Proceedings of the
Annual Stability Conference
Structural Stability Research Council
San Antonio, Texas, March 19-22, 2024

A fast, scalable shell finite element formulation implemented with open-source software

Cristopher D. Moen[1]

Abstract

A shell finite element formulation framework is described as part of a new open-source software package that aims to provide both easy of use, computational efficiency, and scalability to large structural stability problems. The element formulation framework is generally posed to accept details into a software object that include: the nodal degrees of freedom; the element deformation function which approximates the element deflected shape; a mapping operator that defines the relationship between the deformation function and the nodal degrees of freedom; structural response functions such as ODEs or energy functions that define the structural behavior; and an equilibrium objective function. A unique part of the formulation is that functions and operators are defined symbolically and used numerically which allows for convenient control over the mechanics considered in the shell element. A formulation example is provided for a classical 4 node shell element. Next steps are discussed that include the consideration of element-to-element deformation compatibility, boundary conditions, loads, and the solution of deformation field as an optimization problem.

1. Introduction

Shell element formulations in finite element analysis software used for structural stability calculations are typically derived with the same flow: stiffness matrices mapping nodal discretization response to element behavior are derived by taking derivatives of assumed element shape functions and substituting them into equilibrium equations. The shape functions and equilibrium equations are 'baked in', and the element nodal deformation results are obtained either by inversion of the stiffness matrix or solving for the deformation vector from a system of equations. What if we could work another way though; a way that might help position us for computational advantages while conveniently managing the specific mechanics considered, e.g., bending, membrane, shear, geometric nonlinearities, and large deflections.

There is a whole new world out there of powerful open-source software supporting scientific machine learning and data-driven models. These software packages solve large systems of nonlinear equations in a flash, perform automatic differentation to find gradients, and run multi-variate optimization to solve for neural network coefficients that minimize an error loss function. They can

[1]President, RunToSolve, LLC, <cris.moen@runtosolve.com>

numerical integrate. They can provide symbolic manipulations of higher order equations. They can tell you what function that will work best to match an objective function. And the solution workflows are easily parallelizable meaning they can be applied to very (computationally) large problems.

You might be picking up on something now. That these open-source tools are not only useful for scientific machine learning, they are helpful to us, the structural stability community. And one focus area that has strong potential for impact is shell finite element analysis (FEA). Shell FEA has been used to bring clarity to many complicated thin-walled structures and systems, from metal building frames to steel storage racks to cold-formed steel joists. However in practice, in design, in day-to-day engineering, we don't see that much shell FEA applied. It may be because of the additional modeling complexity compared to frame analysis (discretizing a W-shape as a line element or shell elements are different levels of effort, for example), the slower computational speeds compared to a hand calc or a frame analysis, high commercial software license costs, or the difficulty of connecting shell FEA to our typical workflows, our calculation reports, our spreadsheets.

It is well know though that the effort of going to shell FEA has significant potential benefits. We can accurately capture beneficial warping torsion stiffness, clarify the influence of shear center eccentricities for members with singly symmetric cross-sections, and directly capturing local-global buckling interaction.

So maybe we can make these benefits outweight the costs. We can try to make a different kind of shell FEA software package, that takes some of the heavy lifting off of the user, that is open-source (zero cost), that can be potentially as fast as a frame analysis (e.g., SAP 2000 or RISA), and one that has a reliable interface (API) that can be incorporated into company's workflow. What would a framework look like though? This is the objective of this paper, to introduce this new software package and to work through some of these details, starting with the computational shell element formulation.

2. Open-Source Software

The name of the new open-source software package is `InstantShell.jl` and it is written to focus on 3D thin-walled structures where buckling deformation is of interest. It is written in Julia, a scientific computing language developed at MIT known for its composability and computational speed, see Bezanos et al.(2012). Julia is popular in data science and machine learning communities. There are several package dependencies used from the Julia SciML organization at the core of `InstantShell.jl`, including `Symbolics.jl` for handling symbolic equations (taking derivatives for example), `Optimization.jl` used to solve for nodal displacements, and `HCubature.jl`for numerical integration. There are other open-source shell finite element software packages available, including Hale et al.(2018) that the author has gained inspiration from, and that the reader should consider as well.

The shell element formulation is presented in this paper. Coordinate transformations in 3D, boundary conditions, external loading, element meshing and compatibility, and visualization will be dealt with in future work. Some of the work has already been completed, see the `InstantShell.jl` GitHub repository.

3. Shell Element Formulation

A shell element is comprised of some nodes and their degrees of freedom that respond in a certain way together when external stresses (loads, tractions) or (imposed) displacements are applied. The element deformed shape is defined by an approximate function, e.g. a polynomial or a Fourier series. The element nodal deformations from external influences (stresses, loads) are solved by substituting the deformation function transformed by a mapping operator into equations relating deformation and response (think ODE or energy functions), and applying a physics-based equilibrium objective function, for example conservation of energy or the principle of stationary energy. The mapping operator is important because it defines the relationship between the deformation function coefficients and the nodal deformation for each degree of freedom.

The high-level description above identifies the primitives (bolded terms) that are needed to define a shell element formulation so that we start getting organized from a software perspective. The primitives can then be written as a software object:

```
struct Shell
  nodal_degrees_of_freedom::Vector{Num}
  element_deformation_function::Vector{Num}
  structural_response_equation::Vector{Num}
  node_deformation_mapping::Vector{Matrix{Num}}
  equilibrium_objective::Vector{Num}
end
```

Each of the primitives in the object have a type `Num` (special to Symbolics.jl) which allows for equations to be explicitly written symbolically and later solved numerically. (There are similar symbolic types and symbolic algebra packages in other languages, e.g., SymPy in Python.)

The primitives are defined as either a `Vector` or a `Matrix` to accommodate different kinds of shell elements, for example a 4 node element or a 9 node element with multiple deformation functions and response equations (e.g., bending, membrane, shear).

4. Example: 4 Node Shell Element

Consider the 4 node bending-deformable geometrically nonlinear shell element formulation defined by Kapur and Hartz (1966) and summarized by Haskell (1970). The element has a width of $2a$ and depth of $2b$, thickness t, elastic modulus E, and Poisson's ratio μ. The element has a local $x-y$ coordinate system centered in the element (e.g., x ranges from $-a$ to a, y ranges from $-b$ to b) with the z out-of-plane axis following the right-hand rule. There are 3 nodal deformation degrees of freedom per node 1, 2, 3, and 4:

$$r_N = \begin{bmatrix} \delta_{1z} \\ \theta_{1x} \\ \theta_{1y} \\ \delta_{2z} \\ \theta_{2x} \\ \theta_{2y} \\ \delta_{3z} \\ \theta_{3x} \\ \theta_{3y} \\ \delta_{4z} \\ \theta_{4x} \\ \theta_4 \end{bmatrix} \tag{1}$$

where δ is the deformation in the element local z direction (out-of-plane) and θ is the rotation about the specified local axis.

The deformation function for the element is a cubic polynomial:

$$w = a_1 + a_2 x + a_3 y + a_4 x^2 + a_5 xy + a_6 y^2 + a_7 x^3 + a_8 x^2 y + a_9 xy^2 + a_{10} y^3 + a_{11} x^3 y + a_{12} xy^3 \tag{2}$$

and the mapping operator A that relates nodal deformation degrees of freedom to the element deformation function, i.e., $r_N = A[a_1 \dots a_{12}]'$ is:

$$A = \begin{bmatrix} 1 & -a & -b & a^2 & ab & b^2 & -a^3 & -a^2 b & -b^2 a & -b^3 & a^3 b & b^3 a \\ 0 & 1 & 0 & -2a & -b & 0 & 3a^2 & 2ab & b^2 & 0 & -3a^2 b & -b^3 \\ 0 & 0 & 1 & 0 & -a & -2b & 0 & a^2 & 2ab & 3b^2 & -a^3 & -3b^2 a \\ 1 & -a & b & a^2 & -ab & b^2 & -a^3 & a^2 b & -b^2 a & b^3 & -a^3 b & -b^3 a \\ 0 & 1 & 0 & -2a & b & 0 & 3a^2 & -2ab & b^2 & 0 & 3a^2 b & b^3 \\ 0 & 0 & 1 & 0 & -a & 2b & 0 & a^2 & -2ab & 3b^2 & -a^3 & -3b^2 a \\ 1 & a & b & a^2 & ab & b^2 & a^3 & a^2 b & b^2 a & b^3 & a^3 b & b^3 a \\ 0 & 1 & 0 & 2a & b & 0 & 3a^2 & 2ab & b^2 & 0 & 3a^2 b & b^3 \\ 0 & 0 & 1 & 0 & a & 2b & 0 & a^2 & 2ab & 3b^2 & a^3 & 3b^2 a \\ 1 & a & -b & a^2 & -ab & b^2 & a^3 & -a^2 b & b^2 a & -b^3 & -a^3 b & -b^3 a \\ 0 & 1 & 0 & 2a & -b & 0 & 3a^2 & -2ab & b^2 & 0 & -3a^2 b & -b^3 \\ 0 & 0 & 1 & 0 & a & -2b & 0 & a^2 & -2ab & 3b^2 & a^3 & 3b^2 a \end{bmatrix} \tag{3}$$

The A operator was determined here with Symbolics.jl.

The first structural response equation is the element strain energy U, see Chajes (1974) Chapter 6:

$$U = \frac{E}{2(1-\mu^2)^2} \int_{-a}^{a} \int_{-b}^{b} \left[\left(\frac{\partial^2 w}{\partial x^2} \right)^2 + \left(\frac{\partial^2 w}{\partial y^2} \right)^2 + 2\mu \frac{\partial^2 w}{\partial x^2} \frac{\partial^2 w}{\partial y^2} + 2(1-\mu) \left(\frac{\partial^2 w}{\partial x \partial y} \right) \right] dx dy \tag{4}$$

The second response equation is the external work V resulting from applied shell edge stresses σ_x, σ_y, and τ_{xy}, see Timoshenko and Woinowsky-Krieger(1959):

$$V = \frac{t}{2} \int_{-a}^{a} \int_{-b}^{b} \left[\sigma_x \left(\frac{\partial w}{\partial x} \right)^2 + \sigma_y \left(\frac{\partial w}{\partial y} \right)^2 \tau_{xy} \left(\frac{\partial w}{\partial x \partial y} \right)^2 \right] dxdy \tag{5}$$

The shell element equilibrium equation is established based on the conservation of energy:

$$\Pi = U + V = 0 \tag{6}$$

These are the key pieces of the shell element formulation that are entered into the `Shell` object defined above.

5. Next Steps

From here, the next steps would be to define the shell domain for the structure to be studied, discretize the domain into elements, define boundary conditions, add initial geometric imperfections, apply external forces, assemble the system equations, and solve for the free nodal deformation degrees of freedom. There are a few ways to get to the solution. A traditional finite element approach can be followed, by deriving element local elastic and geometric stiffness matrices from the `Shell` object and the energy quantities U and V, assembling the global elastic and geometric stiffness matrices, and solving for the deformation field.

Another approach that is currently being implemented for `InstantShell.jl` is a potentially more computationally efficient, where element-to-element compatibility is considered as a vector operator and the deformation field is solved as an optimization problem using `Optimization.jl`. These efforts are still a work in progress, although small problems including plates with initial imperfections and elastic buckling solutions have been validated.

There is also much to be learned from others, for example from Vigliotti and Auricchio (2021) where automatic differentiation (AD) is applied to solve solid mechanics finite element problems with elegance. And it is being shown that open-source finite element modeling can be implemented to solve very large problems (like millions of degrees of freedom) when attention is paid to the software design, see Verdugo (2022).

6. Conclusion

A shell finite element formulation is presented as the first steps in the development of an open-source shell finite element analysis software package `InstantShell.jl`. The goals for this new software development are to provide a general interface for defining the mechanics of interest in the solution, to provide software that is accessible and easy to use, and to design the software to solve large problems with high-performance computing. The formulation primitives are defined as the nodal deformation degrees of freedom, the element deformation function, the element structural response equations, the mapping between the deformation function and the nodal degrees

of freedom, and the objective function that guides the solution of the finite element deformation field. An example 4 node shell element implementation is described, and next steps are outlined including efficient algorithms for defining element-to-element compatibility that circumvent the traditional global stiffness matrix solutions.

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*.

Chajes, A. (1974). *Principles of structural stability theory*. Waveland Press.

Hale, J.S., Brunetti, M., Bordas, S.P.A., and Maurini, C. (Oct. 2018). "Simple and extensible plate and shell finite element models through automatic code generation tools". *Computers & Structures* 209, 163–181. ISSN: 0045-7949. DOI: 10.1016/j.compstruc.2018.08.001. URL: http://www.sciencedirect.com/science/article/pii/S0045794918306126.

Haskell, W. (1970). "Geometric nonlinear analysis of thin plates by finite elements". Ph.D. thesis. University of Massuchusetts.

Kapur, K.K. and Hartz, B.J. (1966). "Stability of plates using the finite element method". *Journal of the Engineering Mechanics Division* 92(2), 177–195.

Timoshenko, S. and Woinowsky-Krieger, S. (1959). *Theory of Plates and Shells, Second Edition*. McGraw-Hill Book Company, Inc., New York.

Verdugo, F. and Badia, S. (July 2022). "The software design of Gridap: A Finite Element package based on the Julia JIT compiler". *Computer Physics Communications* 276, 108341. DOI: 10.1016/j.cpc.2022.108341. URL: https://doi.org/10.1016/j.cpc.2022.108341.

Vigliotti, A. and Auricchio, F. (2021). "Automatic differentiation for solid mechanics". *Archives of Computational Methods in Engineering* 28(3), 875–895.