A SPLINE-BASED SHAPE OPTIMIZATION WORKFLOW FOR COMPOSITE SHELLS USING METAHEURISTIC ALGORITHM AND CALFEM-PYTHON

Van-Binh Tran[1], Hoang-Anh Pham[2,3]*, Chau-Anh Mai[2], Minh-Tu Tran[2,3]

[1]Ha Tinh University, Ha Tinh, Vietnam
[2]Hanoi University of Civil Engineering, Vietnam
[3]Frontier Research Group of Mechanics of Advanced Materials and Structures (MAMS), Hanoi University of Civil Engineering, Vietnam

*Corresponding author: anhph2@huce.edu.vn

(Received: March 05, 2026; Revised: May 06, 2026; Accepted: June 05, 2026)

DOI: 10.31130/ud-jst.2026.24(7B).568E

Abstract - This paper presents a workflow for the optimization of the shape of composite shells, where the shell model is parameterized by Splines. To achieve this, Gmsh API is used to establish the shell geometry using Splines passing through control points, where the coordinates of these points serve as the design variables. A self-developed module 'geometry.py' is built to ease the modeling task. The CALFEM-Python toolbox, supplemented with the CS-DSG3-A planar triangular shell element, is used as a tool for analyzing shell behavior depending on the geometry from Gmsh. The optimal shell shape is automatically sought through an improved differential evolution optimization framework. The entire optimization workflow is applied to isotropic and laminated composite shell models and evaluated.

Key words - Composite shell; shape optimization; metaheuristics; CALFEM-Python; CS-DSG3-A

1. Introduction

Shell shape optimization is an important branch of structural optimization, focusing on determining the optimal mid-surface geometry to minimize design criteria such as material volume, deflection, stress, or to approach a pure membrane stress state. This problem is particularly complex because the objects of study are three-dimensional structures with complex geometries, including thin shells, cylindrical shells, conical shells, spherical shells, spatial roof shells, and hyperbolic paraboloid shells. When composite materials are combined with shell geometric representation, the problem becomes significantly richer in both design space and computational complexity. Laminated composite shells introduce an additional layer of design variables in parallel with geometric variables: material distribution and fiber orientation.

In early studies related to optimal shell shape design, the design variables were typically the nodal coordinates of the finite element model (FEM) (see, e.g., [1]). Such a choice has serious disadvantages, including: a large number of design variables, difficulties in handling geometric requirements for boundary regularity, and difficulties in maintaining a suitable finite element mesh during the optimization process. When nodal coordinates are treated as shape variables, the set of design variables is usually very large, increasing the cost and difficulty of the optimization process. On one hand, the total number of shape variables depends directly on mesh refinement. On the other hand, two or three design variables are needed to describe the movement of each node. Another drawback is the tendency to produce unrealistic designs. Curvature changes at element interfaces frequently occur in the first iterations. As a result, the optimization process proceeds very slowly, requires a large amount of computation time, and often converges to an undesirable or infeasible shape.

The trend of using Splines (Bezier and B-Spline) was subsequently developed, starting from the research of Braibant and Fleury in 1984 [2], which overcame the drawbacks of the nodal coordinate variable method. They demonstrated that using B-Spline control points as design variables provides significant benefits: a small number of variables, automatic smooth geometry, and easy gradient computation. This is the starting point for most subsequent research on shell shape optimization and is the foundation for the iso-geometric analysis and optimization method (IGA) [3], [4].

Globally, shell shape optimization has made significant progress in both theoretical research and industrial manufacturing. In Vietnam, although the field has attracted attention relatively early, publications remain limited to date. The majority of research on shell structures focuses on the analysis of mechanical behavior (bending [5], buckling, vibration) rather than shape optimization in the full sense. Some representative Vietnamese authors researching shell shape optimization include: Ho-Nguyen-Tan on shape and topology optimization of shell structures [6], Pham Toan Thang and Nguyen Thoi Trung on shape and material optimization of FGM toroidal shells [7], Hung-Nguyen Xuan on optimization of triply periodic minimal surface structures [8], and Quoc-Tinh Bui on free-form surfaces based on NURBS by iso-geometric analysis and particle swarm optimization [9]. A challenge for domestic research is access to computational tools. Shell structure computations with complex geometries primarily rely on commercial software (e.g., ANSYS, ABAQUS, Altair OptiStruct). The exploitation of free open-source tools remains very limited.

It can be said that composite shell shape optimization is the convergence of three parallel streams of development: composite material mechanics, Spline-based modeling and computation, and optimization algorithms. One of the key focuses of our research group is to develop an optimization framework that connects these three domains naturally, creating a design workflow in which

shell modeling, analysis and optimization are performed synchronously using open-source tools.

Unlike isogeometric analysis (IGA), which demands high implementation complexity, or existing commercial software with prohibitive costs, this workflow combines the meshing tool Gmsh [10], the CALFEM-Python analysis tool [11], and the dDEmRao metaheuristic optimization tool [12] on a unified Python platform, with custom-developed modules serving as a flexible bridge between the three tools. Several numerical examples are carried out to illustrate and evaluate the effectiveness of the proposed workflow.

2. Basics of Spline-based shell shape optimization

2.1. Optimization problem and design variables

Shell structures operate simultaneously via 'membrane' and 'bending' mechanisms. Membrane stiffness is proportional to the thickness t, while bending stiffness is proportional to t^3. This makes the shell surface geometry highly influential on performance: a properly curved shell can carry loads primarily through the membrane mechanism, avoiding local bending and significantly reducing maximum stresses.

Gaussian curvature $K = \kappa_1\kappa_2$ and mean curvature $H = (\kappa_1 + \kappa_2)/2$ are the two core geometric characteristics that determine the stress state. Shells with $K > 0$ (elliptic surface, e.g., spherical dome) typically carry compression; shells with $K < 0$ (hyperbolic surface, e.g., cooling tower) carry tension–compression combined in two principal directions. Shape optimization is essentially finding the mid-surface geometry $S(\xi^1, \xi^2)$, where ξ^1, ξ^2 are the two natural axes along the curved surface, that achieves an appropriate distribution of K for the applied loading such that only membrane stresses remain in the shell. A common choice for the objective function is the total strain energy or structural compliance. However, computing these quantities requires intervention in the FEM model, which is not compatible with the case where a 'black-box' analysis tool is used. In this study, the maximum bending stress resultant in the shell, M_{max}, is used as the optimization objective. The shell shape optimization problem can be expressed mathematically as:

$$\text{Find } S(\xi^1, \xi^2) \text{ to minimize } f = M_{max} \tag{1}$$

For modeling purposes, the shell geometric surface is represented by Spline/B-Spline curves. A Spline is a piecewise polynomial function that connects continuously to a certain degree. Typically, for complex geometries, the mid-surface of a shell is described via B-Spline or NURBS:

- B-Spline with basic functions $N_{i,p}(\xi)$ of degree p according to the Cox–de Boor formula:

$$S(\xi^1, \xi^2) = \sum_{ij} N_{i,p}(\xi^1)\, N_{j,q}(\xi^2) P_{ij} \tag{2}$$

- NURBS (Non-Uniform Rational B-Splines) extend B-Splines with weights w_{ij}, allowing exact representation of conic surfaces,

$$S(\xi^1, \xi^2) = \frac{\sum_{ij} w_{ij} N_{i,p}(\xi^1)\, N_{j,q}(\xi^2) P_{ij}}{\sum_{ij} w_{ij} N_{i,p}(\xi^1) N_{j,q}(\xi^2)} \tag{3}$$

In Eqs. (2) and (3), P_{ij} are the control points of the Splines. The coordinates of the control points P_{ij} (the component z_{ij} in this study) are chosen as design variables. The notable advantage is the small number of variables (a few dozen to a few hundred control points are sufficient to describe a smooth surface), and global smooth geometry guaranteeing C^{p-1} continuity. The optimization problem is rewritten as

$$\begin{aligned} &\text{Minimize } f = M_{max}(z_{ij}) \\ &z_{min} \le z_{ij} \le z_{max} \\ &g_{ij} \le g_{max} \end{aligned} \tag{4}$$

where, z_{min} and z_{max} are the lower and upper limits of z_{ij}; g_{ij} is the surface slope at point P_{ij}; and g_{max} is the allowable (physically meaningful) slope.

2.2. Modeling Tool – Gmsh

The modeling tool has the function of digitizing the shell shape, storing data about the points, edges, and surfaces of the shell, and even creating a computational model (e.g., generating a FEM mesh). There are currently quite a few tools that support shell modeling using Splines, including CAD tools with GUI and Toolboxes for programming. In this study, we use Gmsh.

Gmsh [10] is a three-dimensional finite element mesh generation program with an integrated CAD tool and post-processor. Gmsh provides a fast, lightweight, and user-friendly mesh generation tool with parametric input capability and flexible visualization. Gmsh can be controlled by a graphical user interface (GUI), from the command line, using text files written in Gmsh's own scripting language ('.geo' files), or through an application programming interface (API) in C++, C, Python, Julia, and Fortran.

A notable feature when using Gmsh is that Splines do not create meshes manually but rather serve to describe the boundaries and geometric surfaces of the shell. Gmsh then fully automates the generation of high-quality meshes. Gmsh handles complex geometry, controls mesh quality, and automatically numbers nodes and elements.

The benefits of integrating Gmsh in the optimization workflow lie at two key points. First, mesh quality is ensured automatically – Gmsh optimizes element angles, avoids distorted elements that manual meshing easily produces when the curved geometry is complex. Gmsh supports options to automatically refine elements in regions with high curvature, which is particularly valuable when the shell shape changes significantly between optimization generations. Second, through Physical Groups (PGs), boundary conditions and loads can be applied precisely to nodes even if the mesh topology changes each iteration. In Gmsh, PGs are collections of geometric entities (points, curves, and surfaces) assigned fixed integer labels. When Gmsh regenerates the mesh following each update of the design variables, the PG labels are preserved with respect to the geometric entities rather than to the nodes or elements. The function *extract_calfem_data()* (as described in Section 3) reads these labels to filter boundary nodes and surface markers.

Consequently, regardless of any complete alteration in the mesh topology - including changes in the number of nodes and elements - the boundary conditions and applied loads remain correctly mapped to their intended geometric locations.

2.3. FEM Analysis tool - CALFEM-Python

There are many tools for plate/shell computation including commercial software and open-source software. Commercial software (such as ANSYS, Abaqus, NASTRAN) is generally powerful with user-friendly interfaces and good technical support, but the cost is high and users have limited control in learning and research. Open-source software saves costs, and is suitable for learning and research. Some open-source software includes Code-Aster, Salome-Meca, FreeCAD, Elmer, Stabil, and CALFEM. Among them, CALFEM provides users with a complete, easy-to-use FEM computational toolkit with a transparent structure that allows control down to individual element matrices. CALFEM-Python [11], the Python-based version of CALFEM, is a free open-source computational environment suitable for a wide audience in learning and researching structural computation. CALFEM-Python itself also has the capability to create geometric shapes and automatic element meshes for plate/shell structures through integration with Gmsh software.

The procedure for performing FEM analysis using CALFEM-Python follows these steps:

1. Set up the FEM mesh through the data tables: *coords, edofs, dofs, bdofs, elemarkers*

coords – Contains information about node coordinates;

edofs – Contains element connectivity information by degrees of freedom at nodes;

dofs – Contains the list of node degrees of freedom;

bdofs – Contains degrees of freedom assigned by label;

elemarkers – Contains labels assigned to elements.

2. Build the stiffness matrix K and the nodal force vector f of the system: using the built-in functions in the *core.py* module.

3. Apply boundary conditions: declare variables *bc, bcVal*

4. Solve the equation system:

core.solveq(K, f, bc, bcVal)

5. Compute strains and internal forces in elements.

However, using CALFEM-Python for shell structure analysis would be difficult due to limitations in the element library (CALFEM-Python does not have a built-in 3D shell element). To overcome this limitation, we have developed a shell element and utility modules to support CALFEM-Python, in order to exploit this software effectively in learning and research on shell structure computation.

The authors have developed the modules *material.py, element.py, system.py,* and *postprocessing.py* to supplement the CALFEM-Python program. These modules provide functional tools for setting up the stiffness matrix, nodal force vector, and performing displacement and internal force analysis of shell structures. The core of the extension is the CS-DSG3-A shell element, a flat triangular element with a full 6 degrees of freedom at each node (including the drilling rotational degree of freedom). Details of this element are presented in [13]. The advantage of the CS-DSG3-A element is its high accuracy and low computational requirement (compared to higher-order elements such as 'shell6' or 'shell8'). Furthermore, CS-DSG3-A is formulated in general form to be suitable for laminated composite shells.

2.4. Optimization tool - dDEmRao

When continuous variables and available gradients are present, gradient-based algorithms such as MMA (Method of Moving Asymptotes) or SQP (Sequential Quadratic Programming) are effective choices. When the problem contains discrete variables and gradients are unavailable, metaheuristic methods such as GA, DE, and PSO are alternative choices.

The general procedure of a metaheuristic algorithm is as follows:

- Generate an initial set of designs (also called a population): each design is a set: $(v1, v2, ..., vN, fitness)^{old}$.

- Generate a new generation: update design variables according to the operators of the algorithm, compute the corresponding objective function value, obtaining new designs $(v1, v2, ..., vN, fitness)^{new}$.

- Selection: compare $(fitness)^{new}$ with $(fitness)^{old}$, if $(fitness)^{new}$ is better (i.e., smaller in a minimization problem), replace the old design $(v1, v2, ..., vN, fitness)^{old}$ with the new one $(v1, v2, ..., vN, fitness)^{new}$.

- Check convergence: verify the convergence condition or stopping criterion; if satisfied, stop the program and report the results.

The optimization process using metaheuristics typically involves many iterations and thousands of objective function evaluations. To reduce the computational burden, in this study the dDEmRao-DiC algorithm [12] is used in the numerical examples. The dDEmRao algorithm with the DiC technique allows the elimination of designs that are assessed as low-potential without computing the objective function, thereby reducing unnecessary computational effort. Details of the dDEmRao algorithm can be found in reference [12]. The algorithm code can be downloaded from the GitHub repository: https://github.com/HoangAnh-Pham/dDEmRao-DiC.

3. Workflow: Gmsh + Spline + CALFEM-Python + Metaheuristic

The overall structure of the workflow consists of five modules. Details of the modules are presented below.

Module 1. Define shell geometry, design variables:

The shape of the shell is defined by geometric objects: points, edges (lines), and surfaces. These objects are assigned names (tags) (by Gmsh) which will be used to declare the shell geometry structure. The declared variables are as follows:

– Variable containing the list of points:

points = [(x1, y1, z1), (x2, y2, z2, size_factor),...]

where *x, y, z* are the point coordinates; *size_factor* is the mesh size parameter.

– Variable containing the list of edges:

lines = [(p1, p2,...), (p3, p4, p5,...),...]

where *p1, p2,...* are point tags automatically assigned by Gmsh in the order of the points list.

– Variable containing the list of surfaces:

surfs = [(l1, l2,...), (l3, l4, l5,...),...]

where l1, l2,... are line tags assigned in the order from the lines list.

– Next, it is necessary to assign labels (markers) to objects. Labels are used to set up boundary conditions, material, thickness, and loading for the shell. Note that assigning labels to surface objects is mandatory (for nodes and edges it is optional).

markers = {marker1: (type, [obj1, obj2,...]),
marker2: (type, [obj3, obj4,...]),...}

where *marker1, marker2* are numerical labels; *obj1, obj2, obj3*,... are the tags of the labeled objects; *type* is the parameter indicating the object type (0 – point, 1 – line, 2 – surface).

– In the case where the shell surface passes through some internal control points (points not on an edge), these control points are declared additionally as:

constraints = {surf_tag: [p1, p2,...]}

– Finally, set up the mapping from design variables to node coordinates. This is the bridge between the metaheuristic and Gmsh. When design variables are changed, through this mapping variable, the coordinates of nodes will be updated:

connector = {variable_id: [(p1, dim), (p2, dim),...],
variable_id: [(p3, dim), (p4, dim),...]}

where *variable_id* is the index in the design variable list; *p1, p2,...* are point tags; *dim* is the coordinate axis parameter (1 – x, 2 – y, 3 – z).

Module 2. Generate mesh – Gmsh to CALFEM

This is the most important and intricate module for the shell shape optimization problem. In practice, to create element meshes for CALFEM, the following approaches can be used:

– Use Gmsh API directly: the user needs to be proficient in Python and Gmsh API.

– Use CALFEM: CALFEM uses the *Geometry* class to describe points, edges, and surfaces, then *GmshMeshGenerator* calls Gmsh to generate the mesh and returns *coords, edof, dofs, bdofs, elemarkers*. This approach is fairly simple, but has two drawbacks: 1) CALFEM Geometry (mainly supporting 2D) has some limitations in shape creation; 2) It does not process information directly from the Gmsh model: CALFEM creates an intermediate '.geo' file for Gmsh, then uses Gmsh API to create a '.msh' file, and reads this file to return *coords, edof, dofs, bdofs, elemarkers*, which takes more time.

– Use the *geometry.py* module developed in this research. This uses Gmsh API indirectly through a wrapper, with concise and simple commands that make it more accessible to users, while still retaining the full power of Gmsh API. Additionally, it can still be used in parallel with Gmsh API. The steps are as follows,

+ Load the geometry module from the 'shell' directory:

import shell.geometry

+ Declare the geometry model variable:

gmodel = shell.geometry.ShellSplineSurface(points, lines, surfs, markers, constraints, connector)

The above two steps are performed only once when the optimization program is initialized.

+ Create mesh in Gmsh:

shell.geometry.create_mesh(gmodel, show, filltype, el_size_factor)

+ Transfer Gmsh data to CALFEM:

Mesh = shell.geometry.extract_calfem_data(gmodel)

The returned mesh variable is a dictionary with the structure:

Mesh = {'coords': coords, 'edofs': edofs, 'dofs': dofs, 'bdofs': bdofs, 'elementmarkers': elemarkers }

Module 3. FEM analysis: CALFEM-Python

After obtaining the FEM mesh data *coords, edof, dofs, bdofs, elemarkers*, we proceed to declare the computational data for the shell including: material, loading, boundary conditions, etc., and perform structural analysis following the CALFEM-Python procedure.

With the support of the newly added functional modules, the shell computation procedure is carried out simply as presented below.

1. Set up the data variables for the shell configuration:

– Material: organized by group and layer as follows:

Mat = [mat1, mat2,...]

where *mat1, mat2* are lists containing information about material groups. Each material group can include one or more different material layers, organized as a tuple:

mat1 = [layer1, layer2,...]

where *layer1, layer2*,... are tuple variables with the structure:

(matfun, (para1, para2,...), angle, h)

where *matfun* is the material description function; *para1, para2,...* are parameters depending on the material type; angle is the fiber direction angle (for orthotropic materials); and h is the thickness of the material layer. For example, to declare an ISO material layer with elastic modulus *E*, Poisson's ratio *nu*, density *ro*, and thickness *h*:

layer1 = (shell.material.material_ISO, (E, nu, ro), 0, h)

– Shell thickness: organized by group as follows:

Thick = [h1, h2,...]

where *h1, h2,...* are the thickness parameters of the shell surfaces.

– Shell surface structure: specify the material group and thickness for the surfaces in a list:

Face = [(marker, thkID, matID),...]

where *marker* is the label assigned to the surface object; *thkID* is the thickness group index; *matID* is the material group index.

– Supports/boundary conditions: supports along points and edges through labels (markers) and constraint codes (*dim*) in the form:

Bound = [(marker, dim), (marker, dim),...]

where *dim* can take values 0, 1, 2, 3, 4, 5, 6.

– Loading: loads applied to points, edges, or surfaces are specified by marker:

Load = [(marker, (f1, f2, f3), obj_type)]

where *f1, f2, f3* are the load values applied along the 3 local axes 1, 2, 3 or global axes x, y, z; *obj type* is a string variable that can take the values: 'point', 'line', 'face', 'element'. When *obj_type* = 'point' or 'line', *f1, f2, f3* are the load values applied to the object in the global coordinate system, while when *obj_type* = 'face' or 'element' these are uniformly distributed surface loads ('face' – global coordinate system, 'element' – local coordinate system).

2. Initialize the computational model by creating the FEM model for computation with the command:

model = shell.system.shellmodel(prob_type='',
Bound=Bound, Face=Face, Thick=Thick,
Mat=Mat, Load=Load, Mesh=Mesh)

3. Execute the computation:

– Compute displacements:

d, r = shell.system.displacement(model)

– Compute internal forces:

f = shell.system.internalforce(model, d)

– Compute total shell volume:

V = shell.system.shellvolume(model)

4. Extract results:

– Nodal displacements: extracted by

node_disp = model['Disp']

which is an N×6 array containing the displacement components of nodes.

– Internal forces: force components are extracted by

ele_force = model['Force']

which is a dictionary containing internal force values at element centroid, including keys: 'Nxx', 'Nyy', 'Nxy', 'Mxx', 'Myy', 'Mxy', 'Qxz', 'Qyz'. For example, to retrieve the Myy component of all elements: *ele_force['Myy']*.

Module 4. Fitness function

Fitness functions can be programmed separately or integrated depending on the requirements of the algorithm and optimization method. In this study, we build a composite fitness function that will be called by the metaheuristic. The main components of the objective function include:

– Update design variable values to node coordinates through:

shell.geometry.assign_coord(gmodel, variables)

– Check geometric constraints (fast, no FEM required): if node coordinates fall outside the design domain, the design will be rejected.

– Check slope (impractical for manufacturing): if the shell slope in a design is too large, it will also be rejected.

– Generate mesh: call *shell.geometry.create_mesh* and *shell.geometry.extract_calfem_data* in Module 2.

– FEM analysis: call Module 3.

– Compute objective and constraint values: from the obtained shell behavior, compute the objective function value and constraint conditions. Compute the composite fitness value (using the penalty method).

Module 5. Metaheuristic: *dDEmRao-DiC*

This module is packaged in the program *dDEmRao.py*. The simple execution syntax is as follows:

Opt = dDEmRao(fitness, nvars, Lb, Ub, Ng, Tol, NP, F, Cr, label)

where the inputs include:

fitness – objective function

nvars – number of design variables

Lb, Ub – arrays containing the lower and upper bounds of variables

Ng, Tol – the maximum number of iterations and convergence tolerance used to control the stopping condition

NP, F, Cr – population size, scaling factor, and crossover rate of DE, respectively

label – text label ('DiC' for executing DiC method)

The output *Opt* is a tuple containing the optimization results such as: optimal design variables, corresponding objective function value, number of objective function evaluations, etc.

4. Numerical examples

4.1. ISO cylindrical shell

To illustrate the proposed workflow, the shape optimization of an isotropic (ISO) cylindrical shell, taken from Ref. [9], is considered first. The shell has a thickness of 140 mm and covers an area of 8 × 12 m. The initial height of the shell is 4 m. The two long edges are simply supported while the two short edges are free. The elastic modulus of the material is 30 GPa and the Poisson's ratio is 0.3. The shell is subjected to a uniformly distributed surface load of 4 kN/m² in the vertical direction.

The shell geometry is generated by 4 B-Spline curves as shown in Figure 1a. The initial element mesh and control points are shown in Figure 1b, with 7 control points. The initial total compliance of the structure is 0.0232 kNm, corresponding to a strain energy of 11.58 J. The maximum bending moment in the shell is 1.202 kNm. The control points along the long edges (1, 2, 3, 4) are kept fixed, while points 5, 6, and 7 have variable elevations, yielding 3 optimization variables.

The optimization is carried out using the following

parameter settings for dDEmRao: Ng = 200, NP = 25, F = 0.8, Cr = 0.9, and Tol = 10^{-6}. The parameters are chosen based on a prior sensitivity study. The optimization process terminates when either (i) the maximum number of generations Ng is reached, or (ii) the variation in the objective function value between successive generations becomes smaller than Tol.

The optimization yields significant improvements across all criteria (Table 1): the maximum bending moment reduces by 77.79%, the maximum displacement decreases by 69.53%, the volume reduces by 12.41%, and the strain energy drops by 58.12%, demonstrating the effectiveness of the proposed approach. The resulting optimized shell shape is shown in Figure 2, which is consistent with the result reported in Ref. [8] with fixed support. However, the case of simply supported edges reported in [8] gives a different shape with optimized strain energy of 5.36 J, which is higher than that of the present results.

Table 1. *Optimal results of ISO cylindrical shell*

	Init.	Opt.	Impr.
M_{max} (kNm)	1.202	0.267	77.79%
Max. disp. (m)	1.83e-4	5.64e-5	69.53%
Volume (m^3)	19.83	17.37	12.41%
Strain energy (J)	11.58	4.85	58.12%

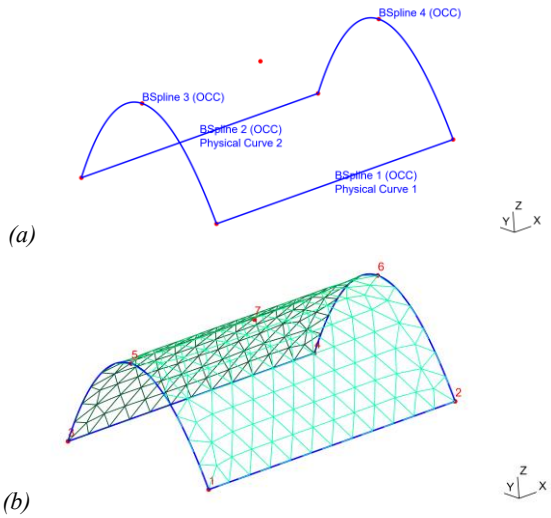

Figure 1. *Geometry and mesh of the cylindrical shell: a) Shell edges defined by B-Spline; b) Control points and mesh*

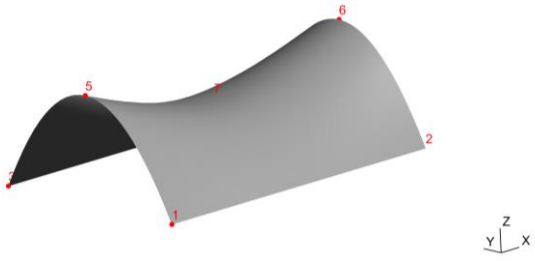

Figure 2. *Optimized shape of the ISO shell*

4.2. Laminated cylindrical shell

Consider a cylindrical shell with the same geometry as that in Section 4.1, but made of a laminated composite material with the following properties: E_2=12.1GPa; E_1=25E_2; G_{12}=G_{13}=0.5E_2; G_{23}=0.2E_2; $\nu_{21} = \nu_{12}$=0.25. Two shell configurations are investigated: an antisymmetric cross-ply layup [0/90/0/90] and an antisymmetric angle-ply layup [−45/45/−45/45].

Table 2 presents the optimization results, including the maximum bending moment and compliance (strain energy), for the two laminated shell configurations. As can be observed, both optimized configurations achieve significant reductions in bending moment and strain energy compared to their respective initial designs. For the cross-ply [0/90/0/90] configuration, the maximum bending moment decreases from 1.36 kNm to 0.76 kNm (a reduction of 44.1%), while the strain energy drops from 2.442 J to 1.075 J (56.0% reduction). The angle-ply [−45/45/−45/45] configuration shows an even more pronounced improvement in bending moment, reducing from 1.38 kNm to 0.56 kNm (59.4% reduction), although its absolute compliance values are higher than those of the cross-ply case both before and after optimization. This suggests that the cross-ply layup inherently provides greater bending stiffness in the principal load-carrying directions, whereas the angle-ply configuration, while starting from a less efficient stress distribution, benefits more substantially from shape modification.

Table 2. *Optimal results of laminated cylindrical shell*

Laminate configuration	Original		Optimized	
	M_{max}	compliance (strain)	M_{max}	compliance (strain)
[0,90,0,90]	1.36	0.0048855 (2.442)	0.76	0.0021511 (1.075)
[-45,45,-45,45]	1.38	0.0145492 (7.275)	0.56	0.0070446 (3.522)

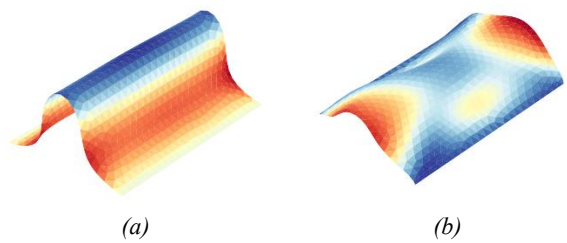

Figure 3. *Shell deformation: a) Initial shape; b) Optimized shape*

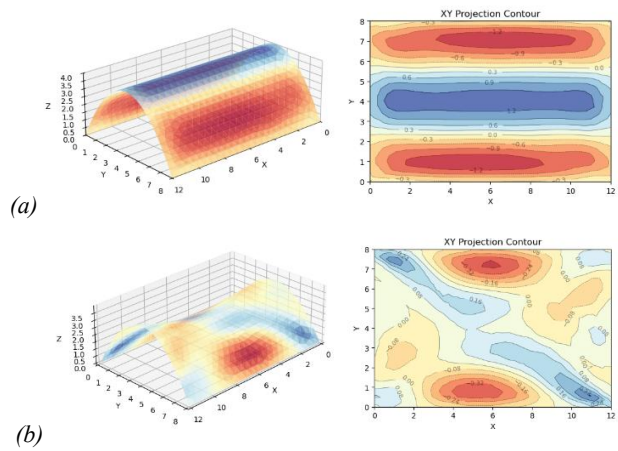

Figure 4. *Myy in angle-ply laminated shell: a) Initial shape; b) Optimized shape*

Figure 3 illustrates the deformed shapes of the initial and optimized shells for the angle-ply configuration. It is evident that the optimized shell exhibits a markedly different curvature distribution compared to the initial cylindrical shape, with the mid-region developing a saddle-like geometry that promotes membrane-dominant load transfer and suppresses local bending deformation. Figure 4 depicts the distribution of bending moment M_{yy} before and after optimization for the angle-ply laminated shell. In the initial configuration (Figure 4a), M_{yy} is concentrated in a large central region with a peak value of approximately 1.38 kNm, indicating significant bending action over a substantial portion of the shell surface. After optimization (Figure 4b), the moment distribution becomes more diffuse and the peak value is reduced to 0.56 kNm, with the high-moment zones redistributed toward the boundary regions. This redistribution is a direct consequence of the shape change, which steers the internal stress state closer to a pure membrane condition - the fundamental goal of shell shape optimization. The figures demonstrate that the procedure not only works but is also effective across two layer configurations that differ in material composition. The correspondence between the numerical results and multilayer composite material theory confirms the physical validity of the results.

In all of the above examples, the dDEmRao algorithm with the DiC filter required approximately 1700 fitness evaluations, representing a reduction of more than 60% in computational cost compared to the case without DiC (5000 fitness evaluations). This confirms the practical efficiency of the proposed algorithm for shape optimization problems where each function evaluation involves a full meshing and finite element analysis cycle.

5. Conclusion

This paper presents a systematic and fully open-source workflow for shape optimization of composite shell structures, built entirely on the Python platform. The proposed framework integrates three specialized tools - Gmsh, CALFEM-Python, and the dDEmRao-DiC metaheuristic algorithm - into a cohesive pipeline that covers all stages from geometric modeling to optimal shape identification. By adopting Spline parameterization through Gmsh, the shell mid-surface is represented by a small number of control point coordinates as design variables, ensuring globally smooth geometry while overcoming the well-known drawbacks of node-coordinate-based shape variables. Structural analysis is carried out within CALFEM-Python, augmented by the authors' self-developed modules centered on the CS-DSG3-A flat triangular shell element with full six-degree-of-freedom formulation, rendering it applicable to both isotropic and laminated composite configurations without the computational overhead associated with higher-order elements. The dDEmRao-DiC algorithm can eliminate low-potential candidate designs prior to objective function evaluation, thereby substantially reducing the number of function calls and addressing a critical practical bottleneck inherent in metaheuristic-driven structural optimization. Numerical examples on isotropic and laminated composite cylindrical shells confirm that the proposed workflow successfully identifies optimal shell geometries minimizing bending stress resultants, with meaningful shape improvements achieved within a tractable computational budget using exclusively free and open-source tools.

Acknowledgments: This research is funded by the Ministry of Education and Training under grant number B2026-XDA-05.

REFERENCES

[1] O. C. Zienkiewicz and J. S. Campbell, "Shape optimization and sequential linear programming", in *Optimum Structural Design*, R. Gallagher and O. C. Zienkiewicz, Eds. New York: Wiley, 1973, pp. 109–126.

[2] V. Braibant and C. Fleury, "Shape optimal design using B-Splines", *Computer Methods in Applied Mechanics and Engineering*, vol. 44, no. 3, pp. 247–267, 1984. https://doi.org/10.1016/0045-7825(84)90132-4

[3] T. J. Hughes, J. A. Cottrell, and Y. Bazilevs, "Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement", *Computer Methods in Applied Mechanics and Engineering*, vol. 194, no. 39–41, pp. 4135–4195, 2005. https://doi.org/10.1016/j.cma.2004.10.008

[4] J. A. Cottrell, T. J. Hughes, and Y. Bazilevs, *Isogeometric Analysis: Toward Integration of CAD and FEA*. John Wiley & Sons, 2009.

[5] C. T. Binh, N. V. Long, và T. M. Tu. "Static Behavior of FGM Cylindrical Panel With Porosities in Hygro-Thermal Environment", *The University of Danang - Journal of Science and Technology*, vol. 22, no. 11B, pp. 123-8, 2024. https://doi.org/10.31130/ud-jst.2024.533E

[6] T. Ho-Nguyen-Tan and H. G. Kim, "An efficient method for shape and topology optimization of shell structures", *Structural and Multidisciplinary Optimization*, vol. 65, no. 4, pp. 119, 2022. https://doi.org/10.1007/s00158-022-03213-0

[7] P. T. Thang, T. Nguyen-Thoi, and J. Lee, "Shape and material optimization for buckling behavior of functionally graded toroidal shells", *Thin-Walled Structures*, vol. 157, pp. 107129, 2020. https://doi.org/10.1016/j.tws.2020.107129

[8] K. D. Nguyen, T. V. Duong, J. Lee, Y. Bazilevs, and H. Nguyen-Xuan, "An efficient NURBS-based reconstruction approach to isogeometric analysis of triply periodic minimal surface structures", *Engineering with Computers*, vol. 41, no. 6, pp. 4479–4509, 2025. https://doi.org/10.1007/s00366-025-02206-z

[9] F. Yang, T. Yu, and T. Q. Bui, "Morphogenesis of free-form surfaces by an effective approach based on isogeometric analysis and particle swarm optimization", *Structures*, vol. 47, pp. 2347–2353, 2023. https://doi.org/10.1016/j.istruc.2022.12.069

[10] C. Geuzaine and J. F. Remacle, "Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities", *International Journal for Numerical Methods in Engineering*, vol. 79, no. 11, pp. 1309–1331, 2009. https://doi.org/10.1002/nme.2579

[11] J. Lindemann et al, "CALFEM for Python", [Online]. Available: https://calfem-for-python.readthedocs.io/en/latest/ [Accessed June 03, 2026].

[12] H. A. Pham and T. C. Vu, "Enhanced differential evolution-Rao optimization with distance comparison method and its application in optimal sizing of truss structures", *Journal of Computational Science*, vol. 80, pp. 102327, 2024. https://doi.org/10.1016/j.jocs.2024.102327

[13] P. H. Anh, T. H. Quoc, T. V. Binh, and T. M. T. Tu, "Development of a triangular shell element integrated in CALFEM-Python for laminated composite shell analysis", in *Proc. 16th National Scientific Conference on Solid Mechanics*, Thai Nguyen, Vietnam, Nov. 28–29, 2025, pp. 70–80. (in Vietnamese).