Author version

Published as:

Zhao, H., Hwang, J. T., & Chen, J. (2025). Open-source shape optimization for isogeometric shells using FEniCS and OpenMDAO. Engineering with Computers, 41(3), 1877-1899. https://doi.org/10.1007/s00366-025-02116-0

BibTeX for this reference:

https://lsdolab.github.io/lsdo-publications/publication/zhao2025open/

```
@article{zhao2025open,
  author = {Han Zhao and John T. Hwang and Jiun-Shyan
    Chen},
  title = {Open-source shape optimization for
    isogeometric shells using FEniCS and OpenMDAO},
  journal = {Engineering with Computers},
  year = {2025},
  volume = {41},
  number = {3},
  pages = {1877-1899},
  doi = {10.1007/s00366-025-02116-0},
  url = {https://doi.org/10.1007/s00366-025-02116-0}
}
```

Open-source shape optimization for isogeometric shells using FEniCS and OpenMDAO

Han Zhao[1], John T. Hwang[1] and J. S. Chen[2,1*]

[1]Department of Mechanical and Aerospace Engineering, University of California San Diego, 9500 Gilman Drive, La Jolla, CA 92093, USA.
[2]Department of Structural Engineering, University of California San Diego, 9500 Gilman Drive, La Jolla, CA 92093, USA.

*Corresponding author(s). E-mail(s): jsc137@ucsd.edu;

Abstract

We present an open-source Python framework for the shape optimization of complex shell structures using isogeometric analysis (IGA). IGA seamlessly integrates computer-aided design (CAD) and analysis models by employing non-uniform rational B-splines (NURBS) as basis functions, enabling the natural implementation of the Kirchhoff–Love shell model due to their higher order of continuity. We leverage the recently developed FEniCS-based analysis framework, PENGoLINS, for the direct structural analysis of shell structures consisting of a collection of NURBS patches through a penalty-based formulation. This contribution introduces the open-source implementation of gradient-based shape optimization for isogeometric Kirchhoff–Love shells with a modular architecture. Complex shell structures with non-matching intersections are handled using a free-form deformation (FFD) approach and a moving intersections formulation. The symbolic differentiation and code generation capabilities in FEniCS are utilized to compute the analytical derivatives. By integrating FEniCS with OpenMDAO, we build modular components that facilitate gradient-based shape optimization of shell structures. The modular architecture in this work supports future extensions and integration with other disciplines and solvers, making it highly customizable and suitable for a wide range of applications. We validate the design-analysis-optimization workflow through several benchmark problems and demonstrate its application to aircraft wing design optimization. The framework is implemented in a Python library named GOLDFISH (Gradient-based Optimization and Large-scale Design Framework for Isogeometric SHells) and the source code will be maintained at https://github.com/hanzhao2020/GOLDFISH.

Keywords: Isogeometric analysis, Kirchhoff–Love shells, non-matching patches coupling, moving intersections, shape optimization, open-source software

1 Introduction

Shape optimization of shell structures is crucial in engineering design to improve structural performance, efficiency, and material utilization [1]. Traditional shape optimization methods using the finite element method (FEM) often struggle with accurately representing complex geometries and generating high-quality meshes [2]. Inaccurate geometry representation introduces errors in the analysis and optimization, while poor mesh quality leads to numerical issues. To address these difficulties, shape optimization using IGA [3, 4] has emerged as a powerful method. IGA offers seamless integration between design geometries and analysis models by adopting the same mathematical description. By utilizing NRUBS [5] or other types of splines [6–8] as basis functions, IGA allows direct analysis of CAD geometries with improved accuracy [9]. This property makes IGA particularly promising for the analysis and optimization of shell structures, enabling updated design geometries to be used directly for structural analysis without the need for finite element (FE) mesh generation. The natural fulfillment of the C^1 continuity requirement in the Kirchhoff–Love shell model [10, 11] further enhances the applicability of IGA to thin-walled structures, which are widespread in structural design.

For real-world complex shell structures typically modeled by multiple NURBS surfaces, patch coupling at surface intersections is needed in analysis and optimization. Maintaining continuities of displacements and rotations at these intersections is essential. Various methods have been proposed to couple separate shell patches, including the bending strip method [12] for conforming discretizations. Nitsche-type formulations [13–17], mortar methods [18–20], and penalty-based methods [21–25] are developed for coupling of Kirchhoff–Love shell patches with non-conforming intersections. Shape optimization for non-matching shell structures poses additional technical challenges, as the intersections must be managed during the optimization process. Without proper handling, initial intersecting shell patches may become separated or self-penetrated, leading to unrealistic structures. The spline composition method [26–29] was proposed for maintaining the surface intersection during shape optimization but requires identifying master surfaces and extruding a 3D solid from them. An FFD [30]-based shape optimization approach [31] ensures the connectivity of shell patches at non-matching intersections but may suffer from significant element distortion with large intersection movements. Recently, [32] proposed an optimization method to overcome the element distortion issue, allowing relative movement between intersecting shell patches while retaining intersections.

Code transparency in the field of design optimization has been gaining more interest. OpenMDAO [33], an open-source framework for multidisciplinary design optimization (MDO), uses a modular architecture that allows users to create custom models for different disciplines and integrate them with various optimization

algorithms. Successful applications of OpenMDAO span aerospace engineering [34–36], wind energy [37], robotics [38, 39], and topology optimization [40–42]. The Python library CSDL [43] addresses large-scale MDO problems using a graph representation to automatically generate adjoint sensitivities. The CSDL-based Python library FEMO [44], coupled with the FEniCS project [45–47], was developed to solve partial differential equation (PDE)-constrained optimization problems, significantly reducing the coding effort for PDE components such as structural mechanics, fluid mechanics, and heat transfer, etc. Despite these advancements, an open-source design optimization framework using IGA has been lacking. This contribution fills that gap by developing an open-source Python library for shape optimization of complex shell structures using IGA, enabling researchers to explore the benefits of IGA in shape optimization problems and advance structural design optimization.

In the proposed framework GOLDFISH, we use OpenMDAO as the optimization toolkit, with plans to incorporate CSDL in future work to make use of the graph-based paradigm. For the IGA solver for structural analysis of non-matching shell structures, we employ the open-source package PENGoLINS [24]. PENGoLINS, a Python framework based on tIGAr [48], uses extraction techniques [49–53] to construct spline basis functions from Lagrange basis functions in the FEM solver FEniCS. tIGAr has been successfully applied in various fields [54–57]. Additionally, an open-source fluid–structure interaction framework [58, 59] is developed based on tIGAr and shows a good application for prosthetic heart valve simulation with isogeometric leaflets. PENGoLINS employs a penalty-based formulation [24] to couple the non-matching isogeometric Kirchhoff–Love shells, which is automated with the code generation technology in FEniCS. The current code framework incorporates IGA for complex Kirchhoff–Love shells with a penalty formulation [21], and employs the FFD-based and moving intersections approaches [31, 32] to handle patch intersections. The modular architecture of OpenMDAO ensures that GOLDFISH can readily couple with other disciplines and extend to more practical problems.

The remainder of this paper is outlined as follows. Section 2 provides an overview of shape optimization approaches for non-matching isogeometric shell structures. Section 3 discusses technical details and framework structures of GOLDFISH. A series of benchmarks with code implementation are presented in Section 4 to validate the framework, alongside demonstrations of aircraft wing applications. Section 5 draws conclusions and discusses future works.

2 Optimization formulations

This section presents the shape optimization schemes for isogeometric shells employed in GOLDFISH. We first review the basic isogeometric Kirchhoff–Love shell formulations in Section 2.1, then introduce the overall shape optimization strategy for isogeometric shells in Section 2.2. In Section 2.3, we discuss methods for handling patch intersections in complex shell structures.

2.1 Kirchhoff–Love shell with isogeometric discretization

The basic derivation of the Kirchhoff–Love shell is presented in this Section to provide the foundation for the subsequent shape optimization formulations. The geometry and unknowns of the Kirchhoff–Love shell are discretized isogeometrically. The Kirchhoff–Love shell theory assumes negligible transverse shear strains and constant thickness during deformation, allowing the shell to be represented by its mid-surface geometry $\mathbf{X} \in \mathbb{R}^d$, with the deformation described by $\mathbf{u} \in \mathbb{R}^d$, where d is the spatial dimension. For an input CAD geometry of the mid-surface parametrized with coordinates $\boldsymbol{\xi} = \{\xi_\alpha\}_{\alpha=1}^2$ and discretized by B-spline or NURBS basis functions $\mathbf{N}(\boldsymbol{\xi}) = \left[N_1(\boldsymbol{\xi})\ N_2(\boldsymbol{\xi})\ \dots\ N_n(\boldsymbol{\xi})\right]$, where n is the number of unknowns, the mid-surface geometry and displacement field are expressed as

$$\mathbf{X}(\boldsymbol{\xi}) = \sum_{i=1}^{n} N_i(\boldsymbol{\xi}) P_i = \mathbf{N}^{\mathrm{T}}(\boldsymbol{\xi})\mathbf{P} \quad \text{and} \quad \mathbf{u}(\boldsymbol{\xi}) = \sum_{i=1}^{n} N_i(\boldsymbol{\xi}) d_i = \mathbf{N}^{\mathrm{T}}(\boldsymbol{\xi})\mathbf{d}\,, \tag{1}$$

where $\mathbf{P}$ and $\mathbf{d}$ are the geometric and displacement control points, respectively. The deformed state of the shell is defined as

$$\mathbf{x}(\boldsymbol{\xi}) = \mathbf{X}(\boldsymbol{\xi}) + \mathbf{u}(\boldsymbol{\xi}) = \mathbf{N}^{\mathrm{T}}(\boldsymbol{\xi})(\mathbf{P} + \mathbf{d})\,. \tag{2}$$

The local covariant basis vectors in the reference and deformed configurations are formulated as

$$\mathbf{A}_\alpha = \mathbf{X}_{,\xi_\alpha} = \mathbf{N}_{,\xi_\alpha}^{\mathrm{T}}\,\mathbf{P} \quad \text{and} \quad \mathbf{a}_\alpha = \mathbf{x}_{,\xi_\alpha} = \mathbf{N}_{,\xi_\alpha}^{\mathrm{T}}\,(\mathbf{P} + \mathbf{d})\,, \tag{3}$$

and the unit normal vectors are given by

$$\mathbf{A}_3 = \frac{\mathbf{A}_1 \times \mathbf{A}_2}{\|\mathbf{A}_1 \times \mathbf{A}_2\|} \quad \text{and} \quad \mathbf{a}_3 = \frac{\mathbf{a}_1 \times \mathbf{a}_2}{\|\mathbf{a}_1 \times \mathbf{a}_2\|}\,, \tag{4}$$

where $\|\cdot\|$ is the Euclidean norm. The metric and curvature coefficients in the reference configuration are given by

$$A_{\alpha\beta} = \mathbf{A}_\alpha \cdot \mathbf{A}_\beta \quad \text{and} \quad B_{\alpha\beta} = \mathbf{A}_{\alpha,\xi_\beta} \cdot \mathbf{A}_3 = -\mathbf{A}_\alpha \cdot \mathbf{A}_{3,\xi_\beta}\,. \tag{5}$$

Analogously, the associated coefficients in the deformed configuration are

$$a_{\alpha\beta} = \mathbf{a}_\alpha \cdot \mathbf{a}_\beta \quad \text{and} \quad b_{\alpha\beta} = \mathbf{a}_{\alpha,\xi_\beta} \cdot \mathbf{a}_3 = -\mathbf{a}_\alpha \cdot \mathbf{a}_{3,\xi_\beta}\,. \tag{6}$$

The coefficients of the membrane strain tensor and curvature change tensor are defined as

$$\varepsilon_{\alpha\beta} = \frac{1}{2}(a_{\alpha\beta} - A_{\alpha\beta}) \quad \text{and} \quad \kappa_{\alpha\beta} = B_{\alpha\beta} - b_{\alpha\beta}\,. \tag{7}$$

By organizing the membrane strain tensor and curvature change tensor using Voigt notation as $\boldsymbol{\varepsilon}$ and $\boldsymbol{\kappa}$, the normal forces and bending moments of the shell, modeled with the St. Venant–Kirchhoff material model, read as

$$\mathbf{n} = t\,\mathbf{C} : \boldsymbol{\varepsilon} \quad \text{and} \quad \mathbf{m} = \frac{t^3}{12}\,\mathbf{C} : \boldsymbol{\kappa}\,, \tag{8}$$

where t is the shell thickness and $\mathbf{C}$ is the material tensor. The total energy of the Kirchhoff–Love shell theory is stated as

$$W_{\mathrm{S}} = W^{\mathrm{int}} - W^{\mathrm{ext}} = \int_S \mathbf{n} : \boldsymbol{\varepsilon} + \mathbf{m} : \boldsymbol{\kappa}\,\mathrm{d}S - \int_S \mathbf{f} \cdot \mathbf{u}\,\mathrm{d}S\,, \tag{9}$$

where $\mathbf{f}$ stands for the external force applied on $\mathbf{S}$. For a detailed derivation of the Kirchhoff–Love shell model, readers are referred to [11, Section 3]. The curvature change tensor and associated bending moments involve the second-order derivative of the displacement, which requires the basis functions in the solution space to be C^1 continuous across element boundaries. This requirement is naturally satisfied by the B-spline and NURBS basis functions, which demonstrate excellent results [10, 60, 61].

2.2 Shape optimization for isogeometric shells

The shape optimization for shell structures can be formulated as the following form

$$\begin{aligned} &\underset{\mathbf{P}}{\text{minimize}}\ f(\mathbf{P}) \\ &\text{subject to}\ \mathbf{g}(\mathbf{P}) \leq \mathbf{0} \\ &\qquad\qquad\ \mathbf{h}(\mathbf{P}) = \mathbf{0}\,, \end{aligned} \tag{10}$$

where $\mathbf{P}$ are the design variables, f is the objective function, and $\mathbf{g}$ and $\mathbf{h}$ represent the inequality and equality constraints, respectively. In shape optimization problems using IGA, the design variables $\mathbf{P}$ are coordinates of the control points that define the shell geometry using B-spline or NURBS basis functions. To enable gradient-based design optimization, it is necessary to derive the total derivative of the objective function with respect to design variables [62, 63]. For shape optimization, the model outputs are typically affected by the displacement of the structure, which depends on the design variables and is referred to as the state variable. Therefore, we express the the objective function as $f(\mathbf{P}, \mathbf{d}(\mathbf{P}))$, and the total derivative is given by

$$\mathrm{d}_{\mathbf{P}} f = \partial_{\mathbf{P}} f + (\partial_{\mathbf{d}} f)^{\mathrm{T}}\,\mathrm{d}_{\mathbf{P}}\mathbf{d}\,, \tag{11}$$

where partial derivatives $\partial_{\mathbf{P}} f$ and $\partial_{\mathbf{d}} f$ can be readily computed. However, to obtain the total derivative in (11), we need to derive the total derivative of the structural displacement with respect to shell control points $\mathrm{d}_{\mathbf{P}}\mathbf{d}$. In the context of shape optimization for shell structures, the relation between the geometry control points $\mathbf{P}$

and displacement $\mathbf{d}$ is typically governed by a system of equations derived from discretized PDEs

$$\mathbf{R}_\mathrm{S}(\mathbf{P}, \mathbf{d}) = \mathbf{0} \,, \tag{12}$$

where $\mathbf{R}_\mathrm{S}$ represents the residual equations. They can be obtained by discretizing the partial derivative of the total energy in (9) with respect to shell displacement

$$\mathbf{R}_\mathrm{S} = \partial_\mathbf{d} W_\mathrm{S} \,. \tag{13}$$

Equation (13) represents a system of nonlinear equations, which can be linearized and solved by Newton–Raphson method for the displacement increments

$$\partial_\mathbf{d} \mathbf{R}_\mathrm{S} \, \Delta\mathbf{d} = -\mathbf{R}_\mathrm{S} \,. \tag{14}$$

The discretization of $\partial_\mathbf{d} \mathbf{R}_\mathrm{S}$ represents the stiffness matrix $\mathbf{K}_\mathrm{S} = \partial_\mathbf{d} \mathbf{R}_\mathrm{S}$ of the shell structure.

With the adjoint method, the total derivative $\mathrm{d}_\mathbf{P}\mathbf{d}$ in (11) can be obtained by taking the total derivative of $\mathbf{R}_\mathrm{S}$ with respect to $\mathbf{P}$

$$\mathrm{d}_\mathbf{P} \mathbf{R}_\mathrm{S} = \partial_\mathbf{P} \mathbf{R}_\mathrm{S} + \partial_\mathbf{d} \mathbf{R}_\mathrm{S} \, \mathrm{d}_\mathbf{P}\mathbf{d} = \mathbf{0} \,, \tag{15}$$

and substituting $\partial_\mathbf{d} \mathbf{R}_\mathrm{S}$ by $\mathbf{K}_\mathrm{S}$

$$\mathrm{d}_\mathbf{P}\mathbf{d} = -\mathbf{K}_\mathrm{S}^{-1} \, \partial_\mathbf{P} \mathbf{R}_\mathrm{S} \,. \tag{16}$$

The partial derivative $\partial_\mathbf{P} \mathbf{R}_\mathrm{S}$ can be computed from the residual vector in a similar way to the stiffness matrix.

Substituting (16) into (11), we can obtain the total derivative for the shape optimization problem of an isogeometric Kirchhoff–Love shell as

$$\mathrm{d}_\mathbf{P} f = \partial_\mathbf{P} f - (\partial_\mathbf{d} f)^\mathrm{T} \, \mathbf{K}_\mathrm{S}^{-1} \, \partial_\mathbf{P} \mathbf{R}_\mathrm{S} \,. \tag{17}$$

Using the total derivative, we can apply gradient-based optimization algorithms [64] to determine the optimal shape of a given baseline design of a shell structure.

2.3 Treatment of surface intersections

CAD geometries are typically modeled by multiple NURBS patches for real-world shell structures. The multi-patch NURBS-based geometries exhibit intersections between patches, where displacement and angular compatibilities need to be maintained during structural analysis to glue the patches together. An illustrative example of two shell patches with one intersection is depicted in Figure 1. Additionally, it is crucial to preserve compatibility at the patch intersections during shape updates in the optimization loop so that the optimized shell structures will not improperly separate or become unrealistic structures. The following sections discuss the methods used to handle shape optimization for shell structures consisting of multiple spline patches.

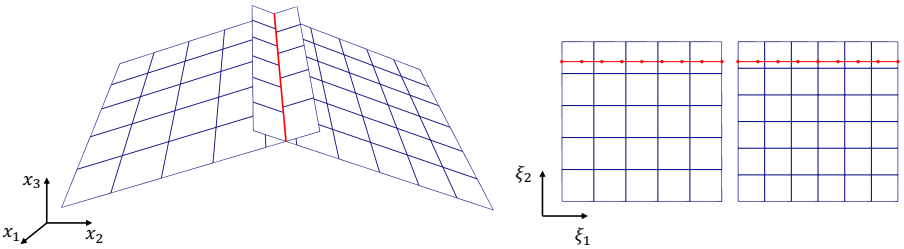


Fig. 1: An illustrative example of two shell patches with one intersection, where shell patches are described by NURBS surfaces. A topologically 1D, geometrically 2D quadrature mesh is generated in the parametric space to maintain the displacement and rotational continuity through a penalty formation.

2.3.1 Compatibility conservation of surface intersections using FFD

In the proposed code framework, we implement the FFD-based shape optimization formulation [31], combined with the Lagrange extraction technique [51], to preserve intersections between separately modeled spline surfaces within a single CAD geometry. In this approach, the entire CAD geometry of a shell structure is embedded in a trivariate B-spline block. The control points of the FFD block are related to the Lagrange nodal points of the shell geometry interpolated with Lagrange polynomials, preserving patch intersections inside the FFD block while modifying the shell geometry. The Lagrange nodal points of the shell structure are updated through the shape changes in the 3D FFD block during shape optimization. Subsequently, a pseudo-inverse system is solved to approximate the control points of the NURBS geometry from the updated Lagrange nodal points using the Lagrange extraction matrices. This approach is illustrated in Figure 2, where the shape of shell patches follows the shape change of the FFD block. Since the shape changes continuously inside the FFD block, the embedded surfaces can maintain their intersections without relative movement. Therefore, the control points of the 3D block are considered design variables.

Using the Lagrange extraction method, the NURBS basis functions of spline patch k can be expressed by Lagrange polynomials $\mathbf{N}_{\mathrm{L}}^{k}$,

$$\mathbf{N}^{k} = \mathbf{M}^{k^{\mathrm{T}}} \mathbf{N}_{\mathrm{L}}^{k} \,, \tag{18}$$

where $\mathbf{M}^{k}$ denotes the extraction matrix. Each entry of $\mathbf{M}^{k}$ is the evaluation of the NURBS basis functions $\mathbf{N}^{k}$ at the parametric location of an Lagrange nodal point $\boldsymbol{\xi}^{k}$,

$$M_{ij}^{k} = N_{j}^{k}(\boldsymbol{\xi}_{i}^{k}) \,. \tag{19}$$

The relationship between shell patches' NURBS control points $\mathbf{P}$ and Lagrange nodal points $\mathbf{P}_{\mathrm{L}}$ for the k-th shell patch can be obtained from the Lagrange extraction matrix

$$\mathbf{M}^{k}\mathbf{P}^{k} = \mathbf{P}_{\mathrm{L}}^{k} \,, \tag{20}$$

It is noted that the extraction operator $\mathbf{M}^k$ is a nonsquare matrix and a pseudo-inverse is necessary to obtain the total derivative of the k-th spline patch,

$$\mathrm{d}_{\mathbf{P}_\mathrm{L}^k}\mathbf{P}^k = \left(\mathbf{M}^{k\mathrm{T}}\mathbf{M}^k\right)^{-1}\mathbf{M}^{k\mathrm{T}}\,. \tag{21}$$

And the total derivative for the entire shell structure with m spline patches is

$$\mathrm{d}_{\mathbf{P}_\mathrm{L}}\mathbf{P} = \mathrm{diag}\left(\mathrm{d}_{\mathbf{P}_\mathrm{L}^1}\mathbf{P}^1, \mathrm{d}_{\mathbf{P}_\mathrm{L}^2}\mathbf{P}^2, \ldots, \mathrm{d}_{\mathbf{P}_\mathrm{L}^m}\mathbf{P}^m\right)\,. \tag{22}$$

The total derivative between the control points of the FFD block and Lagrange nodal points of the k-th shell patch can be derived in a similar way

$$\mathbf{A}^k\mathbf{P}_\mathrm{FFD} = \mathbf{P}_\mathrm{L}^k\,, \tag{23}$$

and the matrix $\mathbf{A}^k$ is constructed as

$$A_{ij}^k = N_{\mathrm{FFD}\,j}(\mathbf{P}_{\mathrm{L}i}^k)\,, \tag{24}$$

where $\mathbf{N}_\mathrm{FFD}$ are the B-spline basis functions of the FFD block and the identity geometric mapping for the FFD block is employed for simplicity. The associated total derivative $\mathrm{d}_{\mathbf{P}_\mathrm{FFD}}\mathbf{P}_\mathrm{L}$ is expressed as

$$\mathrm{d}_{\mathbf{P}_\mathrm{FFD}}\mathbf{P}_\mathrm{L} = \mathrm{diag}\left(\mathbf{A}^1, \mathbf{A}^2, \ldots, \mathbf{A}^m\right)\,. \tag{25}$$

Analogous to (17), the total derivative of the FFD-based shape optimization approach for multi-patch shell structure is expressed as

$$\mathrm{d}_{\mathbf{P}_\mathrm{FFD}}f = \left(\partial_\mathbf{P}f - (\partial_\mathbf{d}f)^\mathrm{T}\,\mathbf{K}^{-1}\,\partial_\mathbf{P}\mathbf{R}\right)\mathrm{d}_{\mathbf{P}_\mathrm{L}}\mathbf{P}\,\mathrm{d}_{\mathbf{P}_\mathrm{FFD}}\mathbf{P}_\mathrm{L}\,, \tag{26}$$

where $\mathbf{P}^\mathrm{T} = \left[\mathbf{P}^{1\mathrm{T}}\ \mathbf{P}^{2\mathrm{T}}\ \ldots\ \mathbf{P}^{m\mathrm{T}}\right]$ is the B-spline or NURBS control points of all shell patches in the CAD geometry, and $\mathbf{P}_\mathrm{L}^\mathrm{T} = \left[\mathbf{P}_\mathrm{L}^{1\,\mathrm{T}}\ \mathbf{P}_\mathrm{L}^{2\,\mathrm{T}}\ \ldots\ \mathbf{P}_\mathrm{L}^{m\mathrm{T}}\right]$ is the Langrange nodal points of the shell geometry. $\mathbf{P}_\mathrm{FFD}$ denotes the control points of the B-spline FFD block. And $\mathbf{d}^\mathrm{T} = \left[\mathbf{d}^{1\mathrm{T}}\ \mathbf{d}^{2\mathrm{T}}\ \ldots\ \mathbf{d}^{m\mathrm{T}}\right]$ represents the displacements for all shell patches. Additionally, $\mathbf{R}(\mathbf{P}, \mathbf{d})$ is the residual vector of the multi-patch shell structure, and $\mathbf{K}$ is the associated stiffness matrix.

For the structural analysis of shell geometries consisting of a collection of NURBS surfaces, we employ a penalty coupling formulation to maintain displacement and rotational compatibility at the intersections. The residual vector and stiffness matrix of the multi-patch shell structure are expressed as

$$\mathbf{R} = \partial_\mathbf{d}\left(\sum_{k=1}^{m} W_\mathrm{S}^k + \sum_{l=1}^{r} W_\mathrm{pen}^l\right) \quad \text{and} \quad \mathbf{K} = \partial_\mathbf{d}\mathbf{R}\,, \tag{27}$$

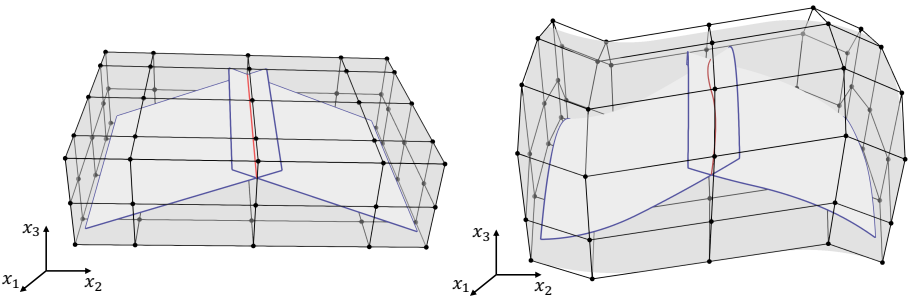


Fig. 2: Illustration of the FFD-based shape optimization approach. The intersecting shell patches are embedded in a trivariate B-spline FFD block, with its control points serving as design variables. Modifying the shape of the FFD block alters the Lagrange nodal points of the shell patches, ensuring the preservation of patch intersections. The updated NURBS control points are then obtained from the Lagrange nodal points using Lagrange extraction matrices.

where W_{pen} is the penalty energy between two intersecting shell patches as illustrated in Figure 1 and r is the number of intersections. The penalty energy of the l-th intersection between shell patches with indices l_1 and l_2 is expressed as

$$W_{\text{pen}}^{l} = \int_{\mathcal{L}} \alpha_d \|\mathbf{u}^{l_1} - \mathbf{u}^{l_2}\|^2 + \alpha_r \left((\sin\phi - \sin\phi_0)^2 + (\cos\phi - \cos\phi_0)^2\right) \mathrm{d}\mathcal{L}\,, \tag{28}$$

where $\mathcal{L}$ represents the intersection curve with associated displacements $\mathbf{u}^{l_1}$ and $\mathbf{u}^{l_2}$, and ϕ_0 and ϕ denote the angles between the two shell patches before and after deformation, respectively. The first term in (28) ensures that the two intersecting shells have the same displacement along the intersection, while the second term maintains the angle between the two shell patches during deformation. The penalty parameters α_d and α_r can be determined based on the material properties of the problem and shell discretizations. Further details of the penalty formulation can be found in [21, Section 2], and the open-source implementation using FEniCS is presented in [24].

Substituting (22) and (25) into (26), we can obtain the total derivative of the FFD-based approach for non-matching shell structures to perform shape optimization. The parametric locations of surface intersections are maintained by embedding the shell patches into the FFD block and there is no relative movement between shell patches.

2.3.2 Shape updates with moving intersections

While the FFD-based approach is effective in preserving surface intersections in shell structural optimization, it can significantly deteriorate the quality of shell elements if shape optimization involves substantial movement of patch intersections. To address this issue, [32] proposed an optimization approach for shell structures with moving intersections. In this approach, shell patches are allowed to move relative to other intersecting patches in the parametric coordinates during the shape update process

without distorting the shell elements, while the coupling of updated surface intersections is still enforced in the structural analysis stage. This relative movement of shell patches in the parametric coordinates helps maintain the quality of shell elements, even when the intersections undergo significant relocation. Figure 3 shows the updated shape patches from the original design in Figure 1, where parametric locations of the intersection are updated accordingly due to the relative movement of the shell patches during the optimization process. This method is achieved by considering the parametric locations of the surface intersection as state variables in the optimization framework and formulating a differentiable residual that accounts for the control points of shell patches with indices k_1 and k_2, as well as the associated parametric locations of the intersection $\tilde{\boldsymbol{\xi}}^{k_1}$ and $\tilde{\boldsymbol{\xi}}^{k_2}$

$$\mathbf{R}_{\mathcal{L}} = \begin{bmatrix} \mathbf{N}^{k_1}(\tilde{\boldsymbol{\xi}}^{k_1})\mathbf{P}^{k_1} - \mathbf{N}^{k_2}(\tilde{\boldsymbol{\xi}}^{k_2})\mathbf{P}^{k_2} \\ h_j^{k_1} - h_{j-1}^{k_1} \\ \tilde{\xi}_a^{k_1} - 1 \backslash 0 \\ \tilde{\xi}_b^{k_1} - 1 \backslash 0 \end{bmatrix} = \mathbf{0}\,, \tag{29}$$

where $h_j^{k_1}$ is the element size of the intersection in physical space. The first equation in (29) ensures that the locations of nodal points of the intersection, associated with two spline patches, coincide in physical space. The second equation of in (29) ensures that adjacent physical elements at the intersection have the same length, thereby maintaining a uniform physical element size along the intersection. The last two equations specify the two end points of the intersection, which have parametric coordinates of 0 or 1 depending on the surface edge, assuming the spline patches have a unit square parametric domain with the lower left corner at (0,0).

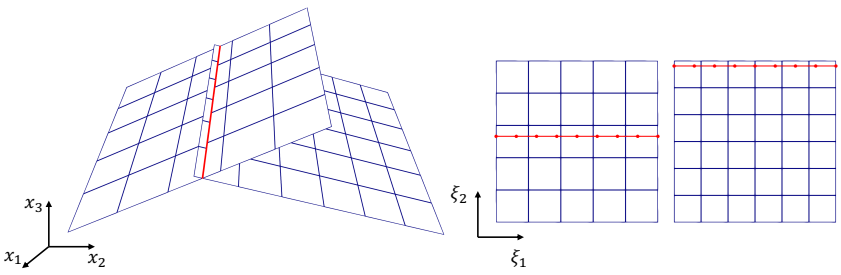


Fig. 3: Updated design of the intersecting patches from Figure 1. The shape of the two shell patches is allowed to change relative to each other, and parametric coordinates of the intersection are determined by solving the implicit equation (29).

The inclusion of parametric coordinates of patch intersections in the optimization problem impacts the structural analysis results, making the structural displacement

dependent on both $\mathbf{P}$ and $\tilde{\boldsymbol{\xi}}$. This relation is expressed as

$$\mathbf{R}(\mathbf{P}, \tilde{\boldsymbol{\xi}}, \mathbf{d}) = \mathbf{0} \,. \tag{30}$$

The non-matching residual of structural analysis $\mathbf{R}$ and associated stiffness matrix $\mathbf{K} = \partial_{\mathbf{d}}\mathbf{R}$ are defined in (27). In contrast to the approach in Section 2.3.1, where the relative location between intersecting shell patches within one FFD-block remains fixed during the shape optimization process and $\tilde{\boldsymbol{\xi}}$ is not included in the optimization, this method requires the parametric locations of patch intersections to be treated as state variables. To enable gradient-based optimization, the partial derivatives $\partial_{\mathbf{P}}\mathbf{R}$ and $\partial_{\tilde{\boldsymbol{\xi}}}\mathbf{R}$ in (30) need to be computed. The derivation of these derivatives is discussed in detail in [32, Section 3]. It is noted that while the first-order derivatives of the spline basis functions are considered in the penalty energy for shell coupling, second-order derivatives are required in the partial derivative $\partial_{\tilde{\boldsymbol{\xi}}}\mathbf{R}$, thereby C^1 continuity across element boundaries is necessary. This requirement is naturally satisfied by the NURBS basis functions.

By incorporating differentiable intersections, the total derivative for the shape optimization problem is formulated as

$$\mathrm{d}_{\mathbf{P}} f = \partial_{\mathbf{P}} f - (\partial_{\mathbf{d}} f)^{\mathrm{T}} \mathbf{K}^{-1} \left[\partial_{\mathbf{P}}\mathbf{R} - \partial_{\tilde{\boldsymbol{\xi}}}\mathbf{R} \, (\partial_{\tilde{\boldsymbol{\xi}}}\mathbf{R}_{\mathcal{L}})^{-1} \partial_{\mathbf{P}}\mathbf{R}_{\mathcal{L}} \right] . \tag{31}$$

Since the number of design variables is typically much larger than the number of model outputs, the adjoint method is employed to compute sensitivities in the optimization problem for enhanced efficiency.

3 Design of GOLDFISH

Section 3.1 outlines the design of GOLDFISH and its software dependencies, which facilitate the open-source implementation. A discussion of the key OpenMDAO components for shape optimization of isogeometric shell structures is presented in Section 3.2.

3.1 Software dependencies and workflow

The design of GOLDFISH leverages the code generation capabilities in FEniCS to automate the computation of symbolic Gateaux derivatives for gradient-based optimization, while OpenMDAO is used to ensure modularity and flexibility across various design conditions and disciplines. The Python library is built on a suite of open-source software dependencies, streamlining the entire design-analysis-optimization workflow.

The structural analysis is performed using PENGoLINS [24], a Python library designed for complex shell structures modeled by isogeometric Kirchhoff–Love theory. In PNGoLINS, CAD geometries of shell structures are discretized isogeometrically and are directly available for analysis without FE mesh generation. Shell CAD geometries consisting of a collection of non-conforming NURBS surfaces are coupled using a penalty formulation [21], where a penalty energy, as discussed in (28),

is integrated to preserve displacement continuity and angular compatibility along the intersection between shell patches. In the coupling procedure, a geometrically 2D, topologically 1D quadrature mesh is generated in the parametric space between two intersecting shell patches to serve as the integration domain for the penalty energy. This quadrature mesh is first positioned at the parametric location of the intersection with respect to the first shell patch to interpolate the displacement and covariant basis vectors. It then performs a similar operation with respect to the second shell patch. With the interpolated displacements and rotational quantities, the penalty energy and associated derivatives can be computed using FEniCS, enabling us to solve the coupled system of the complex shell structure. A schematic visualization of the coupling procedure is shown in Figure 1.

PENGoLINS makes use of the Python interface of OpenCASCADE [65] as the geometry engine so that CAD geometries can be imported to extract knot vectors and control points directly and use them for IGA. Another key feature inherited from OpenCASCADE is the ability to approximate NURBS patch intersections. This functionality is used to compute the parametric locations $\tilde{\boldsymbol{\xi}}$ of surface intersections, which define the integration domains for the penalty energy in (28). Meanwhile, the computed parametric coordinates serve as the initial guess when solving the implicit equation between NURBS surface control points and intersection parametric coordinates in (29). The IGA capabilities in PENGoLINS are powered by tIGAr, a Python library developed based on FEniCS that leverages the existing FE assembly routines. tIGAr constructs NURBS basis functions from Lagrange polynomials using the extraction technique [49, 51], with the Galerkin approximation process fully automated using FEniCS.

The numerical optimization is conducted using OpenMDAO [33], which computes the total derivative of optimization problems using direct or adjoint methods, depending on the problem specifications. In shape optimization problems, the number of design variables is typically much larger than the number of model outputs, such as objective functions and constraints. OpenMDAO automatically organizes the partial derivatives provided by components into total derivatives using the adjoint method, significantly improving the efficiency of derivative computation. The modular design of OpenMDAO also facilitates and standardizes the implementation of individual components for the optimization problem. Each partial derivative in (26) and (31) can be implemented as a standard OpenMDAO component, which is connected automatically during the optimization process. This modular design greatly enhances the flexibility and applicability of the code framework, allowing it to be adapted to more customized problems. A series of essential components for shell shape optimization are discussed in Section 3.2. The optimization problem can be solved using the open-source optimizer SLSQP [66] or the commercial optimizer SNOPT [67]. A schematic code structure of GOLDFISH is illustrated in Figure 4.

Figure 4 illustrates the streamlined workflow of GOLDFISH, where users only need to provide the NURBS-based CAD geometry of the shell structure and define the optimization problem by specifying objective functions and constraints within OpenMDAO. The code framework then automatically performs the structural analysis and shape optimization on the NURBS-based geometry without FE mesh

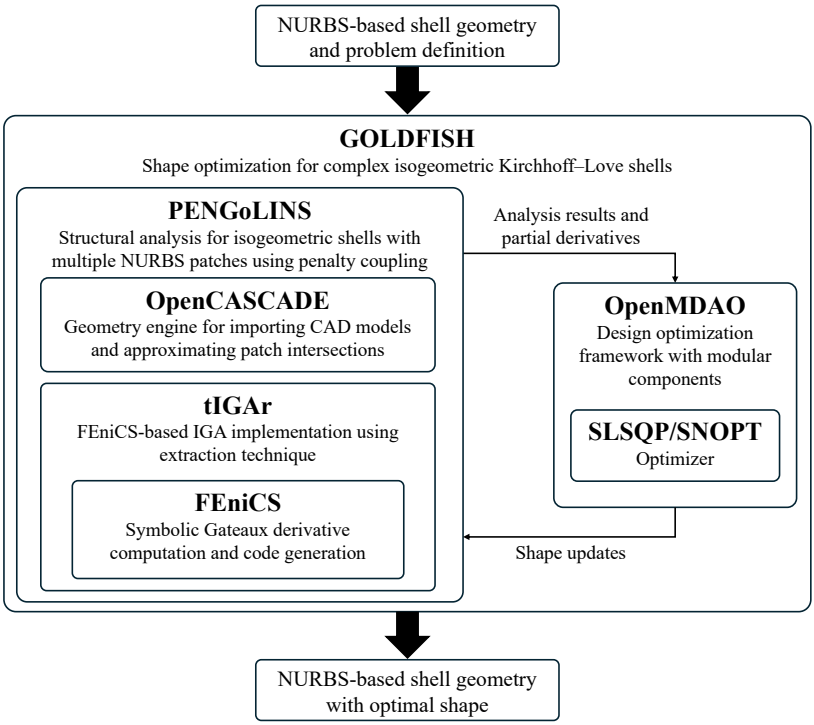


Fig. 4: The design of Python library GOLDFISH and its software dependencies. PENGoLINS is used for structural analysis and OpenMDAO is employed for numerical optimization. Analytical partial derivatives are computed in individual components in GOLDFISH. Both input and output for the software are NURBS-based shell geometry.

generation. A Lagrange extraction matrix is generated using (19) for each shell patch to express the IGA spline bases by the Lagrange polynomial bases used in FEniCS. Both types of basis functions are associated with the 2D parametric meshes of the spline patches, as shown in Figure 1. These 2D parametric meshes are defined by the knot vectors and are generated automatically in FEniCS. The final output is also a NURBS-based geometry with updated control points that define the optimal shell shape. Throughout the optimization loop, shape updates and structural analysis are all performed directly on the CAD geometry. This integration considerably simplifies the shape optimization process.

3.2 Optimization components of shell shape optimization

This section reviews the essential building blocks for an IGA-based shape optimization problem. For the shape optimization problem described in (10) with the associated total derivative (17), the design variables are the coordinates of control points $\mathbf{P}$ of the NURBS surface defining the shell geometry. A standard InputsComp

is created to provide the independent design variables to the following core components.

- *DispComp*: An implicit OpenMDAO component that takes control points $\mathbf{P}$ of the shell surface as input and returns the corresponding displacement $\mathbf{d}$ by solving the isogeometric Kirchhoff–Love shell problem $\mathbf{R}_S(\mathbf{P}, \mathbf{d}) = \mathbf{0}$, where $\mathbf{R}_S$ is defined in (13). Meanwhile, this component computes partial derivatives $\partial_{\mathbf{P}}\mathbf{R}_S$ and $\partial_{\mathbf{d}}\mathbf{R}_S$.

- *ObjectiveComp*: An explicit OpenMDAO component calculates the objective function f for the optimization problem, which typically depends on the shape of the shell and its displacements $f(\mathbf{P}, \mathbf{d})$. Additionally, this component provides partial derivatives $\partial_{\mathbf{P}} f$ and $\partial_{\mathbf{d}} f$.

With the partial derivatives computed by the two core components, the total derivative (17) is constructed automatically in OpenMDAO to guide shape updates until the optimal solution is reached.

3.2.1 Components for FFD-based shape optimization

As discussed in Section 2.3.1, the FFD-based approach is applied to real-world CAD geometries consisting of multiple non-conforming NURBS patches to maintain the intersections. In this approach, control points of the trivariate B-spline block $\mathbf{P}_{\text{FFD}}$ serve as the design variables. Additional components are implemented for this approach to automate the optimization process, as outlined below.

- *CPFFD2SurfComp*: An explicit component computes the corresponding Lagrange nodal points $\mathbf{P}_{\text{L}}$ for the given control points of the FFD block $\mathbf{P}_{\text{FFD}}$, as illustrated in (23)–(25). It also provides the derivative $\mathrm{d}_{\mathbf{P}_{\text{FFD}}}\mathbf{P}_{\text{L}}$ by evaluating the basis functions of the FFD block at the Lagrange nodal points of the shell surfaces in their initial configuration.

- *CPFE2IGAComp*: An implicit component solves for the NURBS control points of the shell patches $\mathbf{P}$ given the input Lagrange nodal points $\mathbf{P}_{\text{L}}$, using the residual equation $\mathbf{M}\mathbf{P} - \mathbf{P}_{\text{L}} = \mathbf{0}$. The matrix $\mathbf{M}$ is the global extraction operator for the entire shell structure. This implicit component is employed to bypass the large matrix inversion as shown in (21). Since the degrees of freedom (DoFs) of $\mathbf{P}$ are fewer than $\mathbf{P}_{\text{L}}$, $\mathbf{M}$ is a nonsquare matrix. The resulting $\mathbf{P}$ is interpreted as a least-squares fit to $\mathbf{P}_{\text{L}}$.

- *DispComp*: An implicit component solves for the displacement $\mathbf{d}$ of the multi-patch shell structure given the input shell NURBS control points $\mathbf{P}$. Unlike the shape optimization of single patch shell structure, the residual becomes $\mathbf{R}(\mathbf{P}, \mathbf{d}) = \mathbf{0}$ as discussed in (27), which includes a penalty energy to couple the intersecting shell patches. This component also returns the partial derivatives of the non-matching residual $\partial_{\mathbf{P}}\mathbf{R}$ and $\partial_{\mathbf{d}}\mathbf{R}$.

By connecting with the previously mentioned InputsComp and ObjectiveComp, we can perform shape optimization for the complex shell structures while preserving the non-matching patch intersection throughout the optimization process. Figure 5 illustrates the component structure of the FFD-based shape optimization. This code structure is verified in the non-matching arch shape optimization example in Section 4.1.

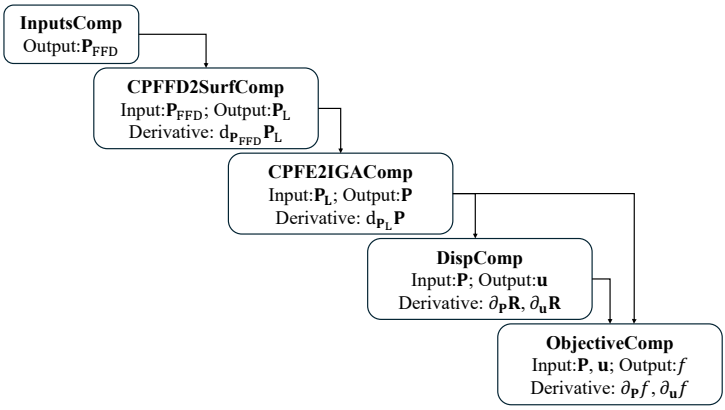


Fig. 5: Component structure for shell shape optimization using the FFD-based approach.

An illustrative implementation example is provided in the following code snippets, demonstrating the use of GOLDFISH within the Python environment. First, we import the OpenMDAO and GOLDFISH libraries.

```
import openmdao.api as om
from GOLDFISH.nonmatching_opt_om import *
```

Next, a class `ShapeOptGroupFFD` inherited from the OpenMDAO group is created for the FFD-based shape optimization problem, and the relevant parameters are initialized. The input of the class is an instance of the non-matching problem `NonMatchingOptFFD`, which takes the CAD geometry, analysis definitions, and optimization conditions. These problem definitions are demonstrated in Section 4.1.

```
class ShapeOptGroupFFD(om.Group):
    def initialize(self):
        self.options.declare('nonmatching_opt_ffd')
    # Define optimization related parameters
    def init_parameters(self):
        self.nmopt_ffd = self.options['nonmatching_opt_ffd']
        self.opt_field = self.nmopt_ffd.opt_field
        self.init_cpffd_design = self.nmopt_ffd.\
                            shopt_init_cpffd_design
        self.input_cp_shapes = [cpffd.size for cpffd
                                in self.init_cpffd_design]
```

A list of OpenMDAO components is then added to the group. The following code snippet shows the input component which takes the control points of the FFD block as the design variables. The `CPFFD2SurfComp` and `CPFE2IGAComp` components connect control points of the FFD block to the NURBS control points of shell patches using the formulation discussed in Section 2.3.1.

```
    def setup(self):
        # Add inputs comp
        inputs_comp = om.IndepVarComp()
        for i, field in enumerate(self.opt_field):
            inputs_comp.add_output(
                VARNAME_CP_FFD_DESIGN+str(field),
                shape=self.input_cpffd_shapes[i],
                val=self.init_cpffd_design[i])
        self.add_subsystem('inputs_comp', inputs_comp,
                           promotes=['*'])
        # Add FFD comp
        self.ffd2surf_comp = CPFFD2SurfComp(
            nonmatching_opt_ffd=self.nmopt_ffd)
        self.ffd2surf_comp.init_parameters()
        self.add_subsystem('CPFFD2Surf_comp',
            self.ffd2surf_comp, promotes=['*'])
        # Add CPFE2IGA comp
        self.cpfe2iga_comp = CPFE2IGAComp(
            nonmatching_opt=self.nmopt_ffd)
        self.cpfe2iga_comp.init_parameters()
        self.add_subsystem('CPFE2IGA_comp',
            self.cpfe2iga_comp, promotes=['*'])
```

Furthermore, the `DispStatesComp` component performs the IGA on the updated CAD geometry and returns the structural response along with partial derivatives. In this example, we use the internal energy as the objective function, so the `IntEnergyComp` component is added to the group.

```
        # Add displacement comp
        self.disp_states_comp = DispStatesComp(
            nonmatching_opt=self.nmopt_ffd)
        self.disp_states_comp.init_parameters()
        self.add_subsystem('disp_comp',
            self.disp_states_comp, promotes=['*'])
        # Add internal energy comp (obj function)
        self.int_energy_comp = IntEnergyComp(
            nonmatching_opt=self.nmopt_ffd)
        self.int_energy_comp.init_parameters()
        self.add_subsystem('int_energy_comp',
            self.int_energy_comp, promotes=['*'])
```

Finally, we can specify the design variables and objective functions within the group to complete the setup of the optimization problem. Additionally, equality and inequality constraints can be specified in the group through the `self.add_constraint` method.

```
        # Add design variable and objective
        for i, field in enumerate(self.opt_field):
            self.add_design_var(
                VARNAME_CP_FFD_DESIGN+str(field),
                lower=DESVAR_L[i], upper=DESVAR_U[i])
```

```
        self.add_objective(VARNAME_INT_ENERGY)
```

3.2.2 Components for moving intersections

When the optimal shell structures require significant repositioning of intersections compared to the baseline design, we employ the moving intersections approach proposed in [32]. This approach allows relative movement of shell patches, ensuring that shell elements maintain good quality in the optimized geometry. In this method, we implement the multilevel design approach [68, 69] to distinguish the design model and analysis model. As such, we can select the dimension of the design space independently from the analysis model, which typically has more DoFs for accurate analysis. By selecting a design model with much fewer DoFs than the analysis model, we can expedite the convergence of the optimizer while preserving the same geometry as the analysis model without introducing geometric errors. The multilevel design is achieved through order elevation and knot refinement of the NURBS surfaces in the design model. As a result, the design variables in this scenario are the control points of the design model. The essential components for shell shape optimization with moving intersections, combined with multilevel design, are listed below.

- *OrderElevationComp*: An explicit OpenMDAO component takes the control points of the coarse design model as input and returns corresponding control points after order elevation, following the relation $\mathbf{N}_{\mathrm{DV}}(\boldsymbol{\xi})\mathbf{P}_{\mathrm{DV}} = \mathbf{N}_{\mathrm{OE}}(\boldsymbol{\xi})\mathbf{P}_{\mathrm{OE}}$, where $\mathbf{N}_{\mathrm{OE}}(\boldsymbol{\xi})$ represents higher-order NURBS basis functions than $\mathbf{N}_{\mathrm{DV}}(\boldsymbol{\xi})$ but with the same unique knots.

- *KnotRefinementComp*: An explicit component computes control points of the analysis model $\mathbf{P}$ from the given input $\mathbf{P}_{\mathrm{OE}}$ using knot refinement for NURBS surfaces. This is achieved using similar formulation $\mathbf{N}_{\mathrm{OE}}(\boldsymbol{\xi})\mathbf{P}_{\mathrm{OE}} = \mathbf{N}(\boldsymbol{\xi})\mathbf{P}$, where $\mathbf{N}(\boldsymbol{\xi})$ has the same order as $\mathbf{N}_{\mathrm{OE}}(\boldsymbol{\xi})$ but with refined interior knots.

- *CPIGA2XiComp*: An implicit component takes control points of the analysis model $\mathbf{P}$ as inputs and solves the residual equation $\mathbf{R}_{\mathcal{L}}$ defined in (29) to determine the parametric coordinates of the movable intersections $\tilde{\boldsymbol{\xi}}$. Partial derivatives $\partial_{\mathbf{P}}\mathbf{R}_{\mathcal{L}}$ and $\partial_{\tilde{\boldsymbol{\xi}}}\mathbf{R}_{\mathcal{L}}$ are computed in this component.

- *DispMintComp*: An implicit component similar to the *DispComp* in Section 3.2.2 that solves structural displacement $\mathbf{d}$ using the penalty-based coupling formulation for isogeometric shells. In addition to control points of the shell patches $\mathbf{P}$, this component also takes the parametric coordinates of the moving intersections $\tilde{\boldsymbol{\xi}}$ as an input to formulate the residual $\mathbf{R}(\mathbf{P}, \tilde{\boldsymbol{\xi}}, \mathbf{d})$. This component also computes the partial derivatives $\partial_{\mathbf{P}}\mathbf{R}$, $\partial_{\tilde{\boldsymbol{\xi}}}\mathbf{R}$ and $\partial_{\mathbf{d}}\mathbf{R}$.

The total derivative in (31), along with a multilevel design method, are obtained by connecting the partial derivatives provided by the components mentioned above. The

connections between individual components are outlined in Figure 6. A numerical example verifies this code structure is demonstrated in Section 4.2.

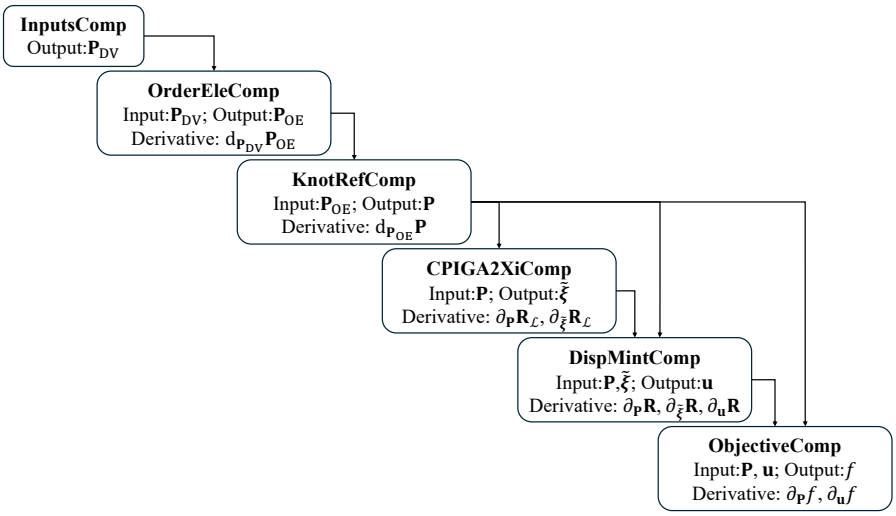


Fig. 6: Component structure for shape optimization with moving intersections and the multilevel design method.

Code snippets are demonstrated to create the OpenMDAO group for the shape optimization of a complex shell structure with moving intersections. The class `ShapeOptGroupMint` requires two arguments. The first is an instance of `CPSurfDesign2Analysis` which provides the data and derivatives for the multilevel design of CAD geometry to reduce the dimension of the design space. The second argument is an instance of the non-matching problem `NonMatchingOpt`, which is the parent class of `NonMatchingOptFFD` without the FFD-related functions. Section 4.2 presents a numerical example using this group.

```
class ShapeOptGroupMint(om.Group):
    def initialize(self):
        self.options.declare('cpdesign2analysis')
        self.options.declare('nonmatching_opt')
    def init_parameters(self):
        self.des2ana = self.options['cpdesign2analysis']
        self.nm_opt = self.options['nonmatching_opt']
        self.opt_field = self.nm_opt.opt_field
        self.init_cp_design = self.des2ana.init_cp_design
        self.input_cp_shapes = [len(cp) for cp
                                in self.init_cp_design]
```

Next, we add the input component which takes control points of the coarse CAD geometry as the design variables. The order elevation and the knot refinement components are included to perform the k-refinement for the multilevel design, producing the fine analysis model.

```
    def setup(self):
        # Add inputs comp
        inputs_comp = om.IndepVarComp()
        for i, field in enumerate(self.opt_field):
            inputs_comp.add_output(
                VARNAME_CP_SURF_COARSE+str(field),
                shape=self.input_cp_shapes[i],
                val=self.init_cp_design[i])
        self.add_subsystem('input_comp', inputs_comp,
                           promotes=['*'])
        # Add order elevation comp
        self.cp_order_ele_comp = CPSurfOrderElevationComp(
            cpdesign2analysis=self.des2ana)
        self.cp_order_ele_comp.init_parameters()
        self.add_subsystem('CP_order_ele_comp',
            self.cp_order_ele_comp, promotes=['*'])
        # Add knot refinement comp
        self.cp_knot_refine_comp = CPSurfKnotRefinementComp(
            cpdesign2analysis=self.des2ana)
        self.cp_knot_refine_comp.init_parameters()
        self.add_subsystem('CP_knot_refine_comp',
            self.cp_knot_refine_comp, promotes=['*'])
```

Subsequently, the `CPIGA2XiComp` component is added to calculate the parametric coordinates of the intersections for a given set of surface control points by solving the implicit equation (29) and computing the partial derivatives. The displacement component for moving intersections `DispMintComp` is then added to evaluate the structural responses for updated surface control points and intersection locations. Similarly, the internal energy component is used to define the objective function.

```
        # Add CPIGA2Xi comp
        self.cpiga2xi_comp = CPIGA2XiComp(
            nonmatching_opt=self.nm_opt)
        self.cpiga2xi_comp.init_parameters()
        self.add_subsystem('CPIGA2xi_comp',
            self.cpiga2xi_comp, promotes=['*'])
        # Add displacement comp with moving int
        self.disp_states_comp = DispMintStatesComp(
            nonmatching_opt=self.nm_opt)
        self.disp_states_comp.init_parameters()
        self.add_subsystem('disp_comp',
            self.disp_states_comp, promotes=['*'])
        # Add internal energy comp (objective function)
        self.int_energy_comp = IntEnergyComp(
            nonmatching_opt=self.nm_opt)
        self.int_energy_comp.init_parameters()
        self.add_subsystem('int_energy_comp',
            self.int_energy_comp, promotes=['*'])
```

Lastly, we assign the coarse control points of the shell patches as design variables and designate the internal energy as the objective function to complete the problem setup.

```
        for i, field in enumerate(self.opt_field):
            self.add_design_var(
                VARNAME_CP_SURF_COARSE+str(field),
                lower=DESVAR_L[i], upper=DESVAR_U[i])
        self.add_objective(VARNAME_INT_ENERGY)
```

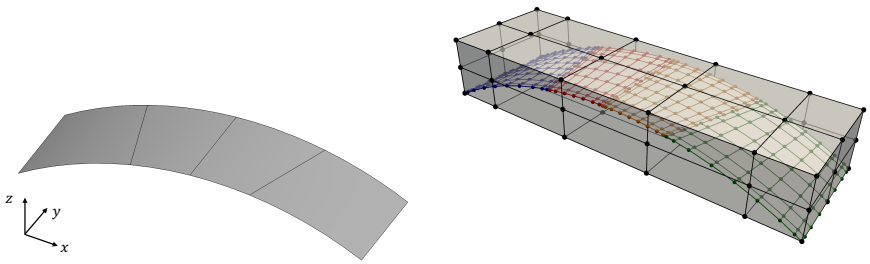
The preprocessing and setup of the non-matching problems, as well as the usage of the OpenMDAO groups mentioned above, are discussed in detail in Section 4.

4 Numerical examples

In this section, we present the implementation of benchmark problems to demonstrate the use of GOLDFISH and validate it with reference solutions. Then we showcase the application of GOLDFISH to the design optimization of aircraft wings.

4.1 Non-matching arch shape optimization

We use the arch shape optimization problem to verify the accuracy of the FFD-based shape update scheme in GOLDFISH. The benchmark problem was first proposed in [62, Section 8], where an arch geometry is subjected to a distributed downward load and fixed at both edges. The optimal shape of the arch with minimum internal energy is a quadratic parabola with an analytical height-to-length ratio so that the external load is entirely supported by membrane forces [62, Section 8]. To examine the capability of the FFD-based approach to preserve non-conforming surface intersections during shape updates, we create an arch geometry consisting of four non-conforming NURBS patches, as shown in Figure 7a. The arch geometry has a width of 3 m, a length of 10 m, and a shell thickness of 0.01 m. Young's modulus of 1000 GPa and Poisson's ratio of 0.0 are used for material properties. A 3D B-spline FFD block is generated to enclose the initial arch geometry, with the isogeometric discretization of both the FFD block and the arch geometry demonstrated in Figure 7b. The following listings provide essential implementation details for the FFD-based shape optimization using GOLDFISH.



(a) (b)

Fig. 7: (a) The initial design of an arch geometry consisting of four non-matching NURBS patches. (b) The initial arch geometry is embedded in a 3D B-spline block for FFD-based shape optimization.

We first define the material properties, coupling coefficient, and other basic parameters. Considering the three physical directions correspond to [0, 1, 2] in implementation since Python is a zero-based indexing language, we select control points in the z direction for optimization and define `opt_field` as [2].

```
E = Constant(1.0e12) # Young's modulus, Pa
nu = Constant(0.) # Poisson's ratio
h_th = Constant(0.01) # Shell thickness, m
pressure = Constant(1.) # Pressure magnitude, Pa
penalty_coefficient = 1.0e3 # Penalty coefficient
opt_field = [2] # Optimize z-coordinates only
ffd_block_nel = [4,1,1] # Nel of FFD block
p_ffd = 2 # Degree of the B-spline block
```

To perform the FFD-based shape optimization for the non-matching shells, we import the initial CAD geometry of the arch in IGES or STEP format into the running process using the Python interface of OpenCASCADE.

```
topo_shapes = read_igs_file("init_arch_geom.igs",
                            as_compound=False)
# Surface type conversion
occ_surf_list = [topoface2surface(face, BSpline=True)
                 for face in topo_shapes]
num_surfs = len(occ_surf_list)
```

We then create a geometry preprocessor instance to find all patch intersections and compute their parametric coordinates.

```
preproc = OCCPreprocessing(occ_surf_list)
preproc.compute_intersections(mortar_refine=2)
```

Next, we generate a list of tIGAr spline instances to build the extraction matrices. These matrices are utilized in the FFD-based shape update and IGA. The implementation of function `OCCBSpline2tIGArSpline`, which is standard to create a tIGAr spline instance from a given spline surface containing the knot vectors and control points, can be found in the GOLDFISH repository. Fixed boundary conditions are applied to the first and last shell patches and are implemented in this function. We omit these details here to focus on the setup of FFD-based shape optimization.

```
splines = []
for i in range(num_surfs):
    splines += [OCCBSpline2tIGArSpline(
                preproc.BSpline_surfs[i])]
```

The next step is to create the non-matching coupling instance for the list of tIGAr spline instances using `NonMatchingOptFFD`. This allows us to perform automated IGA and set up the FFD-based shape optimization problem.

```
nmopt_ffd = NonMatchingOptFFD(splines, E, h_th, nu)
nmopt_ffd.create_mortar_meshes(preproc.mortar_nels)
# Optimize z-coords for all shell patches
nmopt_ffd.set_shopt_surf_inds_FFD(opt_field, [0,1,2,3])
# Quadrature meshes setup for penalty energy
nmopt_ffd.mortar_meshes_setup(preproc.mapping_list,
          preproc.intersections_para_coords,
          penalty_coefficient)
```

A 3D B-spline FFD block is created by specifying the number of elements, degrees, and limits of the control points in the three directions using function `create_3D_block`.

```
# Create the 3D B-spline FFD block
cp_lims = nonmatching_opt_ffd.cpsurf_des_lims
for field in opt_field:
    cp_range = cp_lims[field][1]-cp_lims[field][0]
    cp_lims[field][1] = cp_lims[field][1]+0.2*cp_range
FFD_block = create_3D_block(ffd_block_nel, p_ffd, cp_lims)
```

By providing the knot vectors and control points of the FFD block to the non-matching coupling instance and arranging the related constraints on the FFD block control points, we complete the setup of the FFD block for shape optimization. The method `set_shopt_align_CPFFD` eliminates redundant design variables by aligning control points in the width direction. The `set_shopt_pin_CPFFD` method fixes control points on the lower edges of the FFD block, while `set_shopt_regu_CPFFD` ensures control points stay within the range of their adjacent neighbors, preventing unrealistic shapes.

```
# Set FFD block to the shell optimization problem
nmopt_ffd.set_shopt_FFD(FFD_block.knots, FFD_block.control)
# Set CP alignment in the width direction
nmopt_ffd.set_shopt_align_CPFFD(align_dir=[[1]])
# Set constraints to fix two lower edges of the block
nmopt_ffd.set_shopt_pin_CPFFD(pin_dir0=[2], pin_side0=[[0]],
                              pin_dir1=[1], pin_side1=[[0]])
# Set constraints to prevent self-penetration
nmopt_ffd.set_shopt_regu_CPFFD()
```

A list of the PDE residual forms based on the Kirchhoff–Love shell theory with St. Venant Kirchhoff material model is generated for all shell patches. These residual forms are expressed as FEniCS Unified Form Language (UFL) [47] Form objects, which allows for for automatic computation of symbolic derivatives. By inputting the residual forms into `nmopt_ffd`, the stiffness matrix $\mathbf{K}$ and residual vector $\mathbf{R}$ of the non-matching shell structure are assembled in PENGoLINS subroutines.

```
source_terms = []
residuals = []
for i in range(num_surfs):
    X = nmopt_ffd.splines[i].F # Shell geometry
    # Calculate curvilinear basis vectors
    A0,A1,A2,_,_,_ = surfaceGeometry(nmopt_ffd.splines[i], X)
    v_vec = as_vector([Constant(0.), Constant(0.),
                       Constant(1.)])
    # Constant downward distributed pressure
    force = as_vector([Constant(0.), Constant(0.),
                       -pressure*inner(v_vec, A2)])
    source_terms += [inner(force, nmopt_ffd.splines[i]\
        .rationalize(nmopt_ffd.spline_test_funcs[i]))\
        *nmopt_ffd.splines[i].dx]
    residuals += [SVK_residual(nmopt_ffd.splines[i],
                  nmopt_ffd.spline_funcs[i],
                  nmopt_ffd.spline_test_funcs[i],
                  E, nu, h_th, source_terms[i])]
nmopt_ffd.set_residuals(residuals)
```

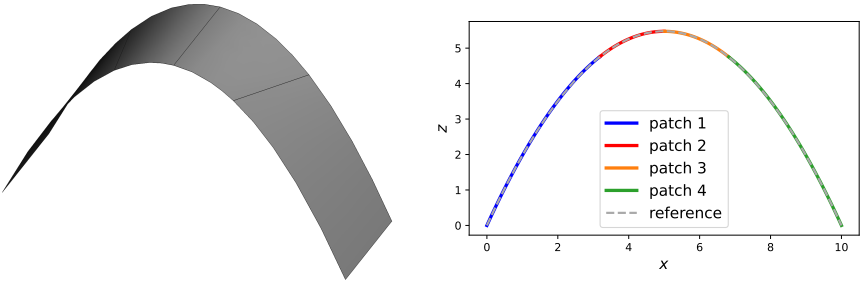
Subsequently, we construct the FFD-based shape optimization model using the component structure and implementation discussed in Section 2.3.1, and then create the OpenMDAO optimization problem.

```
# Create the FFD-based shape optimization model
model = ShapeOptGroupFFD(nonmatching_opt_ffd=nmopt_ffd)
model.init_parameters()
prob = om.Problem(model=model)
```

Finally, the SLSQP optimizer with a tolerance of 10^{-12} is selected for this problem, and the objective function is minimized using method `run_driver()`.

```
prob.driver = om.ScipyOptimizeDriver()
prob.driver.options['optimizer'] = 'SLSQP'
prob.driver.options['tol'] = 1e-12
prob.driver.options['maxiter'] = 1000
# Set up and run the optimization problem
prob.setup()
prob.run_driver()
```

The optimized arch geometry after 40 iterations is shown in Figure 8a. A comparison of the sliced view of the optimized arch with the analytical optimal shape is presented in Figure 8b, where the optimized geometry closely aligns with the reference solution from [62]. The height-to-length ratio of the optimized solution is 5.4748, which shows a negligible difference from the analytical value of 5.4779.



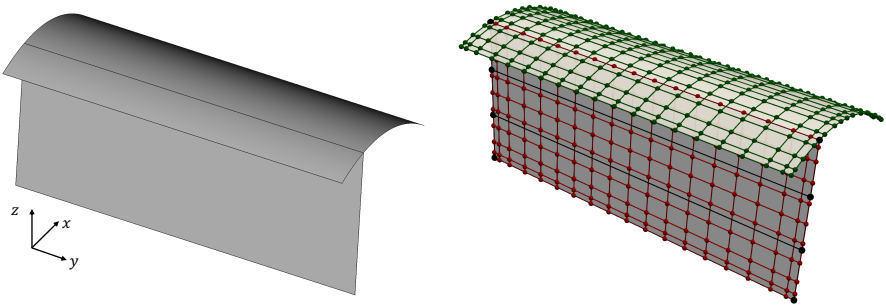
(a) (b)

Fig. 8: (a) The optimized design of the arch geometry. (b) The cross-sectional view of the optimized arch compared with the analytical optimum.

4.2 T-beam shape optimization with moving intersections

In this section, we use a T-beam geometry to demonstrate the use of GOLDFISH for shape optimization of multi-patch shell structures with moving intersections. The initial T-beam geometry is described by two B-spline surfaces. The top patch is a parabolic curved surface with dimensions of 2 m in width, 5 m in length, and spans from 0 m to 0.3 m in height. The vertical patch is a flat surface with dimensions of 2

m × 5 m and is positioned at one quarter of the top patch in the horizontal direction in the baseline design. The CAD geometry of the T-beam, shown in Figure 9a, is subjected to a distributed pressure in the downward vertical direction and is fixed at one end.



(a) (b)

Fig. 9: (a) The baseline design of the T-beam geometry consists of two B-spline patches. The top patch is a curved surface. (b) Isogeometric discretization of the initial T-beam, where the red and green control points represent DoFs in the analysis model, and black control points represent DoFs in the design model for shape optimization.

The optimal design with minimum internal energy is obtained when the vertical patch is positioned at the center of the top patch, maintaining a constant volume. In this example, we optimize the shape of the vertical patch while keeping the geometry of the top patch fixed. Meanwhile, the intersection between the two patches is allowed to move during the optimization process. Both shell patches have a thickness of 0.1 m, with material properties of Young's modulus $E = 10^7$ Pa and Possion's ratio $\nu = 0$. The discretization of the coarse design model for the vertical patch is indicated by black lines in Figure 9b, while the red and green points denote the discretization of the fine analysis model. The problem setup and GOLDFISH implementation are illustrated in the following code snippets.

The problem parameters definitions and CAD geometry import are similar to the previous example and will not be repeated. For the shape optimization, an instance of the geometry processor `OCCPreprocessing` and an instance of the non-matching problem `NonMatchingOpt` are created. We specify the fields of the control points for optimization as x and z coordinates, corresponding to `opt_field` as $[0, 2]$. Further, we specify the surface indices to be optimized in each field. For the vertical patch with an index of 1, the shape optimization surface indices are $[[1], [1]]$. We proceed to create and set up the quadrature meshes using `mortar_meshes_setup`, similar to the previous example. The key difference is that the argument `transfer_mat_deriv`, which defaults to 1, is set to 2 in this case since the partial derivative of the non-matching

residual with respect to intersection parametric coordinates requires second-order derivations of the spline basis functions.

```
opt_field = [0,2]
shopt_surf_inds = [[1], [1]]
nmopt.set_shopt_surf_inds(opt_field, shopt_surf_inds)
nmopt.set_geom_preprocessor(preproc)
nmopt.create_mortar_meshes(preproc.mortar_nels)
nmopt.mortar_meshes_setup(preproc.mapping_list,
      preproc.intersections_para_coords,
      penalty_coefficient, transfer_mat_deriv=2)
```

We use the `check_intersections_type` method to check the types of intersections. An intersection is treated as differentiable if it is not located at the edges of both intersecting spline patches. Next, the method `create_diff_intersections` is used to generate associated data for the differentiable intersections. The argument `num_edge_pts` is a list of integers, where each item specifies the number of points in the quadrature mesh used to enforce the T-junction. For this T-beam example, there is only one T-junction, and since the vertical patch is set to remain straight in the axial direction during optimization, a single point is sufficient to ensure the T-junction.

```
preproc.check_intersections_type()
preproc.get_diff_intersections()
nmopt.create_diff_intersections(num_edge_pts=[1])
```

We then use the geometry preprocessor to initialize an instance of `CPSurfDesign2Analysis` to establish the multilevel design framework between the design model and the analysis model.

```
des2ana = CPSurfDesign2Analysis(preproc, opt_field,
                                shopt_surf_inds)
```

We assume the imported CAD geometry represents the analysis model in the workflow. Therefore, we need to define the space for the design model. We first specify the order and knots of the design model. In this example, the horizontal location of the vertical patch is described by a cubic B-spline in the ξ_1 direction and a linear B-spline in the ξ_2 direction, while the vertical location is described by linear B-splines in both directions. All spline curves in the design model have a single knot span. The orders and knot vectors for the design model are defined as follows.

```
init_p_list = [[[3,1]], [[1,1]]]
init_knots_list = [[[[0, 0, 0, 0, 1, 1, 1, 1],
                     [0, 0, 1, 1]]],
                   [[[0, 0, 1, 1],
                     [0, 0, 1, 1]]]]
```

The orders and knot vectors for the model after order elevation are given by the following variables.

```
p_list_ele = [[[3, 3]], [[3, 3]]]
knots_list_ele = [[[[0, 0, 0, 0, 1, 1, 1, 1],
                    [0, 0, 0, 0, 1, 1, 1, 1]]],
                  [[[0, 0, 0, 0, 1, 1, 1, 1],
                    [0, 0, 0, 0, 1, 1, 1, 1]]]]
```

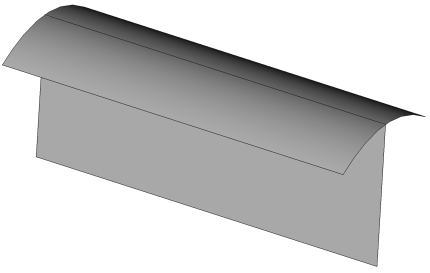
Next, we pass the multilevel design information to the instance `des2ana`. To keep the vertical patch straight in the axial direction, the horizontal and vertical coordinates are aligned along the axial direction using the `set_cp_align` method.

```
des2ana.set_init_knots_by_field(init_p_list,
                                init_knots_list)
des2ana.set_order_elevation_by_field(p_list_ele,
                                     knots_list_ele)
des2ana.set_knot_refinement()
des2ana.set_cp_align(field=0, align_dir_list=[1])
des2ana.set_cp_align(field=2, align_dir_list=[1])
```

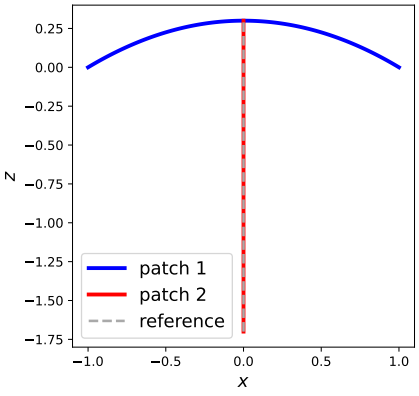
Furthermore, we can create the shape optimization model with moving intersections by passing the `nmopt` and `des2ana` instances to the `ShapeOptGroupMint` class and create the optimization problem.

```
model = ShapeOptGroupMint(cpdesign2analysis=des2ana,
                          nonmatching_opt=nmopt)
model.init_parameters()
prob = om.Problem(model=model)
```

Then the optimizer is configured using `prob.driver.options`. The optimization problem is set up with `prob.setup()` and solved by `prob.run_driver()`. Using the SLSQP optimizer with a tolerance of 10^{-9}, the optimized geometry after 22 iterations is shown in Figure 10a. A cross-sectional view of the T-beam is illustrated in Figure 10b, indicating that the vertical patch moves to the center of the top patch, thereby minimizing the internal energy of the T-beam for the given load and boundary conditions with sufficiently small errors. The optimized configuration in Figure 10 also demonstrates that the T-junction between the top and vertical patches is well preserved.



(a)

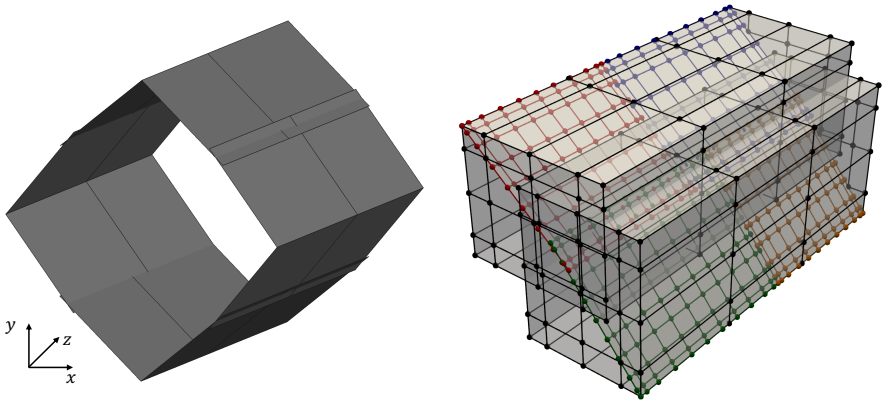


(b)

Fig. 10: (a) The optimized T-beam geometry with a curved top patch. (b) Cross-sectional view of the optimized T-beam, the vertical patch is moved to the center of the top patch and maintains the T-junction.

4.3 Tube under internal pressure

A tube under internal pressure is considered in this section. The initial design of the tube is shown in Figure 11a. To validate the GOLDFISH framework, we model a quarter of the initial tube using four non-matching B-spline patches, with three differentiable intersections between the upper and lower pair of patches. Each pair of shell patches is embedded in one FFD block to maintain their edge intersection, while the intersections between the two FFD blocks are allowed to move during the optimization process. This setup enables the analytical optimal solution, which is a cylindrical tube. The configuration of the FFD blocks and their associated discretizations are illustrated in Figure 11b. The optimization of this tube combines both the FFD-based approach and the moving intersections method.



(a) (b)

Fig. 11: (a) The baseline design of a tube geometry, with a quarter of the tube modeled by four non-matching B-spline patches. (b) Two FFD blocks are employed, one for each pair of B-spline surfaces with an edge intersection. Relative movement is allowed between the two FFD blocks.

The following code snippets illustrate the setup of the tube optimization problem using GOLDFISH. As in previous examples, we begin by creating instances of `OCCPreprocessing` and `NonmatchingOptFFD` for the imported CAD geometry. For both FFD blocks, we optimize the vertical and horizontal coordinates of their control points. Since the axial direction of the tube is the z direction, the optimization field is set as [0,1] for both FFD blocks to update their x and y coordinates. Shell patches with indices 0 and 1 are embedded in the first FFD block, while the remaining two patches are embedded in the second FFD block. Thus, the `opt_surf_inds` is defined as [[0,1], [2,3]]. This information is passed to the non-matching problem using the method `set_shopt_surf_inds_multiFFD`.

```
opt_field = [[0,1],[0,1]]
opt_surf_inds = [[0,1], [2,3]]
```

```
nm_opt.set_shopt_surf_inds_multiFFD(opt_field, opt_surf_inds)
nm_opt.set_geom_preprocessor(preproc)
```

We then create two lists containing the knot vectors `shopt_ffd_knots_list` and control points `shopt_ffd_knots_list` for the trivariate B-spline solids used to define the FFD blocks, as demonstrated in Figure 11b. The definitions of these lists are omitted for clarity. The method `set_shopt_multiFFD` is used to obtain the data of the FFD blocks.

```
nm_opt.set_shopt_multiFFD(shopt_ffd_knots_list,
                          shopt_ffd_control_list)
```

Next, the control points of both FFD blocks are aligned in the axial direction using the method `set_shopt_align_CP_multiFFD`. The control points on the faces of the FFD blocks that lie on the symmetric planes are fixed using `set_shopt_pin_CP_multiFFD`. Additionally, the method `set_shopt_regu_CP_multiFFD` is employed to prevent self-penetration of the FFD blocks.

```
nm_opt.set_shopt_align_CP_multiFFD(ffd_ind=0,
                                   align_dir=[[2],[2]])
nm_opt.set_shopt_align_CP_multiFFD(ffd_ind=1,
                                   align_dir=[[2],[2]])
nm_opt.set_shopt_pin_CP_multiFFD(ffd_ind=0, pin_dir0=[0,0],
                                 pin_side0=[[0],[0]])
nm_opt.set_shopt_pin_CP_multiFFD(ffd_ind=1, pin_dir0=[1,1],
                                 pin_side0=[[0],[0]])
nm_opt.set_shopt_regu_CP_multiFFD()
```

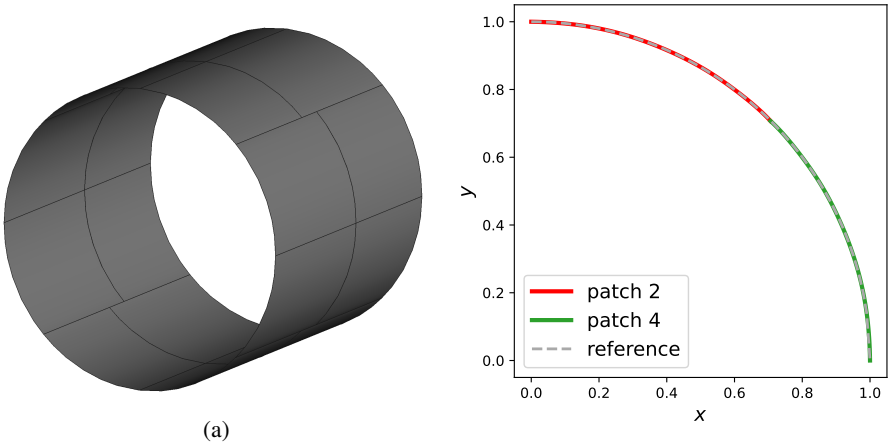
The remainder of the optimization setup is identical to the previous examples, which involves creating quadrature meshes for the intersections and defining PDE residuals for the Kirchhoff–Love shells. The OpenMDAO optimization problem is then created using `om.Problem`. The SNOPT optimizer is employed for this problem with a tolerance of 10^{-2}. The converged tube geometry after 142 iterations is shown in Figure 12a. A cross-sectional view of a quarter of the tube is displayed in Figure 12b and compared with an exact quarter circle for validation. The surface intersections between the two pairs of shell patches move to the edges of the patches, forming a cylindrical tube to achieve the optimal shape. This demonstrates that both the FFD-based and moving intersections approaches can work simultaneously in shape optimization for non-matching shell structures.

4.4 Application to aircraft wings

This section presents the optimization results for aircraft wings to demonstrate the applicability of GOLDFISH to complex shell structures.

4.4.1 Wing shape optimization using FFD-based approach

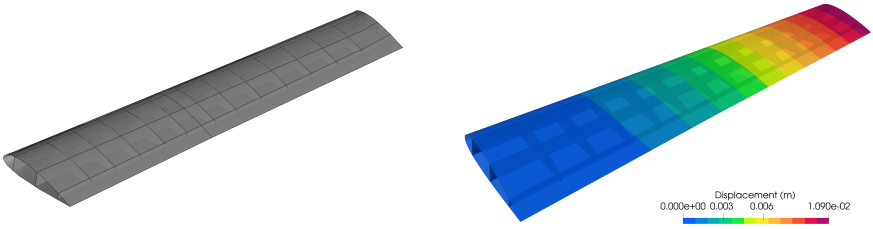
In the first application, we use the FFD-based approach to optimize the shape of an electric vertical take-off and landing (eVTOL) aircraft wing. The CAD geometry of the eVTOL wing, shown in Figure 13a, is composed of 21 B-spline patches and 87



(a) (b)

Fig. 12: (a) The resulting geometry of the tube with minimum internal energy. Surface intersections between the spline patches transit to edge intersections in the optimized design. (b) Comparison of the optimized geometry with an exact cylindrical tube in the cross-sectional view.

intersections. The wing is clamped at the root and subjected to an upward-facing distributed pressure. The material properties used are those of aluminum, with Young's modulus of 68 GPa and Poission's ratio of 0.35. All shell patches have a thickness of 3 mm. The contour plot of the wing displacement in the baseline design is shown in Figure 13b.

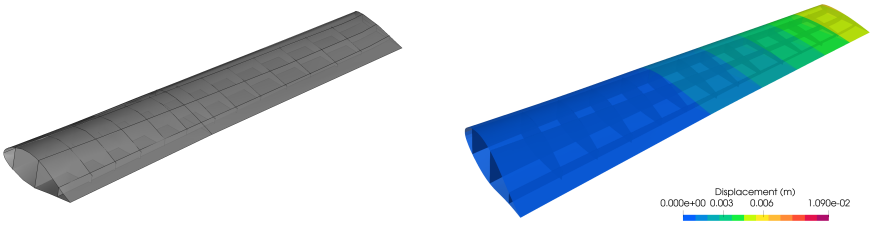


(a) (b)

Fig. 13: (a) The initial CAD geometry of an eVTOL wing. (b) The displacement field of the eVTOL wing under a distributed load.

To perform shape optimization for the eVTOL wing, the entire geometry is embedded in an FFD B-spline block for shape update. The control points of the FFD block in the vertical direction serve as the design variables. The internal energy of the wing is minimized under a constant volume constraint. A regularization term is

added to the objective function to smooth the gradient changes of the surface in the vertical direction, preventing oscillatory shapes. The optimized geometry is shown in Figure 14a. The cross-section of the wing root becomes wider to provide greater support under the distributed pressure, while the wing tip narrows in the vertical direction to compensate for the increased volume at the wing root. The corresponding displacement contour plot of the optimized wing is displayed in Figure 14b, where the maximum displacement magnitude is noticeably decreased. The internal energy of the optimized wing is reduced by 49.1% compared to the baseline design.



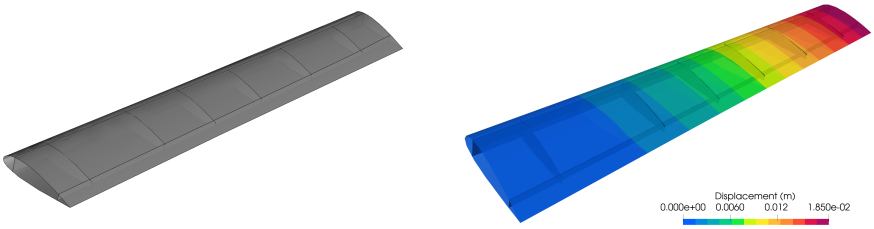
(a) (b)

Fig. 14: (a) The optimized CAD geometry of the eVTOL wing using the FFD-based approach. (b) The displacement field of the optimized eVTOL wing.

4.4.2 Internal structures shape optimization

This section focuses on the optimization of the layout of internal structures of the eVOTL wing, where intersections between outer skins, spars, and ribs are allowed to move during the optimization process. The initial geometry of the eVTOL wing is shown in Figure 15a, consisting of 11 B-spline patches and 32 intersections. Among these intersections, four of them are located on the edges of intersecting outer surfaces and therefore remain fixed. The remaining 28 intersections are movable to optimize the layout of the internal sub-structures. The boundary conditions, loading conditions, material properties, and geometric parameters are the same as those in Section 4.4.1, and the displacement contour plot is shown in Figure 15b.

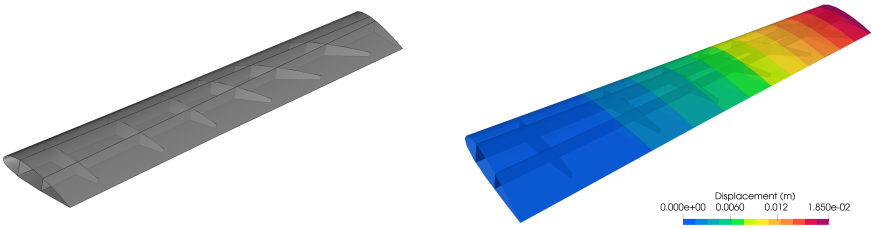
In this application, the outer surfaces remain unchanged during the shape optimization. Internal spars are allowed to have rigid body translation within the envelope of the outer surfaces, while the ribs can translate and rotate along the lines from 15% to 80% of the distance from the leading edge to the trailing edge but remain planar surfaces. Each rib has a volume constraint set to less than 1.5 times its initial value to prevent excessively stretched elements. The optimized geometry, which minimizes internal energy, is shown in Figure 16a, and the associated displacement field is visualized in Figure 16b. The tip displacement is slightly decreased as the contour lines move toward the wing tip. The internal energy of the optimized wing is reduced by



(a) (b)

Fig. 15: (a) The baseline design of an eVTOL wing. (b) The displacement of the initial eVTOL wing.

5.2% compared to the initial design. Additionally, the T-junctions between the internal structures and outer skins are maintained, despite the changes in the locations of internal structures.



(a) (b)

Fig. 16: (a) The CAD geometry of the eVTOL wing with optimized internal ribs and spars. (b) The displacement field of the optimized eVTOL wing.

The eVTOL wing shape optimization examples validate the applicability of GOLDFISH to complex shell structures with arbitrary intersections. The overall shape optimization exhibits a significant reduction in internal energy. Furthermore, the optimization of internal spars and ribs not only reduces the objective function but also leads to innovative layouts of internal structures. This section showcases two representative design scenarios for the eVTOL wing, further examples of wing design can be found in [31, Section 6] using the FFD-based approach and in [32, Section 6] with moving intersections.

5 Conclusions

This paper introduces GOLDFISH, an open-source Python library for the shape optimization of complex shell structures. The structural analysis employs an isogeometric

Kirchhoff–Love shell model coupled with a penalty formulation for patch intersections, and control points of shell patches are modified to update the shell shape. In this approach, FE mesh generation is no longer required in the optimization loop. This framework is developed based on FEniCS, leveraging its automatic differentiation and code generation capabilities to compute derivatives, thereby enabling gradient-based design optimization. The code framework accepts B-spline or NURBS-based CAD geometries as input and returns the optimized geometry in the same format, streamlining the workflow from geometry design through structural analysis to shape optimization.

The modular design of the framework inherited from OpenMDAO is presented along with the essential components for FFD-based shape optimization and the moving intersections approaches. A suite of benchmark problems, accompanied by code implementations, is provided to examine the effectiveness of the framework. Furthermore, the applications to aerospace structures are demonstrated, resulting in innovative designs for the layout of internal spars and ribs in an eVTOL wing. Nevertheless, conditions from other disciplines, such as aerodynamics [70, 71], electric motors [72], and noise control [73], need to be considered for practical designs. This presents opportunities for future work on integrating various disciplines to enhance the framework for more practical shell structure designs. The code framework is maintained on the GitHub repository [74].

Acknowledgments

Han Zhao was supported by NASA grant number 80NSSC21M0070 when preparing the original submission. We thank Dr. David Kamensky at the University of California San Diego for helpful discussions on FEniCS implementation.

References

[1] Bletzinger, K.-U., Ramm, E.: Form finding of shells by structural optimization. Engineering with computers **9**, 27–35 (1993)

[2] Hardwick, M.F., Clay, R.L., Boggs, P.T., Walsh, E.J., Larzelere, A.R., Altshuler, A.: DART system analysis. Technical Report SAND2005-4647, Sandia National Laboratories (2005)

[3] Hughes, T.J.R., Cottrell, J.A., Bazilevs, Y.: Isogeometric analysis: CAD, finite elements, NURBS, exact geometry, and mesh refinement. Computer Methods in Applied Mechanics and Engineering **194**, 4135–4195 (2005)

[4] Cottrell, J.A., Hughes, T.J.R., Bazilevs, Y.: Isogeometric Analysis: Toward Integration of CAD and FEA. Wiley, Chichester (2009)

[5] Piegl, L., Tiller, W.: The NURBS Book. Springer, Berlin, Heidelberg (2012)

[6] Sederberg, T.W., Zheng, J., Bakenov, A., Nasri, A.: T-splines and T-NURCCs. ACM transactions on graphics (TOG) **22**(3), 477–484 (2003)

[7] Giannelli, C., Jüttler, B., Speleers, H.: THB-splines: The truncated basis for hierarchical splines. Computer Aided Geometric Design **29**(7), 485–498 (2012)

[8] Thomas, D.C., Engvall, L., Schmidt, S.K., Tew, K., Scott, M.A.: U-splines: Splines over unstructured meshes. Computer Methods in Applied Mechanics and Engineering **401**, 115515 (2022)

[9] Evans, J.A., Bazilevs, Y., Babuška, I., Hughes, T.J.R.: *n*-widths, sup–infs, and optimality ratios for the *k*-version of the isogeometric finite element method. Computer Methods in Applied Mechanics and Engineering **198**(21), 1726–1741 (2009). Advances in Simulation-Based Engineering Sciences – Honoring J. Tinsley Oden

[10] Kiendl, J., Bletzinger, K.-U., Linhard, J., Wüchner, R.: Isogeometric shell analysis with Kirchhoff–Love elements. Computer Methods in Applied Mechanics and Engineering **198**(49-52), 3902–3914 (2009)

[11] Kiendl, J.: Isogeometric analysis and shape optimal design of shell structures. PhD thesis, Lehrstuhl für Statik, Technische Universität München (2011)

[12] Kiendl, J., Bazilevs, Y., Hsu, M.-C., Wüchner, R., Bletzinger, K.-U.: The bending strip method for isogeometric analysis of Kirchhoff–Love shell structures comprised of multiple patches. Computer Methods in Applied Mechanics and Engineering **199**, 2403–2416 (2010)

[13] Guo, Y., Ruess, M.: Nitsche's method for a coupling of isogeometric thin shells and blended shell structures. Computer Methods in Applied Mechanics and Engineering **284**, 881–905 (2015)

[14] Guo, Y., Heller, J., Hughes, T.J.R., Ruess, M., Schillinger, D.: Variationally consistent isogeometric analysis of trimmed thin shells at finite deformations, based on the STEP exchange format. Computer Methods in Applied Mechanics and Engineering **336**, 39–79 (2018)

[15] Guo, Y., Zou, Z., Ruess, M.: Isogeometric multi-patch analyses for mixed thin shells in the framework of non-linear elasticity. Computer Methods in Applied Mechanics and Engineering **380**, 113771 (2021)

[16] Benzaken, J., Evans, J.A., McCormick, S.F., Tamstorf, R.: Nitsche's method for linear Kirchhoff–Love shells: Formulation, error analysis, and verification. Computer Methods in Applied Mechanics and Engineering **374**, 113544 (2021)

[17] Wang, Y., Yu, Y., Lin, Y.: Isogeometric analysis with embedded stiffened shells for the hull structural mechanical analysis. Journal of Marine Science and

Technology **27**(1), 786–805 (2022)

[18] Brivadis, E., Buffa, A., Wohlmuth, B., Wunderlich, L.: Isogeometric mortar methods. Computer Methods in Applied Mechanics and Engineering **284**, 292–319 (2015). https://doi.org/10.1016/j.cma.2014.09.012. Isogeometric Analysis Special Issue

[19] Horger, T., Reali, A., Wohlmuth, B., Wunderlich, L.: A hybrid isogeometric approach on multi-patches with applications to Kirchhoff plates and eigenvalue problems. Computer Methods in Applied Mechanics and Engineering **348**, 396–408 (2019)

[20] Hirschler, T., Bouclier, R., Dureisseix, D., Duval, A., Elguedj, T., Morlier, J.: A dual domain decomposition algorithm for the analysis of non-conforming isogeometric Kirchhoff–Love shells. Computer Methods in Applied Mechanics and Engineering **357**, 112578 (2019)

[21] Herrema, A.J., Johnson, E.L., Proserpio, D., Wu, M.C.H., Kiendl, J., Hsu, M.-C.: Penalty coupling of non-matching isogeometric Kirchhoff–Love shell patches with application to composite wind turbine blades. Computer Methods in Applied Mechanics and Engineering **346**, 810–840 (2019)

[22] Leonetti, L., Liguori, F.S., Magisano, D., Kiendl, J., Reali, A., Garcea, G.: A robust penalty coupling of non-matching isogeometric Kirchhoff–Love shell patches in large deformations. Computer Methods in Applied Mechanics and Engineering **371**, 113289 (2020)

[23] Proserpio, D., Kiendl, J.: Penalty coupling of trimmed isogeometric Kirchhoff–Love shell patches. Journal of Mechanics **38**, 156–165 (2022)

[24] Zhao, H., Liu, X., Fletcher, A.H., Xiang, R., Hwang, J.T., Kamensky, D.: An open-source framework for coupling non-matching isogeometric shells with application to aerospace structures. Computers & Mathematics with Applications **111**, 109–123 (2022)

[25] Guarino, G., Antolin, P., Milazzo, A., Buffa, A.: An interior penalty coupling strategy for isogeometric non-conformal Kirchhoff–Love shell patches. Engineering With Computers, 1–27 (2024)

[26] Hirschler, T., Bouclier, R., Duval, A., Elguedj, T., Morlier, J.: The embedded isogeometric Kirchhoff–Love shell: From design to shape optimization of non-conforming stiffened multipatch structures. Computer Methods in Applied Mechanics and Engineering **349**, 774–797 (2019)

[27] Hirschler, T.: Isogeometric modeling for the optimal design of aerostructures. PhD thesis, Université de Lyon (2019)

[28] Bouclier, R., Hirschler, T.: IGA: Non-conforming Coupling and Shape Optimization of Complex Multipatch Structures. John Wiley & Sons, Hoboken (2022)

[29] Hao, P., Wang, Y., Jin, L., Ma, S., Wang, B.: An isogeometric design-analysis-optimization workflow of stiffened thin-walled structures via multi-level NURBS-based free-form deformations (MNFFD). Computer Methods in Applied Mechanics and Engineering **408**, 115936 (2023)

[30] Sederberg, T.W., Parry, S.R.: Free-form deformation of solid geometric models. SIGGRAPH Comput. Graph. **20**(4), 151–160 (1986). https://doi.org/10.1145/15886.15903

[31] Zhao, H., Kamensky, D., Hwang, J.T., Chen, J.S.: Automated shape and thickness optimization for non-matching isogeometric shells using free-form deformation. Engineering with Computers, 1–24 (2024)

[32] Zhao, H., Hwang, J.T., Chen, J.S.: Shape optimization of non-matching isogeometric shells with moving intersections. Computer Methods in Applied Mechanics and Engineering **431**, 117322 (2024). https://doi.org/10.1016/j.cma.2024.117322

[33] Gray, J.S., Hwang, J.T., Martins, J.R.R.A., Moore, K.T., Naylor, B.A.: OpenMDAO: An open-source framework for multidisciplinary design, analysis, and optimization. Structural and Multidisciplinary Optimization **59**, 1075–1104 (2019)

[34] Jasa, J.P., Hwang, J.T., Martins, J.R.R.A.: Open-source coupled aerostructural optimization using Python. Structural and Multidisciplinary Optimization **57**(4), 1815–1827 (2018). https://doi.org/10.1007/s00158-018-1912-8

[35] Jasa, J.P., Brelje, B.J., Gray, J.S., Mader, C.A., Martins, J.R.R.A.: Large-scale path-dependent optimization of supersonic aircraft. Aerospace (7), 10 (2020). https://doi.org/10.3390/aerospace7100152

[36] Adler, E.J., Brelje, B.J., Martins, J.R.R.A.: Thermal management system optimization for a parallel hybrid aircraft considering mission fuel burn. Aerospace **9**(5) (2022). https://doi.org/10.3390/aerospace9050243

[37] Herrema, A.J., Kiendl, J., Hsu, M.-C.: A framework for isogeometric-analysis-based optimization of wind turbine blade structures. Wind Energy **22**(2), 153–170 (2019)

[38] Yan, J., Li, N., Luo, T., Tolley, M.T., Hwang, J.T.: Optimal control and design of an underactuated ball-pitching robotic arm using large-scale multidisciplinary optimization. In: AIAA Aviation 2019 Forum (2019). https://doi.org/10.2514/6.2019-3450

[39] Lin, J.-T., Girerd, C., Yan, J., Hwang, J.T., Morimoto, T.K.: A generalized framework for concentric tube robot design using gradient-based optimization. IEEE Transactions on Robotics **38**(6), 3774–3791 (2022)

[40] Chung, H., Hwang, J.T., Gray, J.S., Kim, H.A.: Topology optimization in OpenMDAO. Structural and multidisciplinary optimization **59**, 1385–1400 (2019)

[41] Yan, J., Xiang, R., Kamensky, D., Tolley, M.T., Hwang, J.T.: Topology optimization with automated derivative computation for multidisciplinary design problems. Structural and Multidisciplinary Optimization **65**(5), 151 (2022)

[42] Jauregui, C.M., Schnulo, S.L., Gray, J.S., Kim, H.A.: Level set topology optimization in OpenMDAO. In: AIAA SCITECH 2023 Forum (2023). https://doi.org/10.2514/6.2023-2548

[43] Gandarillas, V., Joshy, A.J., Sperry, M.Z., Ivanov, A.K., Hwang, J.T.: A graph-based methodology for constructing computational models that automates adjoint-based sensitivity analysis. Structural and Multidisciplinary Optimization **67**(5), 76 (2024)

[44] Xiang, R., van Schie, S.P.C., Scotzniovsky, L., Yan, J., Kamensky, D., Hwang, J.T.: Automating adjoint sensitivity analysis for multidisciplinary models involving partial differential equations. Structural and Multidisciplinary Optimization **67**, 1–31 (2024)

[45] Logg, A., Wells, G.N.: DOLFIN: Automated finite element computing. ACM Trans. Math. Softw. **37**(2), 20–12028 (2010)

[46] Logg, A., Mardal, K.-A., Wells, G.N.: Automated Solution of Differential Equations by the Finite Element Method. Springer, Heidelberg (2012)

[47] Alnæs, M.S., Logg, A., Ølgaard, K.B., Rognes, M.E., Wells, G.N.: Unified form language: A domain-specific language for weak formulations of partial differential equations. ACM Trans. Math. Softw. **40**(2), 9–1937 (2014)

[48] Kamensky, D., Bazilevs, Y.: tIGAr: Automating isogeometric analysis with FEniCS. Computer Methods in Applied Mechanics and Engineering **344**, 477–498 (2019). https://doi.org/10.1016/j.cma.2018.10.002

[49] Borden, M.J., Scott, M.A., Evans, J.A., Hughes, T.J.R.: Isogeometric finite element data structures based on Bézier extraction of NURBS. International Journal for Numerical Methods in Engineering **87**, 15–47 (2011)

[50] Scott, M.A., Borden, M.J., Verhoosel, C.V., Sederberg, T.W., Hughes, T.J.R.:

Isogeometric finite element data structures based on Bézier extraction of T-splines. International Journal for Numerical Methods in Engineering **88**, 126–156 (2011)

[51] Schillinger, D., Ruthala, P.K., Nguyen, L.H.: Lagrange extraction and projection for NURBS basis functions: A direct link between isogeometric and standard nodal finite element formulations. International Journal for Numerical Methods in Engineering **108**(6), 515–534 (2016). https://doi.org/10.1002/nme.5216

[52] Fromm, J.E., Wunsch, N., Xiang, R., Zhao, H., Maute, K., Evans, J.A., Kamensky, D.: Interpolation-based immersed finite element and isogeometric analysis. Computer Methods in Applied Mechanics and Engineering **405**, 115890 (2023)

[53] Fromm, J.E., Wunsch, N., Maute, K., Evans, J.A., Chen, J.-S.: Interpolation-based immersogeometric analysis methods for multi-material and multi-physics problems. Computational Mechanics, 1–25 (2024)

[54] Bazilevs, Y., Kamensky, D., Moutsanidis, G., Shende, S.: Residual-based shock capturing in solids. Computer Methods in Applied Mechanics and Engineering **358**, 112638 (2020)

[55] ten Eikelder, M.F.P., Bazilevs, Y., Akkerman, I.: A theoretical framework for discontinuity capturing: Joining variational multiscale analysis and variation entropy theory. Computer Methods in Applied Mechanics and Engineering, 112664 (2019)

[56] Zhang, W., Bui-Thanh, T., Sacks, M.S.: A machine learning approach for soft tissue remodeling. In: Proceedings of FEniCS'19 (2019)

[57] Yang, H., Abali, B.E., Timofeev, D., Müller, W.H.: Determination of metamaterial parameters by means of a homogenization approach based on asymptotic analysis. Continuum mechanics and thermodynamics **32**, 1251–1270 (2020)

[58] Kamensky, D.: Open-source immersogeometric analysis of fluid–structure interaction using FEniCS and tIGAr. Computers & Mathematics with Applications **81**, 634–648 (2021). Development and Application of Open-source Software for Problems with Numerical PDEs

[59] Neighbor, G.E., Zhao, H., Saraeian, M., Hsu, M.-C., Kamensky, D.: Leveraging code generation for transparent immersogeometric fluid–structure interaction analysis on deforming domains. Engineering with Computers **39**(2), 1019–1040 (2023)

[60] Kiendl, J., Hsu, M.-C., Wu, M.C.H., Reali, A.: Isogeometric Kirchhoff–Love shell formulations for general hyperelastic materials. Computer Methods in Applied Mechanics and Engineering **291**(0), 280–303 (2015). https://doi.org/

10.1016/j.cma.2015.03.010

[61] Kamensky, D., Hsu, M.-C., Schillinger, D., Evans, J.A., Aggarwal, A., Bazilevs, Y., Sacks, M.S., Hughes, T.J.R.: An immersogeometric variational framework for fluid–structure interaction: Application to bioprosthetic heart valves. Computer Methods in Applied Mechanics and Engineering **284**, 1005–1053 (2015)

[62] Kiendl, J., Schmidt, R., Wüchner, R., Bletzinger, K.-U.: Isogeometric shape optimization of shells using semi-analytical sensitivity analysis and sensitivity weighting. Computer Methods in Applied Mechanics and Engineering **274**, 148–167 (2014)

[63] Hirschler, T., Bouclier, R., Duval, A., Elguedj, T., Morlier, J.: A new lighting on analytical discrete sensitivities in the context of isogeometric shape optimization. Archives of Computational Methods in Engineering **28**(4), 2371–2408 (2021)

[64] Ahmadianfar, I., Bozorg-Haddad, O., Chu, X.: Gradient-based optimizer: A new metaheuristic optimization algorithm. Information Sciences **540**, 131–159 (2020)

[65] Paviot, T., Feringa, J.: Pythonocc. Technical report, 3D CAD/CAE/PLM development framework for the Python programming language (2018)

[66] Kraft, D.: A software package for sequential quadratic programming. Forschungsbericht- Deutsche Forschungs- und Versuchsanstalt fur Luft- und Raumfahrt (1988)

[67] Gill, P.E., Murray, W., Saunders, M.A.: SNOPT: An SQP algorithm for large-scale constrained optimization. SIAM review **47**(1), 99–131 (2005)

[68] Nagy, A.P., Abdalla, M.M., Gürdal, Z.: Isogeometric sizing and shape optimisation of beam structures. Computer Methods in Applied Mechanics and Engineering **199**(17-20), 1216–1230 (2010)

[69] Nagy, A.P., IJsselmuiden, S.T., Abdalla, M.M.: Isogeometric design of anisotropic shells: optimal form and material distribution. Computer Methods in Applied Mechanics and Engineering **264**, 145–162 (2013)

[70] van Schie, S.P., Zhao, H., Yan, J., Xiang, R., Hwang, J.T., Kamensky, D.: Solver-independent aeroelastic coupling for large-scale multidisciplinary design optimization. In: AIAA Scitech 2023 Forum, p. 0727 (2023)

[71] van Schie, S.P., Warner, M., Fletcher, A., Hwang, J.T.: Enforcing work conservation in modular aeroelastic coupling for multidisciplinary design optimization. In: AIAA SCITECH 2024 Forum, p. 2411 (2024)

[72] Scotzniovsky, L., Xiang, R., Cheng, Z., Rodriguez, G., Kamensky, D., Mi, C., Hwang, J.T.: Geometric design of electric motors using adjoint-based shape optimization. Optimization and Engineering, 1–38 (2024)

[73] Gill, H., Lee, S., Ruh, M.L., Hwang, J.T.: Applicability of low-fidelity tonal and broadband noise models on small-scaled rotors. In: AIAA SCITECH 2023 Forum, p. 1547 (2023)

[74] https://github.com/hanzhao2020/GOLDFISH: GOLDFISH source code