Proceedings of the
Annual Stability Conference
Structural Stability Research Council
Louisville, Kentucky, April 1-4, 2025

Thin shell finite element formulations implemented in open-source software

Cristopher D. Moen[1], Sándor Ádány[2]

Abstract

The Mindlin shell finite element is implemented with classical linear shape functions, and also considering higher order shape functions, different numerical integration rules, and mesh densities. The shell elements are coded in MATLAB and in the Julia language, and examples considering membrane and bending deformations, as well as elastic buckling, are documented and compared to commercial finite element software. These studies highlight that it is challenging to write a generally applicable and accurate shell element because of well known shear locking and zero energy modes. The studies provide an opportunity to explore an open-source finite element software package in the Julia language, and also to lay out future plans for developing and validating an open-source shell finite element library. Computational runtimes for MATLAB and the Julia language implementations are also reported and discussed.

1. Introduction

Shell finite element analysis (FEA) has been used for academic research on thin-walled structures for decades to study elastic buckling and collapse behavior. In recent years, shell FEA has become more widely used by companies in the building construction industry, for example, steel-focused companies that are developing new products and systems, for example steel deck, joists, and purlins. Shell FEA is very useful for exploring structural behavior and providing rich results (full field stress, strains, forces) in ways that carry a heavy burden (time, cost) if implemented in a physical test.

For all of us, both academic and industry professionals, we are empowered by shell FEA software, but also constrained. Commercial finite element packages like Abaqus (2022) and ANSYS deserve credit because they can handle most of what we throw at them, and they have powerful user interfaces for pre and post processing results. They are costly to operate however, especially for large problems where multiple cores (and thus multiple license tokens) and required.

Another constraint for commercial shell FEA users is that the details of the shell finite element formulation, and the actual computer code, are not available for us to look at in commercial programs, and so we must resort to theory manuals and our own parameter studies to study element behavior

[1]President, RunToSolve, LLC, <cris.moen@runtosolve.com>
[2]Associate Research Scientist, Johns Hopkins University, <asandor2@jhu.edu>

and make decisions about element type and mesh density. Most of the time the bread-and-butter elements like the Abaqus S4R behave well, however there are cases where we could run analyses faster and more accurately if we had a 'better' shell finite element formulation and implementation. These kinds of advancements are not really available to us in commercial FEA software.

An alternative to commercial finite element software is academic software, often developed in research groups and sometimes released as open-source software. A popular example of FEA open-source software used widely in thin-walled structures is elastic buckling finite strip analysis in CUFSM (Schafer et al. 2025). For 3D frame structural analysis there is Frame3DD (Gavin 2023), and for earthquake engineering analysis and simulation there is OpenSees (McKenna 2011). What these software packages have in common is passionate academic leadership supported by industry and government collaborations.

The work documented in this paper is part of an effort to gather support and momentum for open-source shell FEA software in our professional community. The long-term plan is to develop an FE implementation that offers options for various element types, such as trilateral and quadrilateral with various number of nodal points, based on Kirchhoff and Mindlin plate theory, and applicable to a range of materials. As a starting point, this paper presents a set of baseline shell membrane and bending solutions solved using different implementations of a common four-node shell element formulation, with results compared to classical solutions and the commercial finite element software Abaqus. Additionally, we explore a general open-source FEA framework called `Ferrite.jl`, written in the Julia scientific and engineering computing language (Bezanson et al. 2012). as a potential home for validated shell element formulations. The shell element studies consider the 'textbook' four-node Mindlin shell element formulation introduced in the following section.

2. Element mechanics

Now, a four-node quadrilateral Mindlin element will be shown in some detail. The material is assumed to be isotropic and linearly elastic.

The involved displacement functions are $u(x, y, z)$, $v(x, y, z)$ and $w(x, y, z)$, interpreted as translations along x, y and z, see Fig. 1.

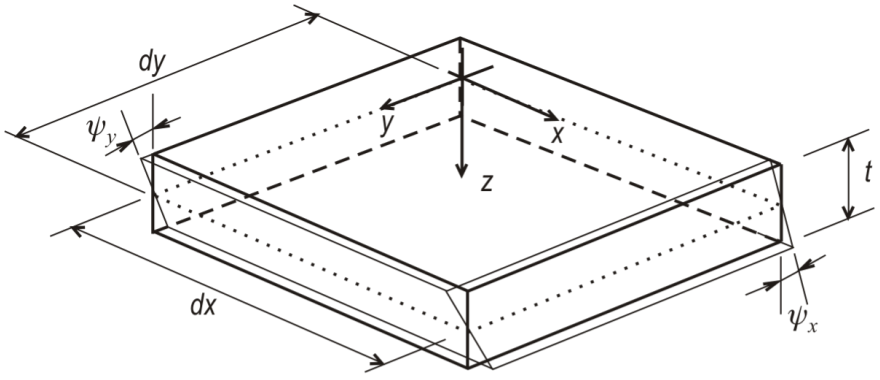

Figure 1: Elementary rectangle of a surface

The element mechanics is the superposition of membrane and bending stress/displacement fields, and w is supposed to be constant along the thickness, hence: $w(x, y, z) = w(x, y)$. Regarding u and v, they are linearly varying across the thickness, and thus, can be expressed as the sum of membrane strains (at $z = 0$) and bending strains (at $z \neq 0$).

$$\epsilon_x = \epsilon_{x,m} + \frac{d\psi_x}{dx} z \tag{1}$$

$$\epsilon_x = \epsilon_{y,m} + \frac{d\psi_y}{dx} z \tag{2}$$

$$\gamma_{xy} = \gamma_{xy,m} + (\frac{\partial \psi_x}{dx} + \frac{\partial \psi_y}{dy}) z \tag{3}$$

where ψ_x and ψ_y are cross-section rotations along x and y directions, respectively (i.e., about y and x axis, respectively).

The membrane displacement functions are $u(x, y)$ and $v(x, y)$ interpreted at $z = 0$. The linear membrane strains are obtained by the derivatives of $u(x, y)$ and $v(x, y)$:

$${\epsilon_{x,m}}^L = \frac{\partial u}{\partial x} \tag{4}$$

$${\epsilon_{y,m}}^L = \frac{\partial v}{\partial y} \tag{5}$$

$${\gamma_{xy,m}}^L = \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \tag{6}$$

The stresses:

$$\begin{bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{bmatrix} = \begin{bmatrix} \frac{E}{1-\nu^2} & \frac{E}{1-\nu^2} & 0 \\ \frac{\nu E}{1-\nu^2} & \frac{E}{1-\nu^2} & 0 \\ 0 & 0 & G \end{bmatrix} \begin{bmatrix} \epsilon_x \\ \epsilon_y \\ \epsilon_z \end{bmatrix} \tag{7}$$

where E and G are the Young's and shear modulus, respectively, and ν is the Poisson's ratio.

The linear strains from the plate bending are calculated by following Mindlin plate theory. Accordingly, in addition to ϵ_x, ϵ_y, and γ_{xy} there are transverse shear strains as follows:

$$\gamma_{xz} = \psi_x + \frac{dw}{dx} \tag{8}$$

$$\gamma_{yz} = \psi_y - \frac{dw}{dy} \tag{9}$$

which are assumed to be constant across the thickness. From the transverse shear strains, shear stresses are obtained as:

$$\begin{bmatrix}\tau_{xz}\\ \tau_{yz}\end{bmatrix} = \begin{bmatrix}\frac{5}{6}G & 0\\ 0 & \frac{5}{6}G\end{bmatrix}\begin{bmatrix}\gamma_{xz}\\ \gamma_{yz}\end{bmatrix} \tag{10}$$

where the 5/6 factor is to take into consideration the difference between the model and real through-thickness shear strain/stress distributions (constant in the model, but approx. quadratic in reality).

Nonlinear strains are calculated from membrane strains only, approximated by the Green-Lagrange tensor. Nonlinear strains are calculated at z=0 only, i.e., assuming constant nonlinear strains across the thickness. From $u(x,y)$, $v(x,y)$, and $w(x,y)$, they are expressed as follows:

$$\epsilon_{x,m}{}^{NL} = \frac{1}{2}\left[\left(\frac{du}{dx}\right)^2 + \left(\frac{dv}{dx}\right)^2 + \left(\frac{dw}{dx}\right)^2\right] \tag{11}$$

$$\epsilon_{y,m}{}^{NL} = \frac{1}{2}\left[\left(\frac{du}{dy}\right)^2 + \left(\frac{dv}{dy}\right)^2 + \left(\frac{dw}{dy}\right)^2\right] \tag{12}$$

$$\epsilon_{y,m}{}^{NL} = \left[\frac{du}{dx}\frac{du}{dy} + \frac{dv}{dx}\frac{dv}{dy} + \frac{dw}{dx}\frac{dw}{dy}\right] \tag{13}$$

3. MATLAB implementation

A four-noded rectangular (flat) shell element is developed in MATLAB. Two basic versions are developed, different in the assumed shape functions. In both versions the shape functions are identical for all the 5 displacement functions.

3.1. Four shape functions

The simplest set of shape functions is 4 linear shape functions. Expressed in a local coordinate system (see Fig. 2):

$$N_1 = \frac{1}{4}(xy - x - y + 1) \tag{14}$$

$$N_2 = \frac{1}{4}(xy - x - y + 1) \tag{15}$$

$$N_3 = \frac{1}{4}(xy + x + y + 1) \tag{16}$$

$$N_4 = \frac{1}{4}(-xy - x + y + 1) \tag{17}$$

Using these 4 shape functions, all the 5 displacement functions is expressed by 4 nodal displacements, hence, the resulted element has altogether 5×4=20 degrees of freedom (DOF). The nodal displacement are the u, v, and w translations and ψ_x and ψ_y rotations at each corner points.

Following the usual steps of finite element derivations, the k_e and k_g elastic and geometric stiffness matrices for the element can be expressed; they are 20×20.

3.2. Six shape functions

Since the finite element resulted from the 4 shape functions has various numerical issues, a possible improvement is to add two quadratic extra shape functions, (see Fig. 2):

$$N_5 = 1 - x^2 \tag{18}$$

$$N_6 = 1 - y^2 \tag{19}$$

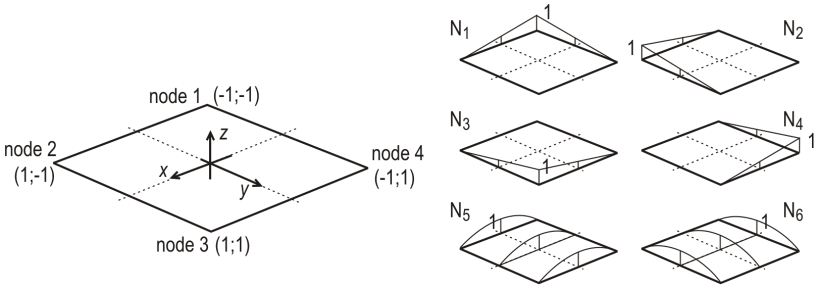

Figure 2: Local coordinates, shape functions

Using these 6 shape functions, all the 5 displacement functions is expressed by 4 nodal displacements, hence, the resulted element has altogether 5×6=30 degrees of freedom (DOF). The DOF are partially the same as above (i.e, u,v,w translations and ψ_x and ψ_y rotations at the corner points), but now there are two additional DOFs: physically these are the amplitudes of the quadratic functions at the middle of the element.

Following the usual steps of finite element derivations, the k_e and k_g elastic and geometric stiffness matrices for the element can be expressed; they are 30×30. However, it is impractical to have nodal DOFs at the middle of the elements, therefore, it is convenient to apply static condensation to eliminate the extra DOFs. In this way, the 30×30 stiffness matrix is transformed into 20×20, keeping only the nodal DOFs at the element's corner points.

3.3. Integration

To calculate the stiffness matrix, the final step is an integration. To obtain a specific element of the stiffness matrix, somewhat simplified and symbolically, the following operation is to be completed:

$$k = \iint f(x, y)\, dx\, dy \tag{20}$$

(Note, theoretically there is integration along the thickness, too, but it always straightforward to complete if the material is elastic, that is why the real challenge is the integral over the surface.) If the element is flat and rectangular, the above integral can be performed analytically, and the

stiffness matrix entries can be expressed in closed format. However, in more general cases the integration cannot be completed analytically, hence, numerical integration must be applied.

$$k = \sum_{i=1}^{n_G} W_i f(x_i, y_i) \tag{21}$$

where W_i are the weights, and n_G is the number of integration points. It is common to use the Gauss quadrature, this is what is applied in our implementation, too. If the element is quadri-lateral, 1, 2×2 or 3×3 integration points are to be applied, depending on the degree of the $f(x, y)$ function. For the actual finite element, 2×2 leads to exact result. However, it is a well-known practical trick to use fewer integration points than theoretically required; accordingly, we will consider two cases: n_G=1 and n_G=4.

4. Julia implementation

The `Ferrite.jl` implementation follows closely the four-node shell element mechanics and MATLAB approach discussed in the previous section, with 4 linear shape functions defined in Eq. 7. A `linear shell tutorial` in `Ferrite.jl` is available, and most of its algorithms are used to produce results for the examples at the end of this paper.

The `Ferrite.jl` software package has built-in capabilities for many of the FEA tasks that are needed to solve shell problems. The material properties, mesh density, and shell thickness are defined up front to calculate the constitutive matrices in Eq. (3) and Eq. (5). Then functions that relate deformation gradients to strains in Eq. (6) are defined. A grid of cells (elements) is generated based on a quadrilateral reference shape. The order of the shape functions is then defined, in our case the order is 1 (linear equations as shown in Eq. 7). Quadrature rules are then defined for the reference quadrilateral element. For this study, 1 integration point is used to take the integral over the element volume for both the in-plane membrane stiffness terms and the out-of-plane stiffness terms. (In the `Ferrite.jl` tutorial example there is also a through-thickness integration calculation that is not implemented for the examples in this study.) A `CellValues` object is used to define and organize the information about each element which facilitates the process of evaluating values of shape functions and gradients of shape functions. There is a `DofHandler` that assigns degree of freedom numbering for each cell, and then a `ConstraintHanlder` where boundary conditions are applied.

The formulation of the stiffness matrix k_e proceeds with iterations over every cell. The way that `Ferrite.jl` calculates the B matrix, i.e. the gradients of the shape functions at each degree of freedom, see Visy and Ádány (2017) Eq. 22, is unique compared to traditional FE approaches in that the derivatives of the strains are calculated numerically with automatic differentiation using the `ForwardDiff.jl` software package. The numerical integration to solve for k_e proceeds using the defined quadrature rules carried in `CellValues`, and the assembly of the global stiffness matrix and the global external force vector are performed with an `assemble!` function. The boundary conditions are applied with the built-in `dh!` function using static condensation, and the displacement field is solved using the Julia language's `A/b` linear equation solver.

The element geometric stiffness matrix k_g is calculated using automatic differentiation to solve for the deformation gradients (i.e., du/dx, dv/dx, ...) from the shape functions. The gradients are substituted into Eq. 11, 12, and 13 and multiplied by the element stresses $\sigma_{x,0}$, $\sigma_{y,0}$ and $\tau_{xy,0}$ to obtain three corresponding contributions to k_g, see Visy and Ádány (2017) Eq. 34. For a general treatment of elastic buckling problems, the reference stresses $\sigma_{x,0}$, $\sigma_{y,0}$ and $\tau_{xy,0}$ would be calculated first with a linear analysis. In the column buckling study later in the following example study, this was not necessary since the reference stress was assumed constant throughout.

Significant effort has been put in by the developers of `Ferrite.jl` to make its internal FE operations (stiffness matrix assembly and integration) computationally efficient. For example, the assembly of the stiffness matrix and force vector is accelerated by preallocated matrix sparsity patterns that are known a priori based on the element formulation, in this case a quadrilateral element with linear shape functions. FE solution performance for the MATLAB and `Ferrite.jl` implementations are explored along side of Abaqus in the following examples.

5. Examples

All examples presented here are for a 100 mm x 1000 mm plate with E = 200000 MPa and ν = 0.30. The plate boundary conditions are shown in Fig. 3. The mesh density is varied as follows: ([2, 2], [2, 8], [2, 32], [10, 92], [30, 322], [98, 980], [312, 3120]) where [number of elements in the 100 mm direction, number of elements in the 1000 mm direction]. The total number of degrees of freedom in each model are then [(2+1) x (2+1) x 5 = 45, 135, 495, 5115, 50065, 485595, 4884365].

The first case represents the possible coarsest discretization, with 2×2=4 elements and 3×3=9 nodes (hence 9×5=45 DOFs), which is certainly insufficient for practical use. The last case represents an extremely dense discretization, unnecessary in practice. Therefore, the selected discretizations cover the full range of possible discretizations, hence provide comprehensive information regarding the element behavior.

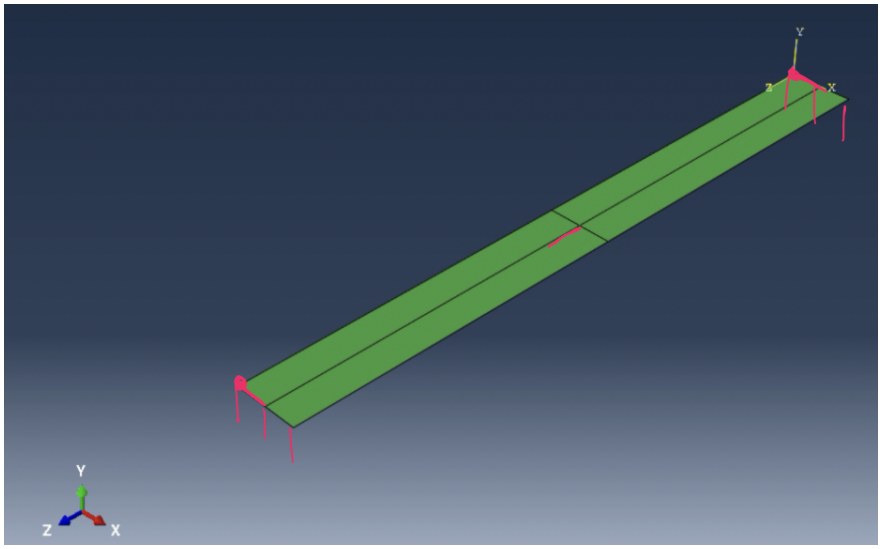

Figure 3: Plate translation is fixed at ends in global Y, at middle node in Z, and at two corner nodes in X

5.1. Out-of-plane bending with midspan distributed load

This example is essentially a beam, simply supported at the two ends, subjected to a concentrated force at the middle. However, the problem is solved as a plate problem, the load acting perpendicularly to the plane of the shell elements. Therefore, the 'concentrated' force is applied as a line load, as shown, the intensity being 1000 N/mm and 1 N/mm for the thick and thin plate, respectively. Moreover, pinned support means that the short (100-mm-long) edges are simply supported. Due to the loading and supports, only the plate degrees of freedom are activated. The maximum (Y-directional) displacements at the middle cross-section are summarized in Table 1.

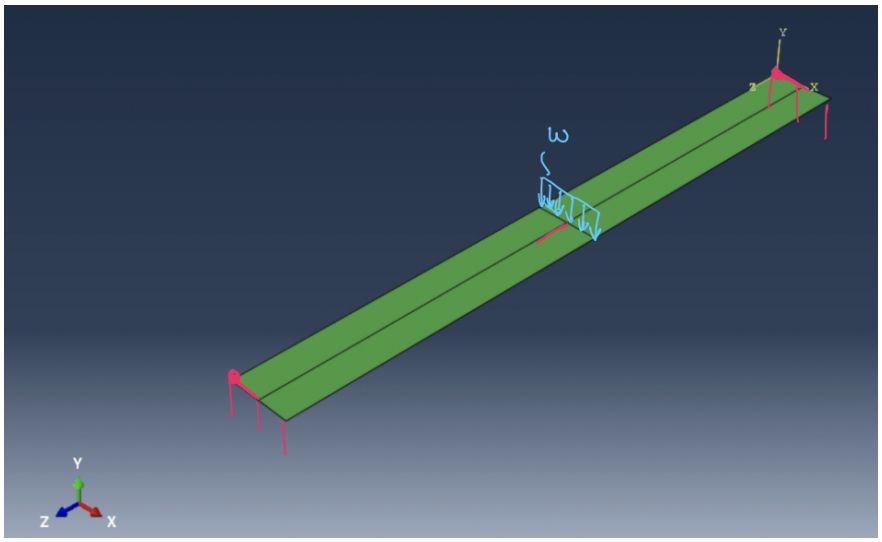

Figure 4: Out-of-plane midspan distributed load w

Since the problem solved is as a plate problem, the displacements along the middle transverse line are not perfectly constant, that is why both the maximum displacement (which occurs at the edge) and the average displacement (i.e., average of the nodal displacements along the line of the middle cross-section) are presented. Due to the simplicity of the problem, the deflection of the plate can be solved analytically, and in this case the solution is 1.289 mm and 156.252 mm for the thick and thin plate, respectively. (Note, the analytical solution without considering the shear deformations, i.e., using classic Euler-Bernoulli beam theory are 1.25 and 156.25 mm, respectively; thus, the effect of shear deformations is negligible if the plate is thin, while observable - though small - if the plate is thick.)

The main observations are as follows. (a) If the discretization is fine enough, all the variants seem to tend to the analytical solution. (b) If the theoretically necessary 2×2 quadrature is used for the integration, the shear stiffness is overestimated. This is especially problematic for thin plates, where the solutions are plain wrong unless the discretization is unrealistically dense. (c) As known from finite element textbooks, if reduced integration (in this case: 1-point quadrature) is used, the shear locking mostly disappears. (d) The `Ferrite.jl` and equivalent MATLAB implementation lead to virtually identical results. (e) The element with 4+2+2 shape function gives practically precise displacements regardless of the discretization, i.e., even if 2×2 elements are used. (f) The Abaqus results are quite good, too, with the exception of the 2×2 discretization.

			MATLAB						Ferrite.jl		ABAQUS	
		Shape functions	4+2+2		4		4		4		S4R	
		Quadrature rule	2x2		2×2		1		1			
	Mesh	Num. DOF	average	max	average	max	average	max	average	max	average	max
t = 100 mm	[2, 2]	45	1.29745	1.29926	0.14297	0.14311	0.97744	0.97931	0.97744	0.97931	0.97615	0.97792
	[2, 8]	135	1.29960	1.30150	0.85496	0.85601	1.27041	1.27228	1.27041	1.27228	1.26963	1.27140
	[2, 32]	495	1.29980	1.30168	1.24248	1.24407	1.28872	1.29059	1.28872	1.29059	1.28818	1.29000
	[10, 92]	5115	1.28949	1.29273	1.28386	1.28706	1.28896	1.29218	1.28897	1.29219	1.28852	1.29175
	[30, 322]	50065	1.28893	1.29240	1.28845	1.29191	1.28888	1.29234	1.28889	1.29235	1.28845	1.29191
	[98, 980]	485595	1.28881	1.29236	1.28876	1.29231	1.28881	1.29235	1.28882	1.29237	1.28838	1.29193
	[312, 3120]	4884365	1.28879	1.29236	1.28878	1.29235	1.28878	1.29236	1.28879	1.29237	1.28836	1.29193
t = 2 mm	[2, 2]	45	153.653	153.654	0.008	0.008	117.307	117.541	117.307	117.541	116.936	117.162
	[2, 8]	135	152.806	152.832	0.125	0.125	153.928	154.162	153.928	154.162	153.685	153.904
	[2, 32]	495	155.565	155.737	1.969	1.969	156.217	156.451	156.217	156.451	155.994	156.214
	[10, 92]	5115	156.136	156.525	14.806	14.810	156.083	156.473	156.114	156.505	156.026	156.414
	[30, 322]	50065	156.080	156.498	85.977	86.106	156.073	156.490	156.103	156.523	156.018	156.435
	[98, 980]	485595	156.065	156.493	143.160	143.525	156.064	156.492	156.094	156.525	156.050	156.478
	[312, 3120]	4884365	156.071	156.502	154.680	155.103	156.061	156.492	156.092	156.525	156.043	156.473

Table 1: FEA deflections at midspan in Y [mm]: out-of-plane bending with midspan distributed load

5.2. Out-of-plane bending with uniform pressure

This example is similar to the previous one, the only difference being the load, which is now a uniformly distributed load over the whole surface. (The beam equivalent of the problem would be a simply supported beam with a uniformly distributed line load over its full length.) The load intensities are 1 N/mm^2 and 0.001 N/mm^2 for the thick and thin case, respectively.

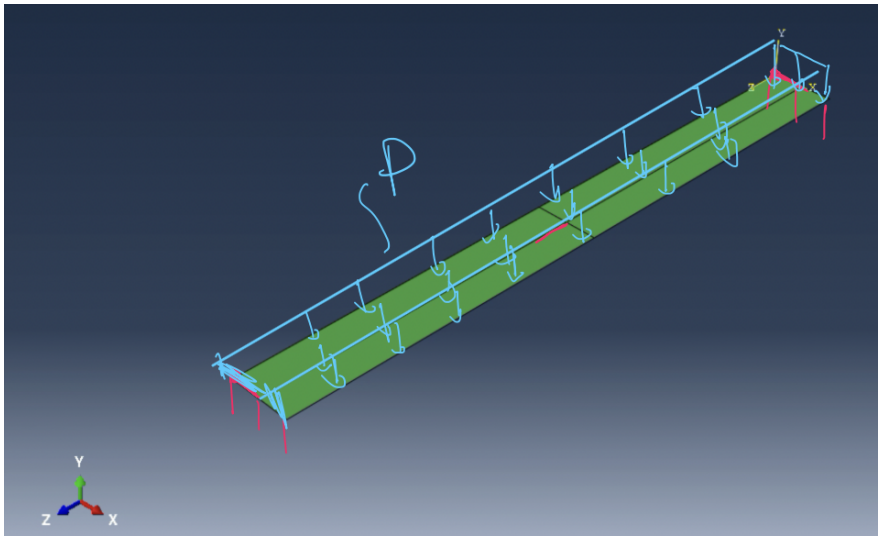

Figure 5: Out-of-plane uniform pressure p

The results are summarized in Table 2. The analytical solutions are 0.80075 and 97.657 mm for the thick and thin case, respectively. The observations are very similar to those mentioned above, hence, not repeated here. The only notable difference is that in this case even the 4+2+2 shape function element leads to imprecise results if 2×2 discretization scheme is employed.

			MATLAB						Ferrite.jl		ABAQUS	
		Shape functions	4+2+2		4		4		4		S4R	
		Quadrature rule	2x2		2×2		1		1			
	Mesh	Num. DOF	average	max	average	max	average	max	average	max	average	max
t = 100 mm	[2, 2]	45	1.29745	1.29926	0.14297	0.14311	0.97744	0.97931	0.97744	0.97931	0.97615	0.97792
	[2, 8]	135	1.29960	1.30150	0.85496	0.85601	1.27041	1.27228	1.27041	1.27228	1.26963	1.27140
	[2, 32]	495	1.29980	1.30168	1.24248	1.24407	1.28872	1.29059	1.28872	1.29059	1.28818	1.29000
	[10, 92]	5115	1.28949	1.29273	1.28386	1.28706	1.28896	1.29218	1.28897	1.29219	1.28852	1.29175
	[30, 322]	50065	1.28893	1.29240	1.28845	1.29191	1.28888	1.29234	1.28889	1.29235	1.28845	1.29191
	[98, 980]	485595	1.28881	1.29236	1.28876	1.29231	1.28881	1.29235	1.28882	1.29237	1.28838	1.29193
	[312, 3120]	4884365	1.28879	1.29236	1.28878	1.29235	1.28878	1.29236	1.28879	1.29237	1.28836	1.29193
t = 2 mm	[2, 2]	45	153.653	153.654	0.008	0.008	117.307	117.541	117.307	117.541	116.936	117.162
	[2, 8]	135	152.806	152.832	0.125	0.125	153.928	154.162	153.928	154.162	153.685	153.904
	[2, 32]	495	155.565	155.737	1.969	1.969	156.217	156.451	156.217	156.451	155.994	156.214
	[10, 92]	5115	156.136	156.525	14.806	14.810	156.083	156.473	156.114	156.505	156.026	156.414
	[30, 322]	50065	156.080	156.498	85.977	86.106	156.073	156.490	156.103	156.523	156.018	156.435
	[98, 980]	485595	156.065	156.493	143.160	143.525	156.064	156.492	156.094	156.525	156.050	156.478
	[312, 3120]	4884365	156.071	156.502	154.680	155.103	156.061	156.492	156.092	156.525	156.043	156.473

Table 2: FEA deflections at midspan in Y [mm]: out-of-plane bending with uniform pressure

5.3. In-plane bending with midspan distributed load

This example is a beam, similar to the one presented in Section 5.1, however, now the load is acting in the plane of the shell elements. Accordingly, now the membrane degrees of freedoms (and only those) are activated. The load intensities are 1000 N/mm for both thickness values. The analytical solutions are 1.289 mm and 64.45 mm for the thick and thin cases, respectively.

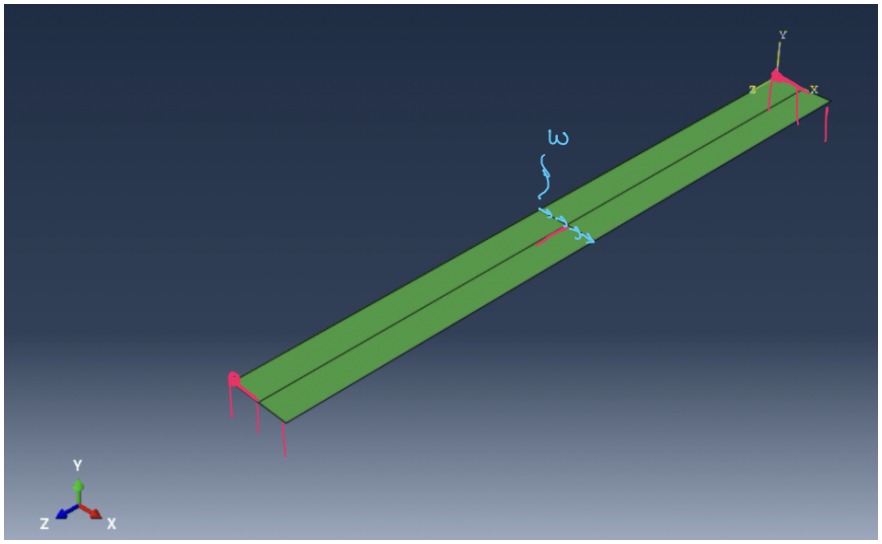

Figure 6: In-plane midspan distributed load w

The various FEM solutions are summarized in Table 3. Observations are as follows. (a) The plate thickness has no effect on the tendencies. (b) The 1-point integration does not work at all. (c) The 2×2 quadrature works well, with the exception of the 2×2 element discretization. (d) The 4+2+2 shape function yields the best results, but only slightly superior to the 4 shape function. (e) The Abaqus results are scattered around the exact solution, depending on the discretization; stabilized only if the discretization is relatively dense. (f) As the discretization is increased, all the variants

seem to tend to a displacement value (approx. 66.0 mm) slightly above the analytical solution (64.45 mm).

			MATLAB						Ferrite.jl		ABAQUS	
		Shape functions	4+2+2		4		4		4		S4R	
		Quadrature rule	2x2		2×2		1		1			
	Mesh	Num. DOF	average	max	average	max	average	max	average	max	average	max
t = 100 mm	[2, 2]	45	0.97299	0.97478	0.12143	0.12159	-2.08E+12	-2.59E+11			0.99822	1.20358
	[2, 8]	135	1.26350	1.26699	0.79059	0.79273	1.28E+12	3.49E+12			1.54333	1.56301
	[2, 32]	495	1.28688	1.29056	1.21229	1.21573	-2.22E+11	1.16E+00			1.70142	1.70637
	[10, 92]	5115	1.29977	1.30177	1.29186	1.29386	1.55E+11	3.55E+11			1.31992	1.32193
	[30, 322]	50065	1.30692	1.30867	1.30519	1.30694	8.63E+09	1.79E+10			1.31608	1.31783
	[98, 980]	485595	1.31306	1.31473	1.31194	1.31361	-9.72E+09	1.05E+09			1.32121	1.32288
	[312, 3120]	4884365	1.31929	1.32093	1.31822	1.31987	-2.77E+08	1.15E+00			1.32732	1.32896
t = 2 mm	[2, 2]	45	48.6495	48.7390	48.6495	48.7390	-2.08E+12	-2.59E+11			78.3104	93.1142
	[2, 8]	135	63.1749	63.3494	63.1749	63.3494	1.28E+12	3.49E+12			87.7029	91.5698
	[2, 32]	495	64.3438	64.5279	64.3438	64.5279	-2.22E+11	1.16E+00			86.6859	87.4398
	[10, 92]	5115	64.9885	65.0886	64.9885	65.0886	1.55E+11	3.55E+11			66.1807	66.2816
	[30, 322]	50065	65.3461	65.4336	65.3461	65.4336	8.63E+09	1.79E+10			65.8330	65.9205
	[98, 980]	485595	65.6531	65.7365	65.6531	65.7365	-9.72E+09	1.05E+09			66.0642	66.1476
	[312, 3120]	4884365	65.9644	66.0466	65.9644	66.0466	-2.77E+08	1.15E+00			66.3669	66.4491

Table 3: FEA deflections at midspan in X [mm]: in-plane bending with midspan distributed load

5.4. In-plane bending with uniform pressure

This example is similar to the one in Section 5.2, however, now the load is acting in the plane of the shell elements. Accordingly, now the membrane degrees of freedoms (and only those) are activated. The load intensities are 1 N/mm^2 for both thickness values. The analytical solutions are 0.80075 mm and 40.0375 mm for the thick and thin cases, respectively. The various FEM solutions are summarized in Table 4. The observations are very similar to those listed in Section 5.3.

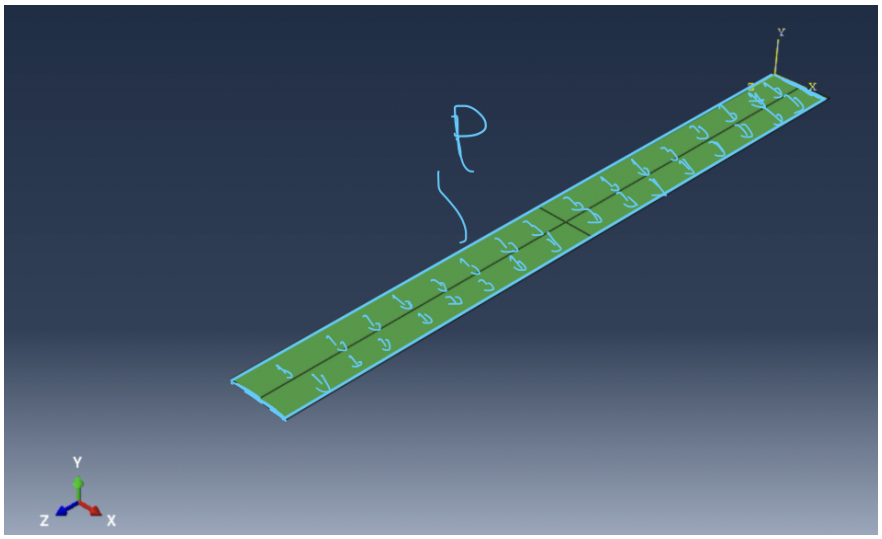

Figure 7: In-plane uniform pressure p

			MATLAB						Ferrite.jl		ABAQUS	
		Shape functions	4+2+2		4		4		4		S4R	
		Quadrature rule	2x2		2×2		1		1			
	Mesh	Num. DOF	average	max	average	max	average	max	average	max	average	max
t = 100 mm	[2, 2]	45	0.48752	0.48837	0.06108	0.06129	-2.08E+12	-2.59E+11			0.58339	0.77020
	[2, 8]	135	0.77946	0.78132	0.48649	0.48763	6.94E+11	1.75E+12			0.95729	0.97287
	[2, 32]	495	0.80237	0.80424	0.75526	0.75702	-2.32E+11	5.24E-01			1.06531	1.06781
	[10, 92]	5115	0.81260	0.81371	0.80730	0.80841	2.13E+11	4.95E+11			0.82820	0.82932
	[30, 322]	50065	0.81952	0.82051	0.81802	0.81901	1.50E+10	3.20E+10			0.82823	0.82922
	[98, 980]	485595	0.82561	0.82656	0.82452	0.82547	-9.58E+09	1.01E+09			0.83378	0.83473
	[312, 3120]	4884365	0.83183	0.83276	0.83077	0.83170	-1.07E+09	4.70E+06			0.83973	0.84067
t = 2 mm	[2, 2]	45	24.3760	24.4187	24.3760	24.4187	-2.08E+12	-2.59E+11			45.5283	59.2935
	[2, 8]	135	38.9729	39.0661	38.9729	39.0661	6.94E+11	1.75E+12			55.5813	58.8758
	[2, 32]	495	40.1183	40.2120	40.1183	40.2120	-2.32E+11	5.24E-01			54.6624	55.2761
	[10, 92]	5115	40.6299	40.6857	40.6299	40.6857	2.13E+11	4.95E+11			41.5810	41.6378
	[30, 322]	50065	40.9763	41.0258	40.9763	41.0258	1.50E+10	3.20E+10			41.4398	41.4893
	[98, 980]	485595	41.2808	41.3281	41.2808	41.3281	-9.58E+09	1.01E+09			41.6923	41.7397
	[312, 3120]	4884365	41.5915	41.6382	41.5915	41.6382	-1.07E+09	4.70E+06			41.9874	42.0341

Table 4: FEA deflections at midspan in X [mm]: in-plane bending with uniform pressure

5.5. Column buckling

All the previous examples involved linear static analysis. In the example presented here linear buckling analysis is performed. The load is a unit compressive stress, acting in the plane of the shell elements, at the short edges of the member, defined as an edge load. In both cases the first mode is a flexural buckling, with one half-wave longitudinally, i.e., the situation is essentially identical to the classic Euler column buckling problem. The first (i.e. lowest) critical stress calculated analytically is 1603.78 and 0.65797 N/mm^2 for the thick and thin member, respectively. Note, in these values the effect of shear deformation is included, that is why the values are somewhat lower compared to the classic Euler-formula prediction.

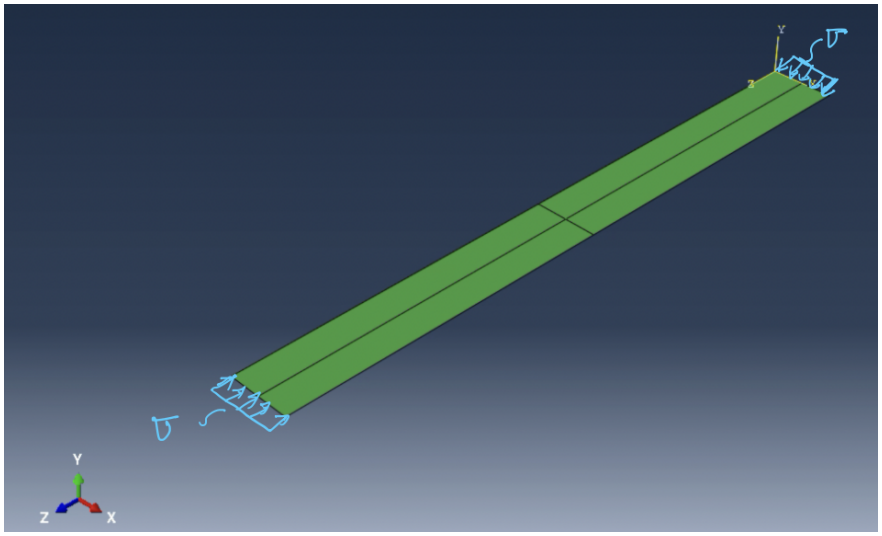

Figure 8: Uniform stress σ at plate ends

The FEM results are summarized in Table 5. Observations are as follows. (a) 1-point integration does not work, because zero-energy deformation of the elements is possible, that is why the

calculated critical load value can be zero. (b) 2×2 integration overpredicts the shear rigidity, consequently the critical values are very high unless the discretization is extremely fine. (c) When 4+2+2 shape functions are employed, the results are quite good, though if the discretization is very coarse, the solution is approximate. (d) Abaqus results are rather imprecise if the discretization is too coarse; but if the discretization is reasonable, the results are reasonable.

			MATLAB						Ferrite.jl		ABAQUS	
		Shape functions	4+2+2		4		4		4		S4R	
		Quadrature rule	2x2		2×2		1		1			
	Mesh	Num. DOF	mode 1	mode 2	mode 1	mode 2	mode 1	model 2	mode 1	mode 2	mode 1	mode 2
t = 100 mm	[2, 2]	45	1928.194	2545.601	17495.284	20581.134	4.73E-10	4.73E-10			2562.600	3356.500
	[2, 8]	135	1613.832	1639.859	2449.647	2630.517	6.95E-12	1.13E-10			1373.300	1645.200
	[2, 32]	495	1594.902	1600.473	1666.176	1698.056	2.21E-12	4.52E-12			1232.100	1606.400
	[10, 92]	5115	1593.100	1603.755	1601.780	1610.708	2.41E-13	5.03E-12			1603.600	1604.300
	[30, 322]	50065	1592.529	1603.981	1593.286	1604.574	2.14E-12	1.73E-11			1604.000	1616.400
	[98, 980]	485595	1592.472	1604.011	1592.551	1604.074	1.71E-12	6.08E-02			1604.000	1617.900
t = 2 mm	[2, 2]	45	0.813526	2545.601	16026.30	16126.25	9.27E-10	1.10E-09			1.070	167.000
	[2, 8]	135	0.679437	2.934890	835.14763(	956.957467	1.64E-13	1.94E-11			0.676	2.940
	[2, 32]	495	0.661846	2.664857	52.250619	175.170801	4.12E-12	7.16E-12			0.660	2.658
	[10, 92]	5115	0.658610	2.641582	6.944807	27.786535	8.98E-13	6.59E-01			0.659	2.645
	[30, 322]	50065	0.658660	2.642407	1.195807	4.785770	2.38E-12	1.34E-11			0.659	2.643
	[98, 980]	485595	0.658673	2.642633	0.718071	2.879209	2.76E-12	8.67E-12			0.659	2.643

Table 5: FEA buckling stress at column ends [N/mm^2]

6. Discussion

The results herein highlight how shape functions, numerical integration techniques, and mesh density influence shell FEA solution quality. It becomes clear that a generally reliable shell element formulation is not that easy to implement. The trends motivate us to continue this work, including the consideration of popular Mixed Interpolation of Tensorial Components (MITC) formulations, e.g., Dvorkin and Bath (1984) and Ko et al. (2017).

There is also the question of computational performance. It is difficult to provide apples-to-apples runtime comparisons of the FEA software considered in this study. Let us consider just the linear analysis (LA) examples (Sections 5.1 to 5.4, excluding the 5.5 elastic buckling). We know the MATLAB and `Ferrite.jl` calculations are working similarly, looping over each element (even though in the examples presented all the element stiffness matrices are the same), and not calculating stresses along the way. The linear equation solvers are also probably similar between MATLAB and `Ferrite.jl`. The solutions are run on roughly similar hardware (Intel I5 processor with 32GB RAM for MATLAB, Apple M3 Max with 36 GB RAM for `Ferrite.jl`). Considering the highest mesh density linear analysis solution (4,884,365 dof), the MATLAB solution time is 180 seconds and the `Ferrite.jl` solution time is 125 seconds (101255279 allocations: 85.01 GiB, provided by `@btime` from `BenchmarkTools.jl`. The ABAQUS wall time for the same analysis was roughly 7200 seconds (120 minutes), however again, ABAQUS is writing data to a database (stresses, displacements), and the solution controls were set so that a step size of typically 14 steps was observed. Another Abaqus benchmark was performed using 1 step and stripping out most of the database demands, resulting in a wall time of about 600 seconds, with deflection results that were different by about 5 percent when compared to the analysis with 14 steps.

This study also gives us motivation and energy to refine these shell element formulations and organize them in an open-source library. The experience with `Ferrite.jl` was overall positive, and a Julia package called `ShellFEA.jl` has been created to serve as a home for Julia shell element formulations as they are coded and validated. (The code for the examples in this study also live there.)

7. Conclusion

A shell finite element formulation study was conducted in MATLAB and the Julia language, considering membrane and bending examples that were compared to theory and to the commercial finite element software ABAQUS. The 'textbook' four-node quadrilateral and more advanced treatments with higher order shape functions, along variations in numerical integration quadrature points, highlighted the usefulness of reduced integration when considering membrane stiffness, and the potential pitfalls for buckling problems if not enough quadrature points were used. Ongoing and planned work is outlined with a long-term goal of coding and validating open-source shell FEA elements that will be organized in an open-source software library set up for high-performance computing.

References

Bezanson, J., Karpinski, S., Shah, V.B., and Edelman, A. (2012). "Julia: A fast dynamic language for technical computing". *arXiv preprint arXiv:1209.5145*.

Dassault Systemes (2022). *Abaqus*. Version 2022. URL: https://www.3ds.com/products/simulia/abaqus.

Dvorkin, E.N. and Bathe, K.-J. (1984). "A continuum mechanics based four-node shell element for general non-linear analysis". *Engineering computations* 1(1), 77–88.

Gavin, H. and Pye, J. (2023). *Frame3DD: A Structural Frame Analysis Program*. Version r591. URL: http://www.sourceforge.net/projects/frame3dd.

Ko, Y., Lee, P.-S., and Bathe, K.-J. (2017). "A new MITC4+ shell element". *Computers & Structures* 182, 404–418.

McKenna, F. (2011). "OpenSees: a framework for earthquake engineering simulation". *Computing in Science & Engineering* 13(4), 58–66.

Schafer, B.W., Ádány, S., Li, Z., and Jin, S. (Jan. 2025). *CUFSM*. Version 5.5. DOI: 10.xxxx/zenodo.xxxxx. URL: https://github.com/thinwalled/cufsm-git.

Visy, D. and Ádány, S. (2017). "Local elastic and geometric stiffness matrices for the shell element applied in cFEM". *Periodica Polytechnica Civil Engineering* 61(3), 569–580.