init:
                    p = RunAPDL(E,t,w,p,aa,bb,lz,alpha,delta)
```

Figure 3.27. Loops to iterate parameters around RunAPDL

Additionally, it is possible to change the boundary conditions inside the MAPDL script, or even add axial loads (as with the axially compressed thin-shell cylinder from subsection 2.2.4.2). Appendix L contains an alternative version of the Python script written for MATLAB.

3.7. Torus Model

Figure 3.28 shows the torus model mentioned in section 3.2, which was considered and analyzed.

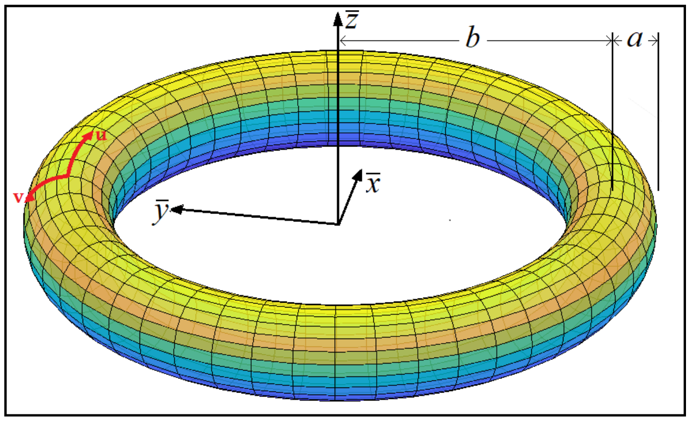


Figure 3.28. Complete torus model

The complete Torus model is parametrized by

$$\mathrm{r}(u,v)=\begin{Bmatrix}\bar{x}(u,v)\\ \bar{y}(u,v)\\ \bar{z}(u)\end{Bmatrix}=\begin{Bmatrix}\left(b+a\cdot\sin\left(\frac{v}{a}\right)\right)\sin\left(\frac{u}{b}\right)\\ \left(b+a\cdot\sin\left(\frac{v}{a}\right)\right)\cos\left(\frac{u}{b}\right)\\ a\cdot\cos\left(\frac{v}{a}\right)\end{Bmatrix}, \qquad (3.14)$$

where $0 \le v < 2\pi a$ and $0 \le u < 2\pi b$. It was envisaged this model would offer a gamut of curvature dyads ranging from positive to negative values of k_G. A linear elastic solution for the membrane forces of an externally pressurized Torus shell was obtained analytically and using FEM (section 4.1). The analytically derived membrane forces are rather uniform. Localized additional membrane forces were later added to a segment of the torus as shown in Fig. 3.29 to instigate localized buckling.

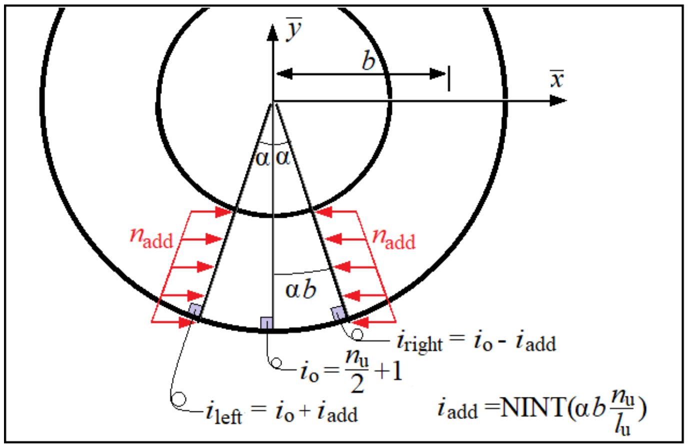

$\bar{y}$
b
$\bar{x}$
α
α
n_{add}
n_{add}
αb
$i_{\text{right}} = i_o - i_{\text{add}}$
$i_o = \frac{n_u}{2} + 1$
$i_{\text{left}} = i_o + i_{\text{add}}$
$i_{\text{add}} = \text{NINT}(\alpha b \frac{n_u}{l_u})$

Figure 3.29. Additional forces on torus model

4. RESULTS

4.1. Torus Results

An analytical solution for the linear elastic membrane forces of a torus shell subject to uniform external pressure is presented hereinafter. Per (Hoogenboom, 2017, p. 30), the torus curvatures are

$$k_{xx} = -\left(a + \frac{b}{\sin(v/a)}\right)^{-1}, \tag{4.1}$$

$$k_{yy} = -\frac{1}{a} \tag{4.2}$$

and

$$k_{xy} = 0. \tag{4.3}$$

The Lamé parameters, in turn, are given by

$$\alpha_x = 1 + \frac{a}{b}\sin\left(\frac{v}{a}\right) \tag{4.4}$$

and

$$\alpha_y = 1. \tag{4.5}$$

Equations (2.7) and (2.8), and use of chain rule, yield the in-plane curvatures

$$k_x = \frac{1}{\alpha_x} \cdot \frac{\partial \alpha_x}{\partial y} = \frac{1}{\alpha_x \alpha_y} \cdot \frac{\partial \alpha_x}{\partial v} = \cos\left(\frac{v}{a}\right) \cdot \left(a \sin\left(\frac{v}{a}\right) + b\right)^{-1} \tag{4.6}$$

and

$$k_y = \frac{1}{\alpha_x \alpha_y} \cdot \frac{\partial \alpha_y}{\partial u} = 0. \tag{4.7}$$

Lastly, the external pressure is defined in local coordinates as

$$p_x = p_y = 0 \text{ and } p_z = -p. \tag{4.8}$$

It is assumed that n_{xx} is a constant obtained by Barlow's formula (Hoogenboom, 2017, p. 8) and n_{yy} is a function of *v*, i.e., $n_{yy} = n_{yy}(v)$. Force equilibrium at the cut in Fig. 4.1 yields

$$n_{xx} = \frac{p \cdot a}{2}. \tag{4.9}$$

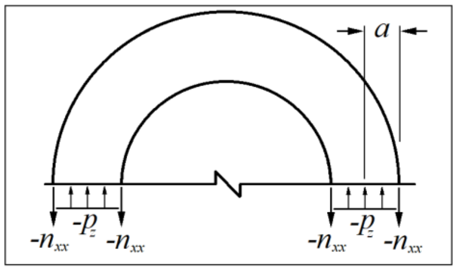


Figure 4.1. Calculation based on Barlow's formula

Substituting the above relations into Eq. (2.12) yields

$$\frac{\partial n_{yy}}{\partial v} = (n_{xx} - n_{yy}) \cdot k_x = (n_{xx} - n_{yy}) \cdot \cos\left(\frac{v}{a}\right) \cdot \left(a \cdot \sin\left(\frac{v}{a}\right) + b\right)^{-1}, \tag{4.10}$$

whose solution is

$$n_{yy} = \frac{\int \cos\left(\frac{v}{a}\right) \cdot n_{xx} \cdot dv + C}{a \cdot \sin\left(\frac{v}{a}\right) + b} = \frac{\int \cos\left(\frac{v}{a}\right) \cdot \frac{p \cdot a}{2} \cdot dv + C}{a \cdot \sin\left(\frac{v}{a}\right) + b} = \frac{p \cdot a}{2} \cdot \frac{a \cdot \sin\left(\frac{v}{a}\right) + C}{a \cdot \sin\left(\frac{v}{a}\right) + b}. \tag{4.11}$$

Further substitution of Eq. (4.9) and (4.11) into (2.10) yields $C = 2b$, i.e., n_{yy} is given by

$$n_{yy} = \frac{p \cdot a}{2} \cdot \left[a \cdot \sin\left(\frac{v}{a}\right) + 2b\right] \cdot \left[a \cdot \sin\left(\frac{v}{a}\right) + b\right]^{-1}. \tag{4.12}$$

Substituting Eq. (4.9) and (4.12) into (2.10), (2.11), and (2.12) shows their correctness. Furthermore, they form the only solution. Fig. 4.2 compares the analytical solution to the FEM solution of a torus model with E = 10000MPa, t = 5mm, v = 0.2, a = 2000mm, b = 10000mm, p = 0.001MPa, n_v = 60 and n_u = 180.

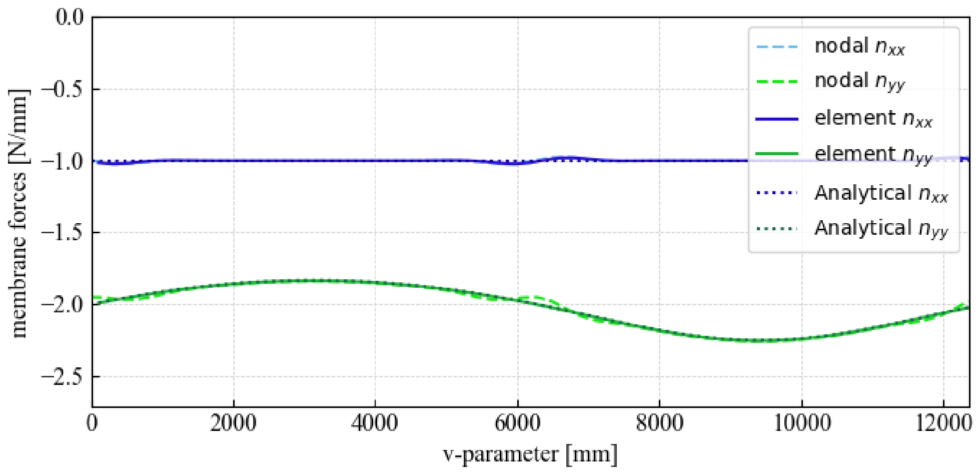


Figure 4.2. Torus linear elastic membrane forces

Additional forces were added per Fig. 4.3. The result of adding forces at an angle of 10 degrees to each side ($\alpha = 10^o$) with a magnitude of half the analytical n_{xx} is plotted in Fig. 4.3.

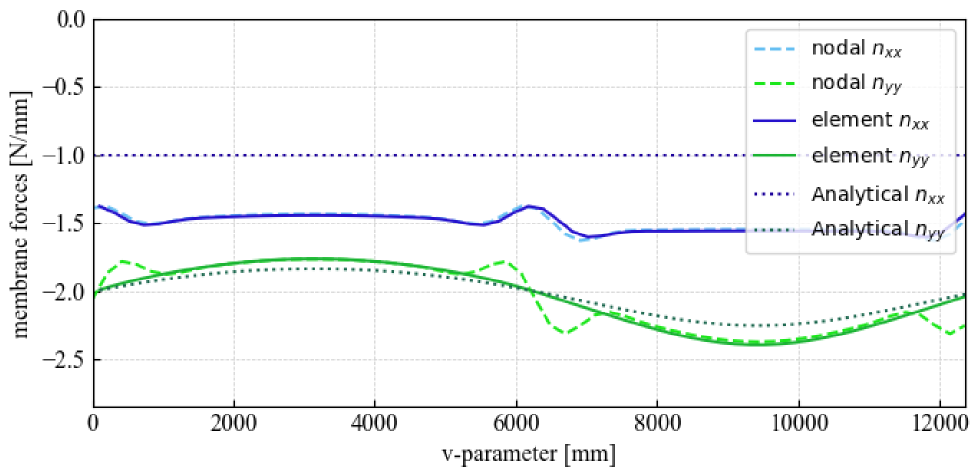


Figure 4.3. Torus linear elastic membrane forces, $n_{\text{add}} = 0.5\ n_{xx}$

Augmenting n_{xx} due to pressure only by n_{add} yields an expected total of about $1.5n_{xx}$ due to pressure alone. n_{xx} also changes abruptly wherever $k_m = 0$ (at v = 0mm and v = 6283 mm). n_{yy}, on the other hand, sees very slight difference except for a noticeably discrepancy between the results extracted directly at the element level and at the nodal level. Very similar patterns are observed for the case in which $\alpha = 10^o$ and $n_{add} = n_{xx}$.

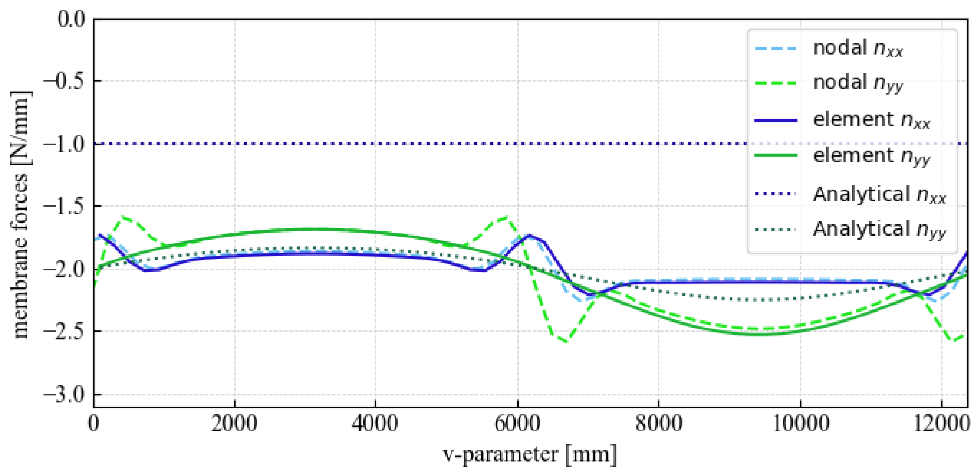


Figure 4.4. Torus linear elastic membrane forces, $n_{\text{add}} = 1.0\ n_{xx}$

The discrepancy between the nodal and element results shall unfortunately remained unexplained, but the presence of bending moments at the locations where $k_G = 0$ may have had some influence. Overall, n_{add} yields load peaks wherever $k_G = 0$. Not surprisingly, buckling occurs at this location over the entire circumference. Remarkably, n_{add} has a negligible influence on the buckling modes and critical load. It is for this reason that the complete torus model was demoted and the script was instead provided for the toroidal shell segments which shows much more uniform membrane forces and much more localized sinusoidal buckling patterns. Figure 4.5 shows the buckling modes for $n_{\text{add}} = 0.5\ n_{xx}$.

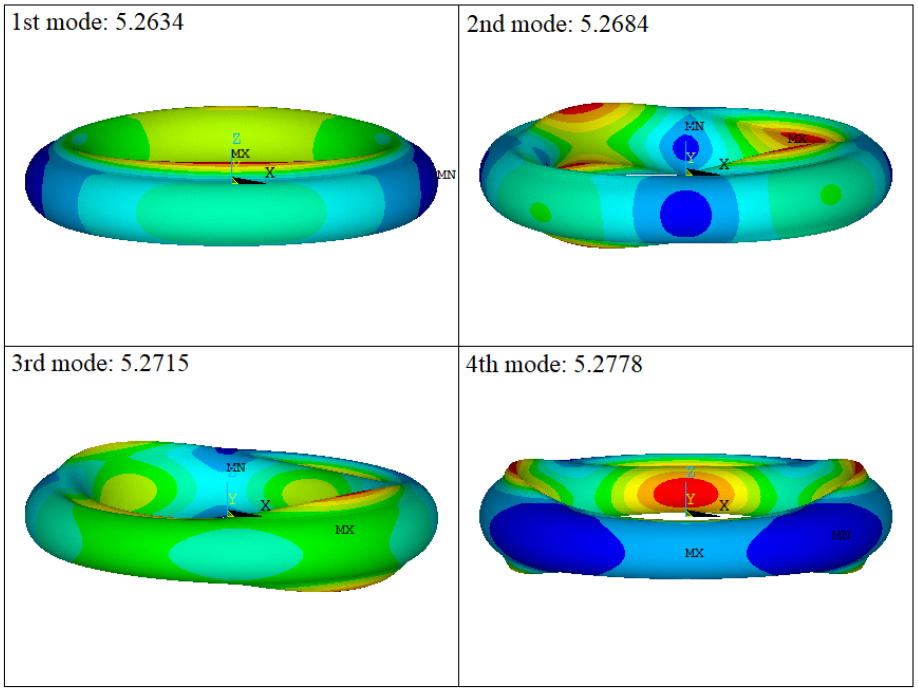


Figure 4.5. Torus linear buckling modes, $n_{\text{add}} = 0.5n_{xx}$

4.2. Cylinder GNIA Results

The capabilities of the MAPDL *ShellBuckling* macros were tested by comparing its results to the notorious axially compressed thin-shell cylinder. The cylindrical model was achieved by setting the input parameters to E = 100000MPa, t = 5mm, ν = 0.3, a = 2000mm, b = 50000000mm, l_u = 5000mm, n_u = 30, n_v = 90 and replacing the external pressure by downward force applied at the top nodes. The force on each node equaled $2 \cdot 0.6 \cdot \pi \cdot E \cdot t^2 / n_v$, such that the resulting n_{xx} would equal the buckling load predicted by Eq. (2.13). The first three buckling modes corresponding to λ_C can be seen in Fig. 4.6.

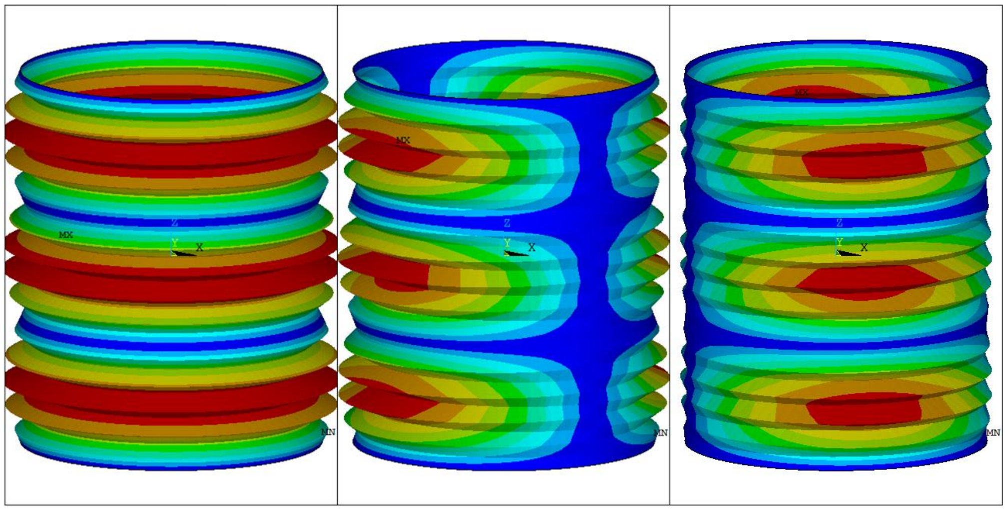


Figure 4.6. Thin-shell cylinder buckling modes

The initial imperfections are based on the non-axisymmetrical (second) buckling mode. The results of increasing the initial imperfections are shown in Fig. 4.7. The results are compared to Eq. (2.20) and (2.21).

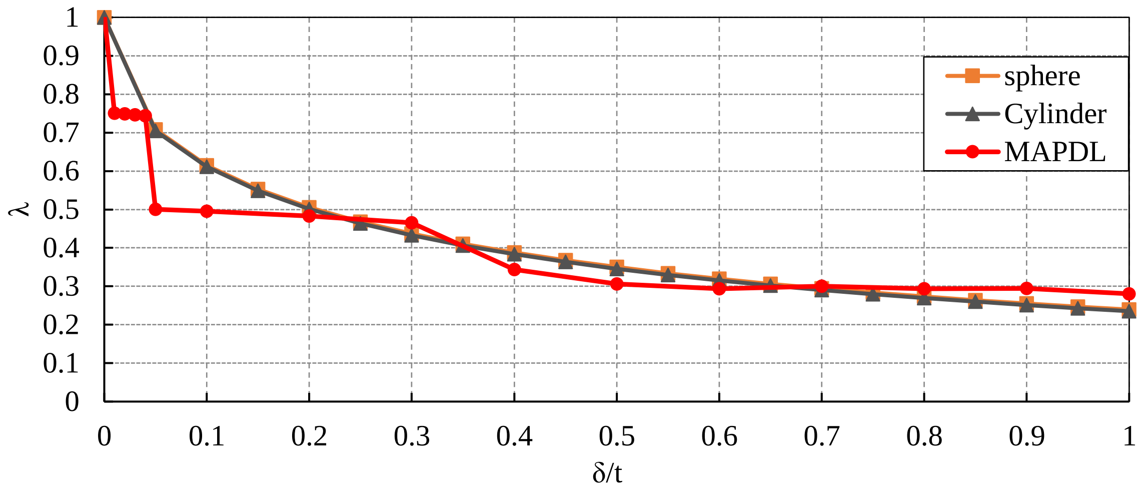


Figure 4.7. Thin-shell cylinder imperfection sensitivity

There is reasonable agreement between the theoretical results and the finite element analysis results, especially at values of δ/*t* between 0.2 and 1.

4.3. Example Results from the Python Script

An excerpt of the test runs that have been performed is presented herein. Table 4.1 contains all prescribed fields from Fig. 3.1. The last column contains the percentage difference between the absolute deflection (defl_max) of the node with maximum deflection and the local *z* displacement (uz_max) of that same node after the LBA. This value should ideally be small (2~10). In the rare occurrence this value is large, it means the geometry update for the GNIA did not proceed as planned and discarding the results of said analysis would be recommended.

Table 4.1. Excerpt of test runs

E	t	v	a	b	l_v	n_v	l_z	n_u	δ	$n_{xx,avg}$	$n_{yy,avg}$	$n_{xy,avg}$	n_{xx}	n_{yy}	n_{xy}	λ_C	λ_S	%
100000	5	0.3	1100	3000	6.283	90	4000	30	0.5	35.0042	-196.1515	0	20.4376	-188.8473	0	0.83055	0.96321	8.7
100000	5	0.3	1100	3000	6.283	90	4000	30	2.5	35.0042	-196.1515	0	20.4376	-188.8473	0	0.83055	0.94579	8.7
100000	5	0.3	1100	3000	6.283	90	4000	30	5	35.0042	-196.1515	0	20.4376	-188.8473	0	0.83055	0.94649	9.2
100000	5	0.3	1100	3000	6.283	90	4000	30	10	35.0042	-196.1515	0	20.4376	-188.8473	0	0.83055	0.94558	9.1
100000	5	0.3	1100	3000	6.283	90	4000	30	25	34.8872	-195.496	0	20.3693	-188.2161	0	0.83334	0.82731	9.1
100000	5	0.3	1100	3200	6.283	90	4000	30	0.5	30.1293	-187.5943	0	17.5912	-178.6537	0	0.85905	0.99994	1.5
100000	5	0.3	1100	3200	6.283	90	4000	30	2.5	31.8031	-198.0162	0	18.5685	-188.5789	0	0.81384	0.94303	3.2
100000	5	0.3	1100	3200	6.283	90	4000	30	5	31.8031	-198.0162	0	18.5685	-188.5789	0	0.81384	0.92708	3.4
100000	5	0.3	1100	3200	6.283	90	4000	30	10	33.3459	-207.6221	0	19.4693	-197.727	0	0.77619	0.91111	3.2
100000	5	0.3	1100	3200	6.283	90	4000	30	25	31.0594	-193.3858	0	18.1343	-184.1691	0	0.83333	0.87672	1.5
100000	5	0.3	1100	3400	6.283	90	4000	30	0.05	42.7961	-292.3634	0	24.9879	-276.0395	0	0.54967	0.7033	8.6
100000	5	0.3	1100	3400	6.283	90	4000	30	0.5	28.2284	-192.8436	0	16.4821	-182.0763	0	0.83333	0.99855	4.5
100000	5	0.3	1100	3400	6.283	90	4000	30	2.5	28.2284	-192.8436	0	16.4821	-182.0763	0	0.83333	0.99174	3
100000	5	0.3	1100	3400	6.283	90	4000	30	5	28.2283	-192.8425	0	16.482	-182.0753	0	0.83333	0.99335	3.9
100000	5	0.3	1100	3400	6.283	90	4000	30	10	28.2281	-192.8415	0	16.4819	-182.0743	0	0.83334	0.95321	4
100000	5	0.3	1100	3600	6.283	90	4000	30	0.05	26.1291	-193.9661	0	15.2574	-181.8573	0	0.80555	0.96819	1.4
100000	5	0.3	1100	3600	6.283	90	4000	30	0.5	25.2579	-187.4994	0	14.7488	-175.7944	0	0.83334	0.99763	1
100000	5	0.3	1100	3600	6.283	90	4000	30	2.5	25.2581	-187.5005	0	14.7488	-175.7954	0	0.83333	0.99595	1.3
100000	5	0.3	1100	3600	6.283	90	4000	30	5	25.2579	-187.4994	0	14.7488	-175.7944	0	0.83334	0.98907	1.3
100000	5	0.3	1100	3600	6.283	90	4000	30	10	25.2582	-187.5015	0	14.7489	-175.7964	0	0.83333	0.9634	1
100000	5	0.3	1100	3800	6.283	90	4000	30	0.05	23.5343	-188.3836	0	13.7435	-175.5999	0	0.81709	0.99861	4.1
100000	5	0.3	1100	3800	6.283	90	4000	30	0.5	23.0756	-184.7118	0	13.4757	-172.1772	0	0.83333	0.99686	4.7
100000	5	0.3	1100	3800	6.283	90	4000	30	2.5	23.0754	-184.7107	0	13.4756	-172.1762	0	0.83334	0.99266	4.4
100000	5	0.3	1100	3800	6.283	90	4000	30	5	23.0757	-184.7128	0	13.4757	-172.1782	0	0.83333	0.9684	4.6
100000	5	0.3	1100	3800	6.283	90	4000	30	10	23.0756	-184.7118	0	13.4757	-172.1772	0	0.83333	0.97945	3.9
100000	5	0.3	1100	4000	6.283	90	4000	30	0.05	21.6182	-185.4265	0	12.6259	-171.9987	0	0.82366	0.99961	0.6
100000	5	0.3	1100	4000	6.283	90	4000	30	0.5	21.3672	-183.2741	0	12.4793	-170.0021	0	0.83334	0.99935	0.6
100000	5	0.3	1100	4000	6.283	90	4000	30	2.5	21.3675	-183.2762	0	12.4795	-170.0041	0	0.83333	0.99781	0
100000	5	0.3	1100	4000	6.283	90	4000	30	5	21.3673	-183.2751	0	12.4794	-170.0031	0	0.83333	0.97068	0
100000	5	0.3	1100	4000	6.283	90	4000	30	10	21.3672	-183.2741	0	12.4793	-170.0021	0	0.83334	0.97018	0

5. CONCLUSIONS

This treatise caters the MAPDL macros and Python code required to execute a batch of nonlinear finite element analyses to generate a database of the nonlinear buckling loads of shell portions with varying membrane forces, curvatures and magnitude of initial geometric imperfections. Some limitations deserve attention.

While the input files prescribe enough number of elements and applied pressure to facilitate convergence, it is difficult to manually verify whether all analyses reached λ_S. To facilitate this task, the macros writes the data points of all force-displacement curves to *ShellGNIA.csv*. A new (Python) code could be written to examine said data points and verify whether the load did indeed plateau or (numerical) divergence occurred at an earlier stage.

Currently the code only employs a single buckling mode to update the model geometry in preparation for the GNIA. It may be imperative to modify that code section to include a wider gamut of buckling modes, using either Eq. (2.24), (2.25) or another method.

Another complication is the duration of each analysis. Test runs have recorded times between 300 to 1000 seconds per analysis. The analyses' duration gets especially large for shells with large radii and small thickness as more elements are needed. Given the sheer number of parameters that need to be varied to generate a comprehensive database the overall duration of the analyses could easily be prolonged.

After completing the database, the task to fit equations to the data could be best accomplished using specialized software or code, and visualizing the effect of individual parameters or pairs of parameters on λ_S by generating various two- and three-dimensional plots.

REFERENCES

ANSYS Inc. (2017). *ANSYS® Academic Research Mechanical, Release 18.1* [Computer software]. Canonsburg, PA., U.S. Retrieved from TU Delft student software portal: https://software.tudelft.nl/

ANSYS Inc. (2009). Command Reference for the Mechanical APDL. *ANSYS Manual*, 15317, 1-1920. Canonsburg, PA: ANSYS, Inc.

ANSYS Inc. (2009). Theory Reference for the Mechanical APDL and Mechanical Applications, *ANSYS Manual*. 3304, Canonsburg, PA: ANSYS, Inc.

Bažant, Z.P., Cedolin L. (1991). *Stability of Structures, Elastic, Inelastic, Fracture and Damage theories*, Oxford University Press, New York, 1991.

Budiansky, B., & Hutchinson, J.W. (1966). A Survey of Some Buckling Problems. *AAIA Journal, 4*(9), 1505-1510.

Budiansky, B., & Amazigo, J.C. (1969). Initial Postbuckling Behavior of Cylindrical Shells Under External Pressure. *Journal of Mathematics and Physics. 47,* 223-235

Blaauwendraad, J., & Hoefakker, J.H. (2013). *Structural Shell Analysis: Understanding and Application* (Solid mechanics and its applications, v. 200). Dordrecht, The Netherlands: Springer.

Borst, R., & Crisfield, M. (2012). *Nonlinear finite element analysis of solids and structures* (2nd ed., Wiley series in computational mechanics). Chichester, West Sussex, United Kingdom: Wiley.

Bushnell, D. (1981). Buckling of Shells-Pitfall for Designers. *AIAA Journal*, *19*(9), 1183-1226.

Chen, T. (2014). *On Introduction Imperfection in the Non-Linear Analysis of Buckling of Thin-shell Structures* (Master's Thesis) TU Delft, Delft, The Netherlands

Cox, H.L. (1940). Stress Analysis of Thin Metal Construction, *Journal Royal Aeronautical Society. 44*, 231

Danielson, D.A. (1970). Buckling and Initial Postbuckling Behavior of Spheroidal Shells Under Pressure. *AAIA Journal*, *7*(5), 936-944.

Hoogenboom. P.C.J. (2017). *CIE4143 Shell Analysis, Theory and Application.* [lecture notes]. Retrieved from http://homepage.tudelft.nl/p3r3s/b17_schedule.html

Hunter J., (2007). Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering*, *9*(3), 90-94.

Hutchinson, J.W. (1967a). Imperfection Sensitivity of Externally Pressurized Spherical Shells. *Journal of Applied Mechanics*, *34*(1), 49-55

Hutchinson, J.W. (1967b). Initial Postbuckling Behavior of Toroidal Shell Segments. *International Journal of Solids and Structures*, *3*, 97-115.

Hutchinson, J.W. (1968). Buckling and Initial Postbuckling Behavior of Oval Cylindrical Shells Under Axial Compression. *Journal of Applied Mechanics, 35*(1), 66-72

Hutchinson, J.W., & Koiter W.T. (1970). Postbuckling Theory, *Applied Mechanics Reviews, 23*(12), 1353-1363.

Kármán, T. & Tsien, H.S. (1939). The Buckling of Spherical Shells by External Pressure. *Journal of the Aeronautical Sciences, 7*(2), 43-50.

Kármán, T. & Tsien, H.S. (1941). The Buckling of Thin Cylindrical Shells Under Axial Compression. *Journal of the Aeronautical Sciences, 8*(8), 303-312.

Koiter, W.T. (1970). On the Stability of Elastic Equilibrium. Tech. Rep. No. AFFDLTR-70-25, Ohio: Air Force Flight Dynamics Laboratory, Wright-Patterson Air Force Base.

McGuire, W., Gallagher, R. H., & Ziemian, R.D., (2000). *Matrix Structural Analysis* (2nd Ed.). Retrieved from: http://mastan2.com/textbook.html

The MathWorks, Inc. (2017). MATLAB® *Release 2017a* [Computer software]. Natick, MA, U.S. License retrieved from TU Delft student software portal: https://software.tudelft.nl/

Voyiadjis, G.Z., & Woelke, P. (2008). *Elasto-Plastic and Damage Analysis of Plates and Shells*. Berlin: Springer.

Weingarten, V.I., Morgan, E.J., & Seide, P. (1965). Elastic Stability of Thin-Walled Cylindrical and Conical Shells under combined Internal Pressure and Axial Compression. *AIAA Journal*, *3*, 500-505.

Ziemian, R. D. (ed.) (2010). *Guide to stability design criteria for metal structures*. (6th Ed.). Structural Stability Research Council, Hoboken, N.J., U.S.: John Wiley and Sons.

APPENDIX A. ShellBuckling

```
ShellBuckling
!%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
! ShellBuckling
! Date:        22 November 2017
! Author:      Erik Giesen Loo
!
! Performs LE, LBA and GNIA of a toroidal shell segment created using ShellModel.
!
! Subroutines:
!  ShellModel
!  ShellLinearElasticResults
!     NodalMembraneForces (optional)
!        TransformationMatrix (optional)
!  ShellBucklingResults
!     PlotBucklingResults (optional)
!     ShellMatrixTransformation
!     NodalDisplacements (optional)
!        TransformationMatrix (optional)
!  GeomNonlinearAnalysis
!  GeomNonlinearAnalysisResults
!
!%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

! Preprocessing
!----------------------------------------------------------------------
/UIS,MSGPOP,4                        ! Sets pop-ups to YES
/UIS,ABORT,OFF                       ! No pop-ups about status of operation in progress
*USE,ShellModel                      ! Create toroidal segment

! Linear Elastic Analysis (Find stresses for buckling analysis)
!----------------------------------------------------------------------
/SOLU
ANTYPE, STATIC                       ! Linear elastic analysis
PSTRES, ON                           ! Prestress effects to be included in buckling analysis
SOLVE
FINISH
*USE,ShellLinearElasticResults      ! Membrane forces

! Linear Buckling Analysis (Find buckling modes and buckling loads)
!----------------------------------------------------------------------
/SOLU
ANTYPE, BUCKLE                       ! Linear buckling analysis
BUCOPT, LANB, 5,0,,CENTER            ! Block Lanczos method, 5 buckling modes
SOLVE
FINISH
*USE,ShellBucklingResults           ! Buckling mode & lambdaC

! Geometrically Nonlinear Analysis with Initial Geometrical Imperfections
!----------------------------------------------------------------------
*USE,GeomNonlinearAnalysis
*USE,GeomNonlinearAnalysisResults

/EOF
```

APPENDIX B. ShellModel

```
ShellModel
!%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
! Shell Model
! Date:        22 November 2017
! Author:      Erik Giesen Loo
! Creates the model of a toroidal shell segment.
!
! Called by:
!  ShellBuckling
!
! Subroutines:
!  None
!
! Input:
!  t = thickness
!  E = Young's modulus
!  w = Poisson's ratio
!  p = external pressure
!  aa = Horizontal radius at u = 0, kyy = -1/aa
!  bb = Vertical radius, kxx = 1/bb
!  nu = number of elements along u-axis
!  nv = number of elements along v-axis
!  lu = Total length of parameter u, from -lu/2 to lu/2
!  lv = Model perimeter at u = 0 as a ratio of aa, i.e., 2*pi
!  alpha = half of ratio of lu that is not loaded by constant pressure
!
! Output:
!  None
!
!%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

/PREP7
/VIEW, ALL, 0, -1, 0.2              ! all windows: camera at point (0,-1,0.2)

! Create element type: shell
!---------------------------------------------------------------------
ET,1,SHELL281                       ! element type: 8 node quadrilateral
R,1,t,t,t,t, , ,                    ! element thickness

! Create material properties
!---------------------------------------------------------------------
MP,EX,1,E                           ! Elastic modulus (linear elastic material)
MP,PRXY,1,w                         ! Poisson's ratio (linear elastic material)

! Create Shell nodes
!---------------------------------------------------------------------
! ty and tx are dummy variables that, added together, act like a Boolean indicating when
! a node should be created. If ty + tx is less than 1, then a node is created.
ty=1
*DO,nj,0,2*nv-1      ! Nodal columns going from nj = 0 to 2*nv-1, i.e., (2*nv) columns
   ty=-ty
   tx=1
   v = nj*lv/nv/2                         ! Parameter v (U-V plane)
   *DO,ni,-nu,nu     ! Nodal rows going from ni = -nu to nu, i.e., (2*nu) rows
      tx=-tx
      *IF,tx+ty,LT,1,THEN                 ! if tx+ty = 2 omit node creation
         u = ni*lu/nu/2                   ! Parameter u (U-V plane)
         x = (aa+(1-COS(u))*bb)*COS(v)    ! x-coordinate x = x(u,v)
         y = (aa+(1-COS(u))*bb)*SIN(v)    ! y-coordinate y = y(u,v)
         z = bb*SIN(u)                    ! z-coordinate z = z(u)
         N,,x,y,z,,,                      ! Create node
      *ENDIF
   *ENDDO
*ENDDO
```

```
! Create Shell elements
!----------------------------------------------------------------------
SHPP,OFF                                  ! no warning aspect ratio
*DO,j,1,nv                                ! j-th element column (along v-axis)
    *DO,i,1,nu                            ! i-th element row (along u-axis)
      k1 = 1 + 2*(i-1) + (j-1)*(3*nu+2)
      k2 = i + 2*j + (3*j - 1)*nu - 1
      *IF,j,LT,nv,THEN                    ! j < nv
         k3 = 1 + 2*(i-1) + j*(3*nu+2)
      *ELSE                               ! j = nv
         k3 = 1 + 2*(i-1)
      *ENDIF
      E,k3,k3+2,k1+2,k1,k3+1,k2+1,k1+1,k2
   *ENDDO
*ENDDO

! Create Dirichlet boundary conditions
!----------------------------------------------------------------------
! Top and bottom rings
*DO,j,1,nv                                ! j-th element column (along v-axis)
   n_bot = 1 + (j-1)*(3*nu+2)             ! n_bot = k1(i = 1,j)
   n_top = 3 + 2*(nu-1) + (j-1)*(3*nu+2)  ! n_top = k1(i = nu,j) + 2
   D,n_top,UX,0,,,,UY,
   D,n_bot,UX,0,,,,UY,UZ
*ENDDO

! Create Neumann boundary conditions
!----------------------------------------------------------------------
ESEL,S,ELEM,,nu/2
*DO,j,1,nv                                ! j-th element column (along v-axis)
   i_bot = 1+(j-1)*nu+NINT(alpha*nu)      ! i_bot-th element row
   i_top = j*nu-NINT(alpha*nu)            ! i_top-th element row
   ESEL,A,ELEM,,i_bot,i_top,1             ! Append elements from i_bot to i_top in steps of 1
*ENDDO
SFE, ALL, 1, PRES, 0, p,,,                ! Add uniform pressure on all selected elements
ALLSEL                                    ! Reselect all elements

FINISH
/EOF
```

APPENDIX C. ShellLinearElasticResults

```
ShellLinearElasticResults
!%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
! ShellLinearElasticResults
! Date:       17 November 2017
! Author:     Erik Giesen Loo
! Writes model parameters and membrane forces to ShellBuckling.csv.
! Writes shell internal forces (elements 1 through nu) in Shell.txt.
!
! Called by:
!  ShellBuckling
!
! Subroutines:
!  NodalMembraneForces (Optional)
!     TransformationMatrix (Optional)
!
! Input:
!  E,t,w,aa,bb,lz,nu,lv,nv,alpha,delta
!
! Output:
!  nx,ny,nxy,nx_avg,ny_avg,nxy_avg
!
!%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
/POST1
ESEL,S,ELEM,,1,nu,1                  ! Select elements from 1 to nu in increments of 1
ETABLE,nxx,SMISC,1                   ! Extract shell nxx membrane force
ETABLE,nyy,SMISC,2                   ! Extract shell nyy membrane force
ETABLE,nxy,SMISC,3                   ! Extract shell nxy membrane force
ETABLE,mxx,SMISC,4                   ! Extract shell mxx moment
ETABLE,myy,SMISC,5                   ! Extract shell myy moment
ETABLE,mxy,SMISC,6                   ! Extract shell mxy moment
ETABLE,qx ,SMISC,7                   ! Extract shell qx shear force
ETABLE,qy ,SMISC,8                   ! Extract shell qy shear force
nx_avg = 0                           ! Re-set nx_avg to 0
ny_avg = 0                           ! Re-set ny_avg to 0
nxy_avg = 0                          ! Re-set nxy_avg to 0
i1 = NINT(alpha*nu)                  ! element number 1
i2 = NINT((1-alpha)*nu)              ! element number 2
*DO,nele,i1,i2
   *GET,nx,ETAB,1,ELEM,nele       $  nx_avg = nx_avg + nx
   *GET,ny,ETAB,2,ELEM,nele       $  ny_avg = ny_avg + ny
   *GET,nxy,ETAB,3,ELEM,nele      $  nxy_avg = nxy_avg + nxy
*ENDDO
nx_avg = nx_avg/(i2-i1+1)            ! nx_avg = Sum(nx)/(i2-i1+1)
ny_avg = ny_avg/(i2-i1+1)            ! ny_avg = Sum(ny)/(i2-i1+1)
nxy_avg = nxy_avg/(i2-i1+1)          ! nxy_avg = Sum(nxy)/(i2-i1+1)
*GET,nx,ETAB,1,ELEM,NINT(nu/2)
*GET,ny,ETAB,2,ELEM,NINT(nu/2)
*GET,nxy,ETAB,3,ELEM,NINT(nu/2)
*CFOPEN,ShellBuckling,csv,,APPEND
*VWRITE,E,t,w,aa,bb,lv,nv,lz,nu
(F10.0,',',F10.3,',',F10.3,',',F10.0,',',F10.0,',',F10.3,',',F10.0,',',F10.0,',',F10.0,',',$)
*VWRITE,delta,nx,ny,nxy,nx_avg,ny_avg,nxy_avg
(F10.3,',',F12.4,',',F12.4,',',F12.4,',',F12.4,',',F12.4,',',F12.4,',',$)
*CFCLOS ! The $ sign suppreses \n (new line command)

/OUTPUT,Shell,txt
PRETAB,nxx,nyy,nxy
PRETAB,qx,qy
PRETAB,mxx,myy,mxy
/OUT
ALLSEL
*USE,NodalMembraneForces
FINISH
/EOF
```

APPENDIX D. NodalMembraneForces (optional)

```
NodalMembraneForces
!%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
! NodalMembraneForces
! Date:        17 November 2017
! Author:      Erik Giesen Loo
! Writes 'nodal' membrane forces to ShellLE.csv.
!
! Called by:
!  ShellLinearElasticResults
!
! Subroutines:
!  TransformationMatrix
!
! Input:
!  E,t,w,p,aa,bb,lz,lu,nu,lv,nv
!
! Output:
!  nx, ny, nxy, qx, qy --> ShellLE.csv
!
!%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

*CFOPEN,ShellLE,csv,,
*VWRITE,
E,t,w,p,a,b,lz,nu,nv
*VWRITE, E,t,w,p,aa,bb,lz,nu,nv
%16.4f,%16.4f,%16.4f,%16.4f,%16.4f,%16.4f,%16.4f,%16.4f,%16.4f
*VWRITE,
z-coordinate,n_xx,n_yy,n_xy,q_x,q_y

*DO,i,1,nu+1                        ! From element 1 to 'imaginary' element nu+1
   nnode = 1+2*(i-1)                ! node number on X-Z plane (at j = 1)
   u = (i-1)*(lu/nu)-lu/2           ! Parameter u
   v = 0                            ! Parameter v (at j = 1)
   z = bb*sin(u)                    ! z-coordinate
   *GET,Sigx,NODE,nnode,S,X         ! Extract sigma x
   *GET,Sigy,NODE,nnode,S,Y         ! Extract sigma y
   *GET,Sigz,NODE,nnode,S,Z         ! Extract sigma z
   *GET,Sigxy,NODE,nnode,S,XY       ! Extract xy shear stress
   *GET,Sigyz,NODE,nnode,S,YZ       ! Extract yz shear stress
   *GET,Sigxz,NODE,nnode,S,XZ       ! Extract xz shear stress

   *DMAT,Sigma,D,ALLOC,3,3          ! Assemble global stress matrix
   *SET,Sigma(1,1),Sigx,Sigxy,Sigxz
   *SET,Sigma(1,2),Sigxy,Sigy,Sigyz
   *SET,Sigma(1,3),Sigxz,Sigyz,Sigz

   *USE,TransformationMatrix        ! Transformation matrix Gamma
   *MULT,Sigma, ,Gamma,TRANS,M3     ! M3 = transpose(Gamma)*Sigma
   *MULT,Gamma, ,M3, ,SigLocal      ! SigLocal = Gamma*M3

   nx = SigLocal(1,1)*t             ! nx = sigx*t
   ny = SigLocal(2,2)*t             ! ny = sigy*t
   nxy = SigLocal(1,2)*t            ! nxy = sigxy*t
   qx = SigLocal(1,3)*2*t/3         ! qx = max(sigxz)*1.5*t
   qy = SigLocal(2,3)*2*t/3         ! qy = max(sigyz)*1.5*t

   *CFOPEN,ShellLE,csv,,APPEND
   *VWRITE,z,nx,ny,nxy,qx,qy
%12.4f, %12.4f, %12.4f, %12.4f, %12.4f, %12.4f
   *CFCLOS
*ENDDO
/EOF
```

APPENDIX E. ShellBucklingResults

```
ShellBucklingResults
!%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
! ShellBucklingResults
! Date:        17 November 2017
! Author:      Erik Giesen Loo
! Writes the buckling load factor to ShellBuckling.txt.
! Outputs the maximum deflection and corresponding node.
!
! Called by:
!  ShellBuckling
!
! Subroutines:
!  PlotBucklingModes (optional)
!  TransformationMatrix
!  NodalDisplacements (optional)
!     TransformationMatrix (optional)
!
! Input:
!  E,t,w,p,aa,bb,lz,lu,nu,lv,nv
!
! Output:
!  lambdaC,defl_max,uz_max,nnode_max,defl_diff
!
!%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
/POST1
buckling_mode = 1
*USE,PlotBucklingModes              ! Plots 2 buckling modes (may be left commented out)
SUBSET,1,buckling_mode,FACT,,,,     ! Load buckling mode
*GET,lambdaC,ACTIVE,0,SET,FREQ      ! Get the buckling load factor lambdaC
*CFOPEN,ShellBuckling,csv,,APPEND
*VWRITE,lambdaC
(F11.5,',',$)
*CFCLOS

!Loop to read deflection of node next to ith element
uz_max = 0
defl_max = 0
*DO,j,1,nv                                         ! j-th element column (along v)
   *DO,i,1,nu+1                                    ! i-th element row (along u)
      nnode = 1 + 2*(i-1) + (j-1)*(3*nu+2)         ! nnode = k1(i,j)
      *GET,u_x,NODE,nnode,U,X                      ! Extract u_x from nnode
      *GET,u_y,NODE,nnode,U,Y                      ! Extract u_y from nnode
      *GET,u_z,NODE,nnode,U,Z                      ! Extract u_z from nnode
      defl = SQRT(u_x*u_x+u_y*u_y+u_z*u_z)         ! Absolute deflection
      u = (i-1)*(lu/nu)-lu/2                       ! u-parameter
      v = (j-1)*(lv/nv)                            ! v-parameter
      *USE,TransformationMatrix                    ! Assemble Gamma matrix
      *VEC,Defl_global,D,ALLOC,3,,,                ! Allocate space for Defl_global
      *SET,Defl_global(1),u_x,u_y,u_z              ! Let Defl_global = [u_x;u_y;u_z]
      *MULT, Gamma, , Defl_global, , Defl_local    ! Defl_local = Gamma*Defl_global
      u_z = Defl_local(3)                          ! local z-displacement
      *IF,ABS(u_z),GT,uz_max,THEN
         uz_max = ABS(u_z)                         ! uz_max = max(u_z)
         defl_max = ABS(defl)                      ! defl_max = max(defl)
         nnode_max = 1 + 2*(i-1) + (j-1)*(3*nu+2)  ! node with highest deflection
      *ENDIF
   *ENDDO
*ENDDO

defl_diff = (defl_max - uz_max)/uz_max*100         ! Control defl_max ~>> u_max
*USE,NodalDisplacements ! Saves nodal disp to ShellLBA.csv (may be left commented out)

FINISH
/EOF
```

APPENDIX F. PlotBucklingModes (optional)

```
PlotBucklingModes
!%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
! PlotBucklingModes
! Date:       17 November 2017
! Author:     Erik Giesen Loo
! Plots two consecutive buckling modes side-by-side.
!
! Called by:
!  ShellBucklingResults
!
! Subroutines:
!  None
!
! Input:
!  None
!
! Output:
!  None
!
!%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

! Change background colors
!--------------------------------------------------------------------------
/RGB,INDEX,100,100,100, 0           ! RGB for index 0
/RGB,INDEX, 80, 80, 80,13           ! RGB for index 13
/RGB,INDEX, 60, 60, 60,14           ! RGB for index 14
/RGB,INDEX, 0, 0, 0,15              ! RGB for index 15

! Create two windows
!--------------------------------------------------------------------------
/WINDOW, 1,LEFT                     ! Create window 1 on the left
/WINDOW, 2,RIGHT                    ! Create window 2 on the right

! Set camera location, angle, and distance
!--------------------------------------------------------------------------
/PLOPT,INFO,0
/VIEW, ALL, 0, -1, 0.2              ! all window: camera at point (0,-1,0.2)
/ANGLE, ALL, 0                      ! all windows: camera angle = 0 degrees
/DIST, ALL, AUTO                    ! all windows: distance = automatic

! Plot 1st buckling mode in window 1
!--------------------------------------------------------------------------
/WINDOW,2,OFF                       ! De-activate window 2
SUBSET,1,buckling_mode,FACT,,,,     ! Load step 1, buckling mode
PLNSOL, U,SUM, 0,1                  ! Contour plot of USUM = u_x+u_y+u_z

! Plot 2nd buckling mode in window 2
!--------------------------------------------------------------------------
/NOERASE                            ! Do not erase window 1
/WINDOW,1,OFF                       ! De-activate window 1
/WINDOW,2,ON                        ! Re-activate window 2
SUBSET,1,buckling_mode+1,FACT,,,,   ! Load step 1, buckling mode + 1
PLNSOL, U,SUM, 0,1                  ! Contour plot of USUM = u_x+u_y+u_z
/EOF
```

APPENDIX G. NodalDisplacements (optional)

```
NodalDisplacements
!%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
! NodalDisplacements
! Date:        28 October 2017
! Author:      Erik Giesen Loo
! Writes the buckling mode nodal displacements to ShellLBA.csv.
!
! Called By:
!  ShellBucklingResults
!
! Subroutines:
!  TransformationMatrix
!
! Input:
!  E,t,w,p,aa,bb,lz,nu,nv
!
! Output:
!  u_x,u_y,u_z,defl --> ShellLBA.csv
!
!%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

! Header
!---------------------------------------------------------------------
*CFOPEN,ShellLBA,csv,,
*VWRITE, E,t,w,p,aa,bb,lz,nu,nv
%12.4f,%12.4f,%12.4f,%12.4f,%12.4f,%12.4f,%12.4f,%12.4f,%12.4f
*CFCLOS

!Loop to read deflection of node next to ith element
!---------------------------------------------------------------------
*DO,j,1,nv                                          ! j-th element column (along v)
   *DO,i,1,nu+1                                     ! i-th element row (along u)
      nnode = 1 + 2*(i-1) + (j-1)*(3*nu+2)          ! nnode = k1(i,j)
      *GET,u_x,NODE,nnode,U,X                       ! Extract u_x from nnode
      *GET,u_y,NODE,nnode,U,Y                       ! Extract u_y from nnode
      *GET,u_z,NODE,nnode,U,Z                       ! Extract u_z from nnode
      u = (i-1)*(lu/nu)-lu/2                        ! u-parameter
      v = (j-1)*(lv/nv)                             ! v-parameter
      *USE,TransformationMatrix                     ! Assemble Gamma matrix
      *VEC,Defl_global,D,ALLOC,3,,,                 ! Allocate space for Defl_global
      *SET,Defl_global(1),u_x,u_y,u_z               ! Defl_global = [u_x;u_y;u_z]
      *MULT, Gamma, , Defl_global, , Defl_local     ! Defl_local = Gamma*Defl_global
      u_x = Defl_local(1)
      u_y = Defl_local(2)
      u_z = Defl_local(3)
      defl = SQRT(u_x*u_x+u_y*u_y+u_z*u_z)
      *CFOPEN,ShellLBA,csv,,APPEND
      *VWRITE,u_x,u_y,u_z,defl,
      %16.8f,%16.8f,%16.8f,%16.8f
      *CFCLOS
   *ENDDO
*ENDDO
/EOF
```

APPENDIX H. GeomNonlinearAnalysis

```
GeomNonlinearAnalysis
!%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
! GeomNonlinearAnalysis
! Date:        17 November 2017
! Author:      Erik Giesen Loo
! Performs a geometrically nonlinear analysis with imperfections (GNIA).
! It updates the geometry using buckling mode(s), then solves the GNIA.
!
! Called by:
!  ShellBuckling
!
! Subroutines:
!  None
!
! Input:
!  delta,uz_max
!
! Output:
!  nsubstep
!
!%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

! Update the geometry
!----------------------------------------------------------------------
/PREP7
FACTOR = delta/uz_max                         ! Factor for UPGEOM
UPGEOM,FACTOR,1,buckling_mode,'file','rst',  ! Add imperfections
/RESET   $  /ERASE   $     /REPLOT           ! Replot
FINISH

/SOLU
! Set analysis type: GNA
!----------------------------------------------------------------------
NCNV,0                              ! Do not terminate program if not-converged
NERR,,,-1                           ! Do not terminate analysis if not-converged
ANTYPE, STATIC                      ! Static analysis
NLGEOM, ON                          ! Nonlinear geometry
TIME, 1                             ! Time at the end of load step

! Set nonlinear controls / solution technique
!----------------------------------------------------------------------
ARCLEN,ON                           ! Arclength ON
nsubstep = 200                      ! # of substeps. Keep as par b/c it is used later in *VEC
NSUBST,nsubstep,,                   ! Number of substeps
NEQIT,50,                           ! Max. number of iterations
CNVTOL,STAT                         ! Convergence tolerance (default)

! Set output controls
!----------------------------------------------------------------------
RESCONTROL,DEFINE,ALL,1,            ! Write new files at every substep
OUTRES,NSOL,ALL,,,,                 ! Write nodal results at every substep
OUTRES,ESOL,ALL,,,,                 ! Write element results at every substep

! Solve
!----------------------------------------------------------------------
SOLVE
FINISH
/EOF
```

APPENDIX I. GeomNonlinearAnalysisResults

```
GeomNonlinearAnalysisResults
!%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
! GeomNonlinearAnalysisResults
! Date:        17 November 2017
! Author:      Erik Giesen Loo
! Writes the nonlinear buckling load factor to ShellBuckling.csv.
!
! Called by:
!  ShellBuckling
!
! Subroutines:
!  None
!
! Input:
!  nnode_max, nsubstep
!
! Output:
!  lambdaS
!  Defl, lambda --> ShellGNIA.csv
!
!%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
*DO,K,1,2

   /POST26
   nn = nsubstep
   ! Set vector arrays tinme, u_x, u_y, and u_z
   !-----------------------------------------------------------------------
   *DEL,time $*DIM,time,ARRAY,nn
   *DEL,u_x  $*DIM,u_x,ARRAY,nn
   *DEL,u_y  $*DIM,u_y,ARRAY,nn
   *DEL,u_z  $*DIM,u_z,ARRAY,nn
   *DEL,Defl $*DIM,Defl,ARRAY,nn

   ! Extract u_x, u_y, and u_z from ANSYS database
   !-----------------------------------------------------------------------
   VGET,time(1),1
   NSOL,2,nnode_max,U,X,      $VGET,u_x(1),2    !Var(2) = u_x
   NSOL,3,nnode_max,U,Y,      $VGET,u_y(2),3    !Var(3) = u_y
   NSOL,4,nnode_max,U,Z,      $VGET,u_z(3),4    !Var(4) = u_z

   ! Do vector operation on u_x, u_y, and u_z to get Defl
   !-----------------------------------------------------------------------
   *VOPER,u_x,u_x,MULT,u_x             ! u_x = u_x^2
   *VOPER,u_y,u_y,MULT,u_y             ! u_y = u_y^2
   *VOPER,u_z,u_z,MULT,u_z             ! u_z = u_z^2
   *VOPER,Defl,u_x,ADD,u_y             ! Defl = u_x^2+u_y^2
   *VOPER,Defl,Defl,ADD,u_z            ! Defl = u_x^2+u_y^2+u_z^2
   *VFUN,Defl,SQRT,Defl                ! Defl = SQRT(u_x^2+u_y^2+u_z^2)
   FINISH

*ENDDO

lambdaS = 0
*DO,k,1,nn
   *IF,time(k),GT,lambdaS,AND,time(k),LT,1,THEN
      lambdaS = time(k)
   *ENDIF
*ENDDO

*CFOPEN,ShellBuckling,csv,,APPEND
*VWRITE,lambdaS,defl_diff
(F11.5,',',F11.1)
*CFCLOS
```

```
*CFOPEN,ShellGNIA,csv,,APPEND
*VWRITE,
('E,t,w,p,aa,bb,lz,nu,nv,alpha,delta')
*VWRITE,E,t,w,p,aa
(F16.8,',',F16.8,',',F16.8,',',F16.8,',',F16.8,',',$)
*VWRITE,bb,lz,nu,nv,alpha,delta
(F16.8,',',F16.8,',',F16.8,',',F16.8,',',F16.8,',',F16.8)
*VWRITE,Defl(1),time(1)
(F16.8,',',F16.8)
*CFCLOS

/EOF
```

APPENDIX J. TransformationMatrix

```
TransformationMatrix
!%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
! TransformationMatrix
! Date:        17 November 2017
! Author:      Erik Giesen Loo
!
! It generates the global to local transformation matrix 'Gamma' for the model parametrized by:
!     x = (aa+(1-cos(u))*bb)*cos(v)
!     y = (aa+(1-cos(u))*bb)*sin(v)
!     z = bb*sin(u)
! The local x- and y- axes follow the u- and v- parameters, respectively.
!
! Called by:
!  NodalMembraneForces (optional)
!  ShellBucklingResults
!  NodalDisplacements (optional)
!
! Subroutines:
!  None
!
! Input:
!  u, v, aa, bb
!
! Output:
!  Gamma
!
!%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

! Assemble transformation matrix
*DMAT,Gamma,D,ALLOC,3,3

*VEC,i_vector,D,ALLOC,3                      ! Create first row (i' = dr/du)
   i_vector(1) = bb*sin(u)*cos(v)
   i_vector(2) = bb*sin(u)*sin(v)
   i_vector(3) = bb*cos(u)
   *NRM, i_vector, NRM2, norm_i, YES         ! Normalize

*VEC,j_vector,D,ALLOC,3                      ! Create second row (j' = dr/dv)
   j_vector(1) = -(aa+(1-cos(u))*bb)*sin(v)
   j_vector(2) =  (aa+(1-cos(u))*bb)*cos(v)
   j_vector(3) = 0
   *NRM, j_vector, NRM2, norm_j, YES         ! Normalize

*VEC,k_vector,D,ALLOC,3                      ! Create third row (k' = i' x j')
   k_vector(1) = i_vector(2)*j_vector(3) - i_vector(3)*j_vector(2)
   k_vector(2) = i_vector(1)*j_vector(3) - i_vector(3)*j_vector(1)
   k_vector(3) = i_vector(1)*j_vector(2) - i_vector(2)*j_vector(1)
   *NRM, k_vector, NRM2, norm_k, YES         ! Normalize

*DO,k,1,3                                    ! Assemble Gamma matrix
   Gamma(1,k) = i_vector(k)
   Gamma(2,k) = j_vector(k)
   Gamma(3,k) = k_vector(k)
*ENDDO
/EOF
```

APPENDIX K. Python Script

```
import numpy as np
import os, subprocess, csv, time

def RunAPDL(E,t,w,p,aa,bb,lz,alpha,delta):

    ansyspath = r'C:\Program Files\ANSYS Inc\v181\ansys\bin\winx64\MAPDL.exe'
    directory = r'C:\Users\Erik\Documents\ANSYS'
    jobname = 'file'
    memory = '4096'
    reserve = '1024'
    inputfile = r'C:\Users\Erik\Documents\ANSYS\ShellBucklingInput.inp'
    outputfile = r'C:\Users\Erik\Documents\ANSYS\OutputFile.txt'
    resultsfile = r'C:\Users\Erik\Documents\ANSYS\ShellBuckling.csv'
    lockfile = r'C:\Users\Erik\Documents\AnSYS\file.lock'

    start = time.clock()

    # Write input file
    input_parameters = ('/NERR,200,10000,,OFF,0 \n'
                        'pi = acos(-1) \n'
                        'E = {:6.0f}     ! N/mm2 Young\'s modulus\n'
                        't = {:4.2f}       ! mm thickness\n'
                        'w = {:3.2f}       ! Poisson\'s ratio\n'
                        'p = {:12.6f}    ! N/mm2 external pressure\n'
                        'aa = {:6.2f}    ! mm horizontal radius at u = 0\n'
                        'bb = {:6.2f}    ! mm vertical radius\n'
                        'lz = {:6.2f}    ! mm model height\n'
                        'lu = 2*asin(lz/2/bb) !mm \n'
                        'lv = 2*pi       ! model perimeter at u = 0 \n'
                        'nu = 2*NINT(ABS(aa)/SQRT(ABS(aa)*t)) \n'
                        'nv = 3*nu       ! number of elements along v-axis \n'
                        'alpha = {:4.2f}    ! ratio of lu that is not loaded by p \n'
                        'delta = {:4.2f}*t ! prescribed imperfection magnitude \n'
                        '*ULIB,ShellBucklingLibrary,mac \n'
                        '*USE,ShellBuckling,pi,E,t,w,p,aa,bb,lz,lu,nu,nv,alpha,delta \n'
                        '/CLEAR'
                        ).format(E,t,w,p,aa,bb,lz,alpha,delta)
    with open(inputfile,'w') as f:
        f.write(input_parameters)

    # Call ANSYS
    try:
        os.remove(lockfile)
        print('lock file removed')
    except:
        print('lock file does not exist')
    callstring = ('\"{}\" -p aa_t_a -dir \"{}\" -j \"{}\" -s read'
                  ' -m {} -db {} -t -d win32 -b -i \"{}\" -o \"{}\"'
                  ).format(ansyspath,directory,jobname,memory,reserve,inputfile,outputfile)
    print('Invoking ANSYS with', callstring)
    proc = subprocess.Popen(callstring).wait()

    # Update pressure field for next analysis
    with open(resultsfile,'r') as f:
        lambdaS = float(list(csv.reader(f))[-1][16])
    p = 1.2*lambdaS*p
    print('Updated pressure is',p,' N/mm2.')

    stop = time.clock()
    print('Elapsed time is ',stop-start,' seconds.')

    return(p)
```

APPENDIX L. MATLAB Script

```
function p = RunAPDL(E,t,w,p,aa,bb,lz,alpha,delta)
tic

%% Input Information: Modify when using on a different PC
ansyspath = 'C:\Program Files\ANSYS Inc\v181\ansys\bin\winx64\MAPDL.exe';
directory = 'C:\Users\Erik\Documents\ANSYS';
jobname = 'file';
memory = '4096';
reserve = '1024';
inputfile = 'C:\Users\Erik\Documents\ANSYS\ShellBucklingInput.inp';
outputfile = 'C:\Users\Erik\Documents\ANSYS\OutputFile.txt';
addpath C:\Users\Erik\Documents\ANSYS
resultsfile = 'ShellBuckling.csv';

%% Write input file
input_parameters = sprintf(['/NERR,200,10000,,OFF,0 \n',...
                        'pi = ACOS(-1) \n',...
                        'E = %6.0f     ! N/mm2 Youngs modulus\n',...
                        't = %4.2f       ! mm thickness\n',...
                        'w = %3.2f       ! Poissons ratio\n',...
                        'p = %12.6f  ! N/mm2 external pressure\n',...
                        'aa = %6.2f   ! mm horizontal radius at u = 0\n',...
                        'bb = %6.2f   ! mm vertical radius\n',...
                        'lz = %6.2f   ! mm model height\n',...
                        'lu = 2*asin(lz/2/bb) !mm \n',...
                        'lv = 2*pi      ! model perimeter at u = 0 \n',...
                        'nu = 2*NINT(ABS(aa)/SQRT(ABS(aa)*t)) \n',...
                        'nv = 3*nu      ! number of elements along v-axis \n',...
                        'alpha = %4.2f   ! ratio of lu that is not loaded by p \n',...
                        'delta = %4.2f*t ! prescribed imperfection magnitude \n',...
                        '*ULIB,ShellBucklingLibrary,mac \n',...
                        '*USE,ShellBuckling,pi,E,t,w,p,aa,bb,lz,lu,nu,nv,alpha,delta \n',...
                        '/CLEAR'],E,t,w,p,aa,bb,lz,alpha,delta);
disp(input_parameters)
fid = fopen(inputfile,'w');
fprintf(fid,input_parameters);
fclose(fid);

%% Call ANSYS
callstring = ['SET KMP_STACKSIZE=5120k & ',...
                    '"',ansyspath,'"',...
                    ' -b ',... batch
                    ' -p aa_t_a',... ansys academic teaching advanced
                    ' -dir "',directory,'"',...
                    ' -j "',jobname,'"',...
                    ' -s read',...
                    ' -m ',memory,...
                    ' -db ',reserve,...
                    ' -t -d win32',...
                    ' -i "',inputfile,'"',...
                    ' -o "',outputfile,'"'];

disp(callstring)
system(callstring,'-echo')

%% Update pressure field for next analysis
fid = fopen(resultsfile, 'rb');
fseek(fid, 0, 'eof');    % Set file position indicator at end of file
fileSize = ftell(fid);  % Get file position indicator = fileSize
frewind(fid);           % Set file position indicator at begin = fseek(fid,0,-1)
data = fread(fid, fileSize, 'uint8');
last_row = sum(data == 10) - 1;
lambdaS = csvread(resultsfile,last_row,16,[last_row 16 last_row 16]);
fclose(fid);
p = 1.2*lambdaS*p;

toc
```