Author version

Published as:

Xiang, R., van Schie, S. P. C., Scotzniovsky, L., Yan, J., Kamensky, D., & Hwang, J. T. (2024). Automating adjoint sensitivity analysis for multidisciplinary models involving partial differential equations. Structural and Multidisciplinary Optimization. Advance online publication. https://doi.org/10.21203/rs.3.rs-4265983/v1

BibTeX for this reference:

https://lsdolab.github.io/lsdo-publications/publication/xiang2024automating/

```
@article{xiang2024automating,
  author = {Ru Xiang and Sebastiaan P. C. van Schie and Luca Scotzniovsky and Jiayao
    Yan and David Kamensky and John T. Hwang},
  title = {Automating adjoint sensitivity analysis for multidisciplinary models
    involving partial differential equations},
  journal = {Structural and Multidisciplinary Optimization},
  year = {2024},
  doi = {10.21203/rs.3.rs-4265983/v1},
  url = {https://doi.org/10.21203/rs.3.rs-4265983/v1}
}
```

Noname manuscript No.
(will be inserted by the editor)

Automating adjoint sensitivity analysis for multidisciplinary models involving partial differential equations

Ru Xiang[1] · **Sebastiaan P. C. van Schie**[1] · **Luca Scotzniovsky**[1] · **Jiayao Yan**[1] · **David Kamensky**[1] · **John T. Hwang**[1]

Received: date / Accepted: date

Abstract We present a general framework for incorporating partial differential equation (PDE)-based models into gradient-based optimization of multidisciplinary systems by integrating FEniCSx with the recently-developed Computational System Design Language (CSDL). CSDL is a domain specific language designed to facilitate adjoint sensitivity analysis multidisciplinary design optimization (MDO). We use CSDL's abstractions to link together sub-models representing different disciplines, not all of which are necessarily modeled by PDEs. For the subsystems which are modeled by PDEs, we use FEniCSx to compute partial derivatives of problem residuals, which CSDL can combine with derivatives from other disciplines using the chain rule and the adjoint method.

The development of this framework is motivated by the problem of optimizing designs of electric vertical takeoff and landing (eVTOL) aircraft where, due to the relative novelty of this class of vehicle, there is currently a large, unexplored design space. Predicting the performance of eVTOL aircraft requires PDE-based modeling of various sub-problems, including electric motors, composite shell structures, and thermoelasticity and electrochemistry of battery packs. For system-level analysis and optimization, these must be coupled to non-PDE components, such as low-fidelity aerodynamic models, geometric design tools, and lumped-parameter battery and circuit models. In this work, we demonstrate the modeling flexibility and efficiency of this framework through classic optimal control and topology optimization problems with known solutions, and also challenging eVTOL-related applications including shape optimization of an electric motor, and aeroelastic coupling for gust response of an eVTOL wing. Given the generality of this framework, we expect it to facilitate research on a wide-range of PDE-constrained MDO problems beyond eVTOL applications.

Keywords Finite element method · Gradient-based optimization · Multidisciplinary design optimization · Aeroelasticity · Electric motor design

1 Introduction

The design and optimization of electrical vertical takeoff and landing (eVTOL) aircraft poses significant challenges due to the complexity in modeling and analyzing multiphysics systems. This includes predicting the stability and performance of eVTOL aircraft under different flight conditions, which requires evaluating a system-level model that integrates sub-models from various disciplines, such as weights, battery design, flight dynamics, and aeroelasticity. In order to achieve a reasonable turnaround time while maintaining sufficient accuracy for global evaluation, the system-level model may contain both high-fidelity PDE-based sub-models and low-fidelity non-PDE sub-models, highlighting the need for a multifidelity computational tool.

Ru Xiang
E-mail: rxiang@ucsd.edu
[1] Mechanical and Aerospace Engineering, University of California San Diego, CA 92093, USA

Multidisciplinary design optimization (MDO) is a widely used approach in this kind of problem, as it can incorporate all of the disciplines simultaneously. In particular, gradient-based MDO is preferable in this case, as evaluating high-fidelity sub-models can be extremely expensive in terms of computational cost.

To numerically solve the PDE components in gradient-based MDO problems, we can utilize the finite element method (FEM) to approximate the desired PDE solution in a finite-dimensional function space over a discretized domain. While the finite element method is widely used in structural analysis, it is also applicable to problems in any discipline that can be described by PDEs, such as heat transfer simulations of battery packs [1], fluid–structure interaction of blood vessels [2], or even weather predictions [3].

However, existing tools for finite element analysis and MDO are not naturally compatible with each other and do not fulfill all the requirements mentioned above. Commercial finite element tools, such as NASTRAN, may come with fully-integrated optimization modules, like NASTRAN SOL200 [4]. However, it is challenging to export essential gradient information from a closed-source commercial solver for coupling with other sub-models, such as a low-fidelity particle-based aerodynamic solver built on purely algebraic equations.

Open-source finite element tools are easier to extract derivative information from, but not general enough to suit all disciplines involving PDE models. For example, while TACS [5] specializes in structural optimization with beams and shells, it is challenging to be extended to multidisciplinary problems like aeroelasticity, as it can only provide finite element solutions for existing disciplines implemented in its C++ library, which currently supports structures and heat transfer.

FEniCS [6] is an open-source tool for solving PDEs that automates the efficient implementation of the finite element method through its abstraction of mathematical modeling and its automatic code generation capability. It uses the Python interface of the Unified Form Language (UFL) [7] to provide a user-friendly front end, where users only need to provide the mathematical model as a variational form of the PDE residual, and FEniCS will call the FEniCS Form Compiler (FFC) [8] to generate finite element code in C++ and solve the problem in the backend of DOLFIN [9].

The library dolfin-adjoint [10] is an adjoint-based optimization extension of FEniCS that minimizes the effort required to build an adjoint model based on a forward solver. However, due to its dependency on FEniCS, it requires that all sub-models be implemented in the Python interface of FEniCS or Firedrake [11], limiting the use of non-PDE components like models represented by algebraic equations for preprocessing and postprocessing steps.

CSDL [12] is a recently developed algebraic modeling language that automates the exact derivative computation for its native operations including array operations such as reshaping, reordering, and matrix-vector products, and fundamental operations such as logarithmic, exponential, and trigonometric functions. CSDL is designed to be compatible with external solvers using customized operations, requiring users to define the operations and the associated partial derivatives in the format that CSDL specifies. This allows CSDL to connect the external solvers with the rest of the model automatically and compute the total derivatives needed for the optimization using the chain rule and the adjoint method.

The aforementioned FEniCS is a prototypical external solver that would extend the capability of CSDL for PDE-based high-fidelity modeling. In this paper, we use FEniCSx [13], the new version of the FEniCS library with significant improvements, including support for a broader range of element types and complex number computation, among others. We will refer to them as FEniCSx and the legacy FEniCS respectively in the remainder of this paper. In this work, we present a general framework that integrates FEniCSx with CSDL for efficient modeling, simulation and optimization of multidisciplinary and multifidelity systems involving some, but not exclusively, PDE-based subsystems.

We present FEMO[1], a generalized tool that enables PDE solutions for multiphysics problems and provides automated derivative computation for optimization process. FEMO offers the following key features that enhance research in PDE-constrained MDO: (1) a highly-

[1] The source code of FEMO is available at `https://github.com/RuruX/femo`

modularized design that can be extended to interface external solvers, meeting the needs of large-scale multidisciplinary modeling and optimization; (2) a user-friendly API that minimizes the barrier of FEniCSx programming for optimization users, allowing them to focus on the modeling itself; and (3) demonstrations of eVTOL-related applications, showcasing efficient modeling for coupling aerodynamic and structural solvers of different levels of fidelity, as well as innovative approach to incorporate shape parametrization into PDE solutions in the motor design application.

The remainder of this paper is organized as follows. Section 2 introduces a model problem—namely, the optimal control of a nonlinear elliptic PDE—to illustrate the proposed methodology. It is followed with the implementation of FEMO in Section 3. In Section 4, we demonstrate the capabilities of this approach through a wide range of applications relevant to eVTOL subsystems, including topology optimization of a cantilever beam, shape optimization of an electric motor, and gust response simulation of an eVTOL wing and the associated sensitivity analysis. Section 5 draws conclusions and discusses possible future extensions of this work and anticipated challenges of applying it to system-level eVTOL aircraft design and optimization.

2 Methodology

We start with the definition of a PDE-constrained MDO problem in Section 2.1, and introduce a specific model problem—the optimal control of a nonlinear elliptic PDE. This model problem is used to illustrate the workflow of solving this optimization problem, which includes formulating the finite element equation with Nitsche's method, and obtaining derivatives for gradient-based optimization through the automatic code generation capabilities in FEniCSx.

2.1 A glimpse at a PDE-constrained MDO problem

The diagram in Fig. 1 illustrates a representative PDE-constrained MDO process, which involves one or two high-fidelity PDE sub-models, and multiple mid- to low-fidelity algebraic sub-models. The sub-model that computes objective function could be a weighted combination of several sub-functions such as efficiency, cost, weight, or maximum loads.

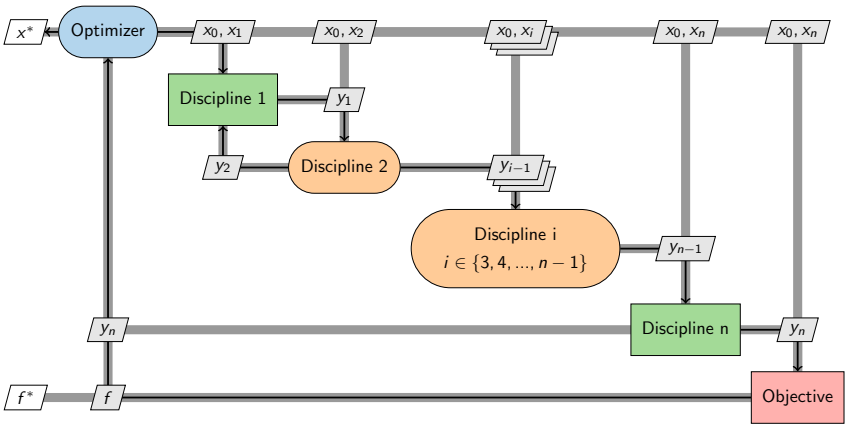


Fig. 1: A representative MDO process that encapsulates the most challenging aspects typically encountered in such problems: a. coupling multiple high-fidelity PDE sub-models; b. combining sub-models for disciplines with and without PDEs. The orange boxes represent the high-fidelity PDE sub-models, while the green boxes represent the mid- to low-fidelity non-PDE sub-models. The pink box represents the sub-model that computes the objective function.

In this process, coupling can occur in various ways among the sub-models, including one-way coupling and two-way coupling between disciplines. For instance, the two-way coupling between Discipline 1 and Discipline 2 could be used for load and displacement updates between an aerodynamic solver and a structural solver in aeroelastic analysis; the one-way coupling between Discipline n and Discipline $n-1$ could represent postprocessing for electric power losses based on solutions of the Maxwell's equations in electromagnetic modeling.

2.2 A model problem: optimal control of a nonlinear elliptic PDE

The methodology for solving a PDE-constrained MDO problem is presented as a generalized approach. To illustrate the workflow, we walk through the modeling process using the optimal control of a nonlinear elliptic PDE as the model problem [14]. This problem is commonly referred to as an inverse problem in the mathematical community, as the goal involves determining the source term given the analytical solution of the PDE. The mathematical model of this problem is formulated as:

$$\begin{aligned} \text{minimize} \quad & J(u,f) = \frac{1}{2}||u-g||_{L^2}^2 + \frac{\alpha}{2}||f||_{L^2}^2 && \text{(1a)} \\ \text{with respect to} \quad & f \\ \text{such that} \quad & -\Delta u + u^3 = f \quad \text{in } \Omega && \text{(1b)} \\ & u = g \quad \text{on } \Gamma \end{aligned}$$

where Ω is the domain of interest, Γ is the outer boundary of Ω. The state variable $u : \Omega \to \mathbb{R}$ and the control variable (or design variable) $f : \Omega \to \mathbb{R}$ are constrained by a boundary value problem (BVP) represented by (1b). The BVP consists of a nonlinear elliptic equation, specifically a nonlinear Helmholtz equation in this case, and subject to a Dirichlet boundary condition. The objective function, defined in (1a), is formulated as the sum of the L^2 norm of the difference between the state variable u and the analytical solution g, and an L^2 regularization term that depends on the control variable f.

The goal of this optimal control problem is to determine the optimal values of f that lead to a state variable u with the desired profile matching the analytical solution g, while satisfying the nonlinear elliptic PDE associated with f. This model problem is relatively simple and easily abstracted in the optimization problem setting, but due to the nonlinearity in the PDE it still presents a challenge.

2.3 Finite element solution of the model problem

Recall the strong form of the nonlinear Helmholtz equation in (1b). Find u, such that

$$\begin{aligned} -\Delta u + u^3 &= f \quad \text{in } \Omega\ , \\ u &= g \quad \text{on } \Gamma\ . \end{aligned}$$

The weak form is formulated as below. Find $u \in \mathcal{V}_g$, such that $\forall\, v \in \mathcal{V}_0$,

$$\int_\Omega \nabla u \cdot \nabla v \, d\Omega + \int_\Omega u^3 v \, d\Omega = \int_\Omega f v \, d\Omega\ , \tag{2}$$

where $\mathcal{V}_g$ and $\mathcal{V}_0$ are the trial and the test spaces defined as:

$$\begin{aligned} \mathcal{V}_g &:= \left\{ u \in H^1(\Omega) : u|_{\partial\Omega} = g \right\}\ , \\ \mathcal{V}_0 &:= \left\{ v \in H^1(\Omega) : v|_{\partial\Omega} = 0 \right\}\ . \end{aligned} \tag{3}$$

Let u_h, v_h be the Galerkin approximation of u, v. The discrete problem then becomes: find $u_h \in \mathcal{V}_h$, such that $\forall\, v_h \in \mathcal{V}_h$,

$$\int_\Omega \nabla u_h \cdot \nabla v_h \, d\Omega + \int_\Omega {u_h}^3 v_h \, d\Omega = \int_\Omega f v_h \, d\Omega\ , \tag{4}$$

where $\mathcal{V}_h$ is a finite-dimensional subspace of $H^1(\Omega)$.

In order to handle the Dirichlet boundary condition in a more systematic manner throughout the optimization process, Nitsche's method [15] is employed for weak enforcement. This approach involves introducing additional boundary terms to both sides of (4). By doing so, the boundary condition is automatically incorporated into the weak form and its associated partial derivatives with respect to the state variable u. With the boundary terms added to (4), we have

$$\begin{aligned}\int_\Omega \nabla u_h \cdot \nabla v_h \, d\Omega + \int_\Omega {u_h}^3 v_h \, d\Omega \; - \int_\Gamma (\nabla u_h \cdot \boldsymbol{n}) v_h \, d\Gamma \\ \mp \int_\Gamma (\nabla v_h \cdot \boldsymbol{n}) u_h \, d\Gamma \; + \int_\Gamma \frac{C_{\text{pen}}}{h} u_h v_h \, d\Gamma \\ = \int_\Omega f v_h \, d\Omega \mp \int_\Gamma (\nabla v_h \cdot \boldsymbol{n}) g \, d\Gamma \; + \int_\Gamma \frac{C_{\text{pen}}}{h} g v_h \, d\Gamma \; ,\end{aligned} \tag{5}$$

where $\boldsymbol{n}$ is the unit normal vector pointing outwards of the outer boundary Γ, h is a parameter indicating the refinement of the mesh (i.e., the cell diameter of the discretized domain), and C_{pen} is a positive constant independent of the mesh refinement. Rearranging terms we obtain the residual form of the nonlinear Helmholtz equation:

$$\begin{aligned}R(u_h, f) = \int_\Omega \nabla u_h \cdot \nabla v_h \, d\Omega + \int_\Omega {u_h}^3 v_h \, d\Omega \; - \int_\Omega f v_h \, d\Omega \\ - \int_\Gamma (\nabla u_h \cdot \boldsymbol{n}) v_h \, d\Gamma \\ \mp \int_\Gamma (\nabla v_h \cdot \boldsymbol{n})(u_h - g) \, d\Gamma \\ + \int_\Gamma \frac{C_{\text{pen}}}{h} (u_h - g) v_h \, d\Gamma \; .\end{aligned} \tag{6}$$

The $\mp$ sign in (6) corresponds to the symmetric $(-)$ and nonsymmetric $(+)$ variants of Nitsche's method. In this paper's applications, the symmetric variant is utilized. It is widely accepted by the finite element community and offers optimal accuracy in both L^2 and H^1 norms [16,17]. The nonsymmetric Nitsche's method has also gained increasing attention for its independence of the choice of penalty parameter C_{pen} [18]. However, we do not delve into the detailed comparison of these two variants as it is beyond the scope of this paper.

2.4 Automated derivative computation for gradient-based optimization

Gradient-based optimization is a preferred approach in MDO problems that involves evaluating high-fidelity sub-models, such as finite element solvers. It uses derivative information from the model to determine the direction of design variable updates, offering efficiency and accuracy. However, a major challenge in gradient-based optimization is ensuring that each component of the model has the required partial derivatives of its outputs with respect to its inputs. These derivatives are essential for computing the total derivative of the objective function with respect to the design variables.

To address this challenge, FEMO automates derivative computation as much as possible. It leverages the automatic differentiation capabilities of FEniCSx to compute partial derivatives of PDE residuals with respect to function inputs (i.e., $\partial R/\partial u$ and $\partial R/\partial f$), and utilizes CSDL for partial derivatives of algebraic operations, such as matrix-vector multiplications, occupying in preprocessing and postprocessing code.

For the model problem in (1), the total derivative of the objective function J with respect to the inputs f can be computed using the chain rule

$$\frac{dJ}{df} = \frac{\partial J}{\partial f} + \frac{\partial J}{\partial u}\frac{du}{df} \; . \tag{7}$$

With $\partial R/\partial u$ and $\partial R/\partial f$ provided by FEniCSx, we can obtain du/df by solving the linear system in (8), and insert it to (7) to compute the total derivative dJ/df.

$$R(u, f) = 0 \Rightarrow \frac{dR}{df} = 0 \Rightarrow \frac{\partial R}{\partial u}\frac{du}{df} = -\frac{\partial R}{\partial f} \ . \tag{8}$$

The assembly of partial derivatives to obtain total derivatives is handled automatically by CSDL, where it employs the direct or adjoint method based on the computational cost.

3 Implementation

In this section, we present a detailed explanation of how FEMO is used to solve PDE-constrained MDO problems. We begin with examples of FEniCSx and CSDL implementation, and then walk through the steps involved in solving the model problem using FEMO.

In Section 3.1, we introduce the native operations in CSDL such as matrix-vector products, and the `CustomOperation` class object, which serves as a wrapper for FEniCSx related operations. In Section 3.2, we provide a detailed explanation of FEniCSx programming workflow, accompanied by code examples for essential functionalities, such as computing partial derivatives and solving linear or nonlinear systems. In Section 3.3, we demonstrate how all the components are assembled together using the Python API of FEMO.

3.1 System-level modeling construction using CSDL

CSDL is a domain-specific language for constructing computational models in MDO. It provides access to a diverse set of operations such as summation, averaging, trigonometric functions, and array operations. Here is a simple model that utilizes built-in CSDL operations. In CSDL, the computation of partial derivative associated with built-in operations is automated. The execution of CSDL models is performed by the `Simulator` object in the Python backend of CSDL, which is implemented using Numpy.

```
class ExampleModel(csdl.Model):
    def define(self):
        # Declare inputs with default values
        x1 = self.declare_variable('x1', val=1.)
        x2 = self.declare_variable('x2', val=1.)

        # Compute the output with built-in operations
        y = x1**3 + sin(x2)*cos(x2)
        self.register_output('y', y)

# Run the simulation with the Python backend
sim = Simulator(ExampleModel())
sim.run()
```

In addition to the built-in operations, CSDL also supports user-defined custom operations, including *implicit* and *explicit* custom operations. *Implicit* custom operations typically involve solving a linear or nonlinear system to compute the output, making them suitable for wrapping PDE solvers. We will demonstrate how we utilize custom operations as wrappers for FEniCSx operations in Section 3.3. A CSDL model can contain multiple sub-models, where each of them has its own operation defined, whether built-in or custom. The total derivatives of models with coupled sub-models are computed using either direct or adjoint methods, depending on the number of design variables and the number of objectives. If there are more design variables than objectives, the adjoint method is executed, which is often the case for MDO problems.

3.2 Automated finite element solutions and derivatives with FEniCSx

FEniCSx is a collection of scientific libraries (UFL, DOLFINx, FFCx, Basix) written in Python and C++, where each library has a specific functionality in solving finite element problems. Fig. 2 illustrates the workflow between these libraries and their respective functionalities.

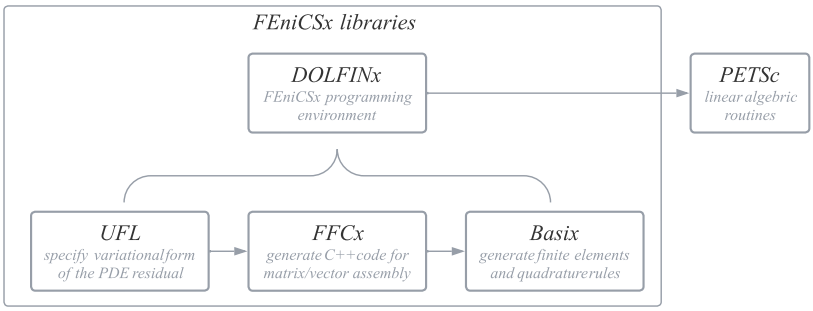


Fig. 2: FEniCSx workflow.

In the workflow for writing a FEniCSx program in Python, the user begins by importing the mesh using the XDMF file reader in DOLFINx. Next, they specify the variational form of the PDE residual using the Uniform Form Language (UFL) [19], which is a domain-specific language for variational expressions. DOLFINx then automatically interfaces with FFCx to generate high-performance finite element code in C++ on the backend.

For example, when solving the model problem, we first create a unit square mesh with triangular elements as the analysis domain. We then define a Lagrange finite element space for the solution u of the nonlinear elliptic PDE of C^1 continuity.

Listing 1 Runscript for solving the model problem

```
N = 16
# Create a 2D mesh with N elements on each edge
mesh = dolfinx.mesh.create_unit_square(N, N)
# Create a Lagrange function space for state variables with C^1 continuity
V =  dolfinx.fem.FunctionSpace(mesh, ('CG', 1))
u =  dolfinx.fem.Function(V)
```

Next, we express the interior terms and the boundary terms with symmetric Nitsche's method of the PDE residual using UFL operations. The code snippet in Listing 2 corresponds to the mathematical formulation described in (6).

The UFL also provides analytical computation of partial derivatives, such as $\partial R/\partial u$ and $\partial R/\partial f$ for the PDE residual R defined in Listing 2, i.e.,

```
dRdu = ufl.derivative(R, u) # is a variational form
```

The analytical derivatives computed by UFL can be assembled into discrete form using DOLFINx, resulting in PETSc arrays. These arrays can be further converted to Numpy or Scipy arrays to interface with CSDL. This process is generalized by FEMO as described in Section 3.3.

FEniCSx offers convenient build-in linear and nonlinear solvers for users and allows for the flexibility of custom solvers tailored to specific problem requirements. In FEMO, we utilize customized nonlinear solvers such as `NewtonSolver` and `SNESSolver` with `MUMPS` as the linear

Listing 2 (Continued) runscript for solving the model problem

```
    # Define the weak form of the nonlinear Helmholtz equation
interior_residual = inner(grad(u), grad(v))*dx \
                  + inner(u**3,v)*dx \
                  - inner(f, v)*dx
boundary_residual = - inner(dot(grad(u), n), v)*ds \
                  - inner(u-g, dot(grad(v), n))*ds \
                  + C_pen*h_E**(-1)*inner(u-g, v)*ds
R = interior_residual + boundary_residual
```

solver from PETSc. The code in FEMO is designed to be case-independent, enabling the solution of PDE problems in any discipline. The implementation details of the solvers, as well as other modularized components such as derivative form assembly and function value extraction, will be discussed in Section 3.3.

3.3 Coupling of FEniCSx and CSDL through FEMO

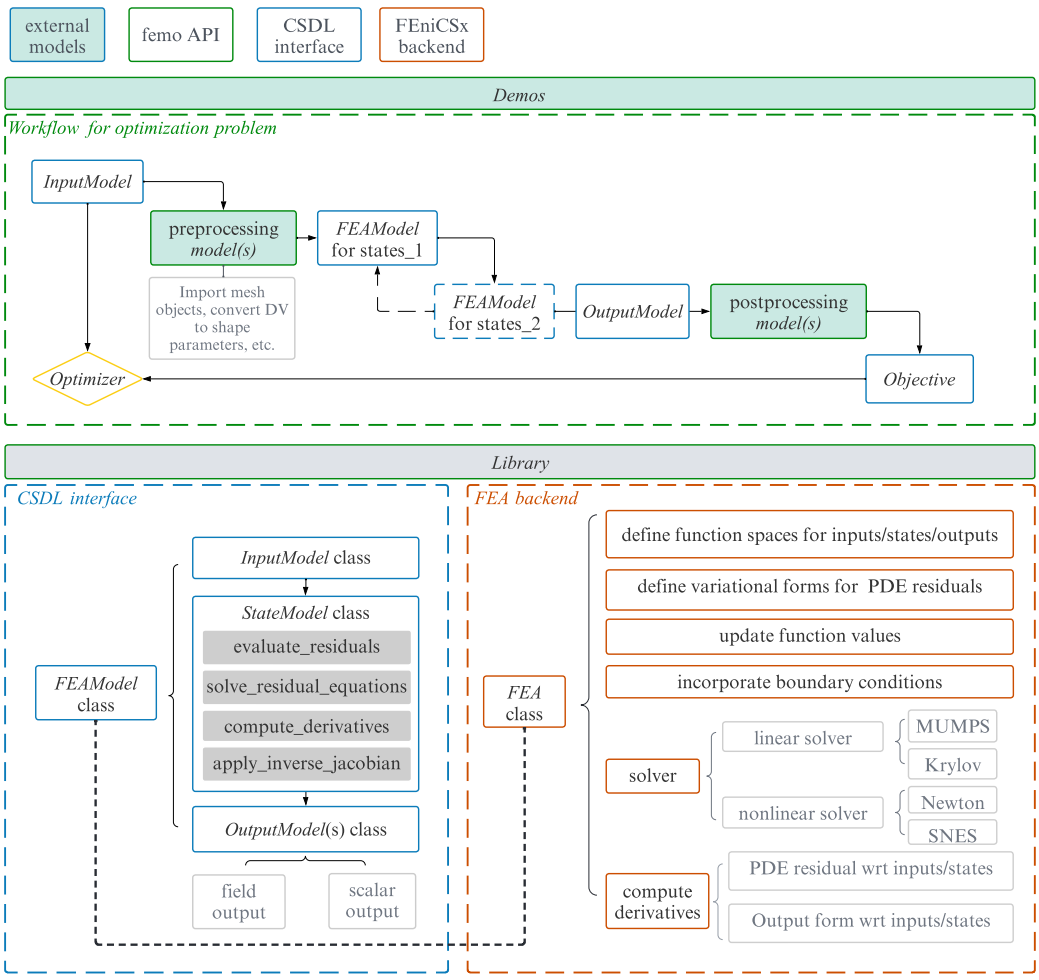


Fig. 3: FEMO framework

Integrating FEniCSx and CSDL for PDE-constrained MDO can be challenging due to their different domain-specific languages and data storage methods. FEniCSx uses UFL to express variational terms, such as integration and differentiation and stores data as PETSc sparse arrays. On the other hand, CSDL utilizes Numpy arrays for algebraic operations. Translating between these two software can be a tedious task, but it can be streamlined through the use of generic class objects. FEMO addresses this challenge by providing a generalized modeling framework for FEA-related computations and automating the data communication between the two libraries, as illustrated in Fig. 3.

The code example below showcases two convenient utility functions provided by FEMO: `assembleMatrix` converts analytical expression to discrete form, and `solveNonlinear` solves the finite element problem. These functions can be easily utilized within the CSDL modeling environment.

```
dRdu_array = assembleMatrix(dRdu)
solveNonlinear(R, u, solver='Newton')
```

By integrating all the utility functions within FEMO's `FEA` class object, we can provide essential functionalities required by CSDL's `CustomOperation`. This integration enables the construction of a generic `StateModel` for computing PDE solutions, and an `OutputModel` for assembling variational forms. Below is a code snippet illustrating the implementation of the generic `StateModel` in FEMO's source code. It takes a user-defined `FEA` object, along with input and output names, as parameters.

```
class StateModel(Model):
    def initialize(self):
        self.parameters.declare('fea', types=femo.FEA)
        self.parameters.declare('output_name', types=str)
        self.parameters.declare('input_name_list', types=list)

    def define(self):
        e = StateOperation(fea=self.fea,
                           input_name_list=input_name_list,
                           output_name=output_name)
        state = csdl.custom(*input_name_list, op=e)
        self.register_output(output_name, state)

# Define an implicit operation
class StateOperation(csdl.CustomImplicitOperation):
    ...
```

For the model problem, we utilize these generic models to create problem-specific CSDL custom operation models. These models can take FEniCSx `Function` or `Form` objects (i.e., `u` is a `Function` object defined in Listing 1) as the inputs and use generalized FEniCSx operations in FEMO to compute the outputs, as shown in Listing 3. The `type="scalar"` argument specifies that this model returns a scalar output, i.e., an integral over a certain domain. There is also a `"field"` type output, which computes distributed field outputs on the finite element mesh, i.e., element-wise and point-wise properties.

The `FEAModel` class is employed to group finite element-related operations together. It provides the flexibility to connect multiple PDE systems by including multiple `FEA` objects in its list argument, i.e., `FEAModel(fea=[fea1,fea2])`. This is used in the motor example in Section 4.3.

The coupling of sub-models in FEMO is automatically handled based on the names of the variables. Each input or output variable in the sub-models has a unique name associated with the object that stores its values. These names are automatically propagated throughout the system by CSDL and are used to connect to sub-models of other disciplines. Therefore, it is crucial to maintain consistency in variable names throughout the modeling process.

Listing 3 (Continued) runscript for solving the model problem

```
from femo.fea.fea_dolfinx import FEA # Interface with FEniCSx operations
from femo.csdl_opt.fea_model import FEAModel # group of FEA models

fea = FEA(mesh)
fea.add_input(name='f', function=f)
fea.add_state(name='u', function=u, residual_form=R, arguments=['f'])
fea.add_output(name='objective',
                # Could be either scalar or field variable
                type='scalar',
                # Defined as a UFL form object
                form=output_form,
                arguments=['u','f'])

# Could contain multiple fea objects for coupled PDE system
# Consists of one state model and several output models
fea_model = FEAModel(fea=[fea])
```

Finally, we ultilize the SNOPT library—a powerful software for solving nonlinear optimization problems [20]—as the optimizer for our numerical tests. We leverage the Python binding of SNOPT in modOpt[2], which can interface with CSDL by creating a `CSDLProblem` object with the simulator, as demonstrated in Listing 4.

Listing 4 (Continued) runscript for solving the model problem

```
from modopt.csdl_library import CSDLProblem
from modopt.snopt_library import SNOPT

# Create a CSDL problem object in modOpt
prob = CSDLProblem(problem_name='nonlinear_Helmholtz_opt',
                   simulator=sim)

# Create an optimizer object in modOpt
optimizer = SNOPT(prob,
                  Major_iterations = 1000,
                  Major_optimality = 1e-9,)

# Solve the optimization problem
optimizer.solve()
```

4 Numerical examples

In this section, we demonstrate the versatility and effectiveness of FEMO through four numerical examples that span various disciplines and complexity levels in modeling. In Section 4.1, we continue using the model problem discussed in previous sections as an example of a pure PDE-constrained problem, as there is no other algebraic sub-models coupled with it. In Section 4.2, we solve a classic density-based topology optimization problem for a 2D cantilever

[2] `https://github.com/LSDOlab/modopt`

beam. This example demonstrates the coupling of a PDE with non-PDE components, specifically the use of a density filter as a preprocessor. In Section 4.3, we tackle a more complex real-world problem for shape optimization of an electric motor. This problem involves solving two coupled PDEs: one for mesh deformation and the other for Maxwell's equations. Additionally, the PDE components are coupled with external preprocessors implemented in CSDL for shape parametrization. In Section 4.4, we construct a multifidelity aeroelastic solver that combines the Vortex Lattice Method (VLM) [21] implemented in CSDL with the Reissner–Mindlin shell solver implemented in FEniCSx. We utilize this aeroelastic solver in a dynamic test case simulating the gust response of an eVTOL wing. We also showcase the automatic computation of derivatives for this coupled system in a static test case. All of the examples utilize the Python binding of SNOPT, available in modOpt, as the optimizer.

4.1 Optimal control of a nonlinear elliptic PDE

4.1.1 Mathematical formulation

The MDO process for the model problem described in (1) is illustrated in Fig. 4. The optimal solution of the problem is denoted as f^*, and the associated state variable is represented by u^*. The objective function value at the end of the optimization iterations is denoted as J^*, i.e., $J^* = J(u^*, f^*)$.

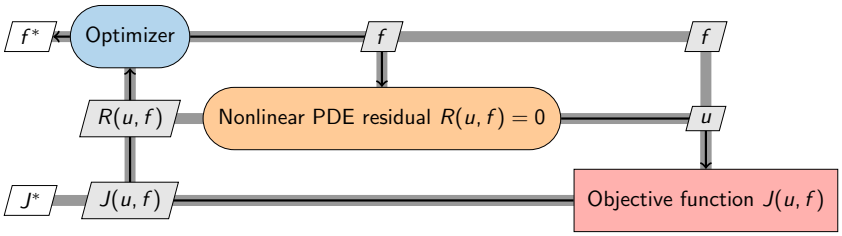


Fig. 4: The MDO process for the optimal control of a nonlinear Helmholtz equation. The orange box represents the finite element solver, and corresponds to the `StateModel` in FEMO; the pink box represents the objective function written as a variational form, corresponding to the `OutputModel` in FEMO.

4.1.2 Numerical results

For numerical analysis, we consider a unit square domain $\Omega = [0,1] \times [0,1]$ and choose $g(\boldsymbol{x}) = \sin(2\pi x_1)\cos(\pi x_2)$ as the analytical solution of u on Ω. We use symmetric Nitsche's method to weakly enforce the Dirichlet boundary condition $u = g$ on Γ. α is a dimensionless constant used in the regularization term of the objective function in (1a). For our specific case, we have selected $\alpha = 6 \times 10^{-7}$.

With the known analytical solution g, the optimal solution of f can be calculated analytically using the strong form of the nonlinear Helmholtz equation in (1). Specifically, we have $f_d = -\Delta g + g^3$, which can be easily implemented using UFL as

```
f_d = -ufl.div(ufl.grad(g)) + g**3
```

where g is a UFL form object defined beforehand.

The optimization is terminated when the norm of the objective gradient, which is the first-order total derivative, drops below a predefined tolerance. It is important to note that when working with finer meshes, the tolerance needs to be tightened to ensure proper convergence of the optimization problem. At the end of the optimization, we expect that the results for u

and f will closely approximate their respective analytical solutions, i.e., $u^* \to g$ and $f^* \to f_d$. To evaluate the success of the optimization problem, we calculate the L_2 norms for both pairs.

We perform a convergence study on mesh sensitivity by running optimization tests with different meshes. The results are summarized in Table 1. As observed, all the values in the last three columns decrease as the mesh becomes finer. Specifically, the error of the control variable f^* exhibits the expected first-order convergence, as reported in [22].

Table 1: A mesh sensitivity study for the optimal control of the nonlinear Helmholtz problem. Each row corresponds to a successfully converged optimization problem with the associated mesh. The L_2 norms, $||f^* - f_d||_{L_2}$ and $||u^* - g||_{L_2}$, represent the differences between the optimized control and state variables and their respective analytical solutions. J^* is the objective function value by the end of the optimization.

Number of elements	$\|\|f^* - f_d\|\|_{L_2}$	$\|\|u^* - g\|\|_{L_2}$	J^*
4×4	24.017097	0.075343	0.003312
8×8	6.999477	0.016861	0.000374
16×16	2.824094	0.004063	0.000205
32×32	1.376501	0.001253	0.000190
64×64	0.673929	0.000806	0.000187
128×128	0.331941	0.000767	0.000187

Fig. 5 illustrates the evolution of the control variable f and the state variable u during the optimization test with a 128×128 mesh. The initial guess for all degrees of freedom in f is set to 0.1. This optimization run requires 549 iterations to converge to a tolerance of 10^{-11}. In this problem, the control variable f is represented as a discontinuous Lagrange function of degree 0, resulting in a total of 128^2 degrees of freedom to be optimized.

Initially, the distribution of f exhibits a scattering profile for $N = 200$. As the optimization progresses, it gradually becomes smoother. This smoothing effect can be attributed to the presence of f in the regularization term of the objective function J.

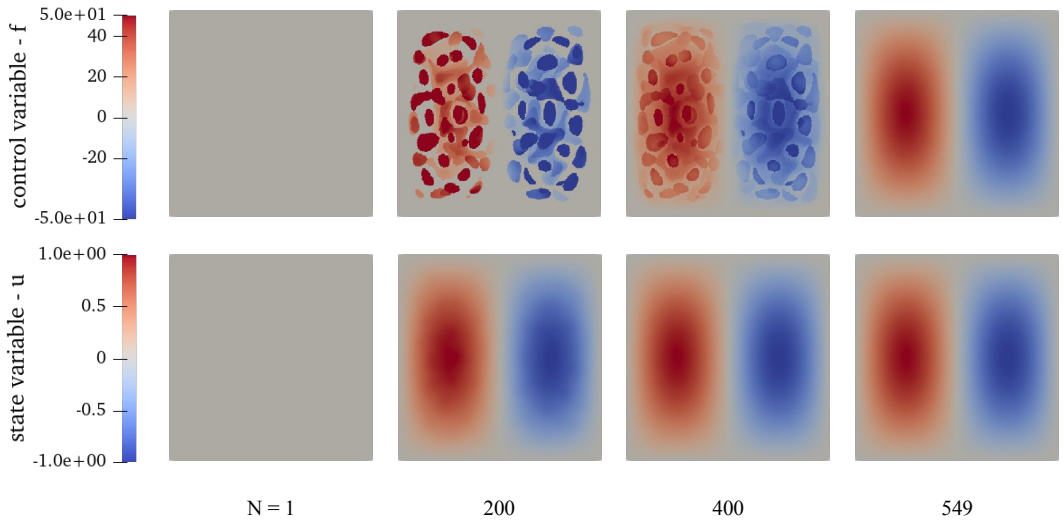


Fig. 5: Snapshots of the optimization process for the model problem with a 128×128 mesh. The control variable f has a uniform value of 0.1 across the entire domain at the first optimization iteration ($N = 1$) and the optimization converges at $N = 549$.

4.2 Topology optimization of a cantilever plate

We demonstrate the versatility of this software by applying it to a classical 2D cantilever beam topology optimization problem using a density-based approach. This problem is a well-known benchmark in the field of topology optimization, and has been extensively studied using various optimization approaches. Notably, Chung et al. [23] implemented the problem using OpenMDAO [24], a NASA developed MDO framework, while Yan et al. [25] automated gradient-based topology optimization by coupling the legacy FEniCS and OpenMDAO in their previous work.

4.2.1 Mathematical formulation

We aim to optimize the density distribution of the material, denoted as ρ, subjected to a constant volume constraint. The structural behavior of the beam is governed by the PDE of linear elasticity. The objective of this optimization problem is to minimize the compliance of the beam, which is defined as the inverse of the stiffness, as indicated in the first equation of (9).

$$\begin{aligned} \text{minimize} \quad & J(\boldsymbol{u}) = \int_\Gamma \boldsymbol{u} \cdot \boldsymbol{f} \, d\Gamma \\ \text{with respect to} \quad & \rho \\ \text{such that} \quad & \tilde{\rho} = \tilde{\rho}(\rho) \\ & \int_\Omega \tilde{\rho} \, d\Omega = \text{constant} \\ & -\nabla \cdot \sigma(\boldsymbol{u}, \tilde{\rho}) = \boldsymbol{f} \\ & \rho_{min} < \rho \leq 1 \end{aligned} \tag{9}$$

To smooth the density distribution and prevent checkerboard patterns, which can cause instability, a distance-based method is employed to map the original density ρ to the filtered density $\tilde{\rho}$. For each element i, its filtered density is computed by summing the weighted contributions of nearby elements, given by the equation:

$$\tilde{\rho}_i = \sum_j w_{ij} \rho_j \ , \tag{10}$$

where $\tilde{\rho}_i$ is the filtered density of element i, ρ_j is the unfiltered density of a nearby element j, and w_{ij} is the weight function that connects element i and j, which satisfies $\sum_j w_{ij} = 1$. In this example, we use a simple conic weight function to compute w_{ij} based on the distance between element i and j, whose formula can be found in [26].

The solid isotropic material with penalization (SIMP) method is employed to incorporate the density distribution into the PDE constraint, which was initially developed by Bendsøe and Kikuchi [27]. In this specific application, the local material property is proportional to the power product of the local density, i.e., $E_i(\tilde{\rho}_i) = \tilde{\rho}_i^3 E_0$. Additionally, a minimum value of density, denoted as ρ_{min}, is imposed to avoid material singularity and maintain numerical stability during the finite element analysis.

Fig. 6 shows the MDO process of this problem. The distance-based density filter sub-model is implemented with CSDL linear algebra operations, and marked in green as it is considered as an external component, while the finite element sub-models are marked in pink (`OutputModel`) and orange (`StateModel`).

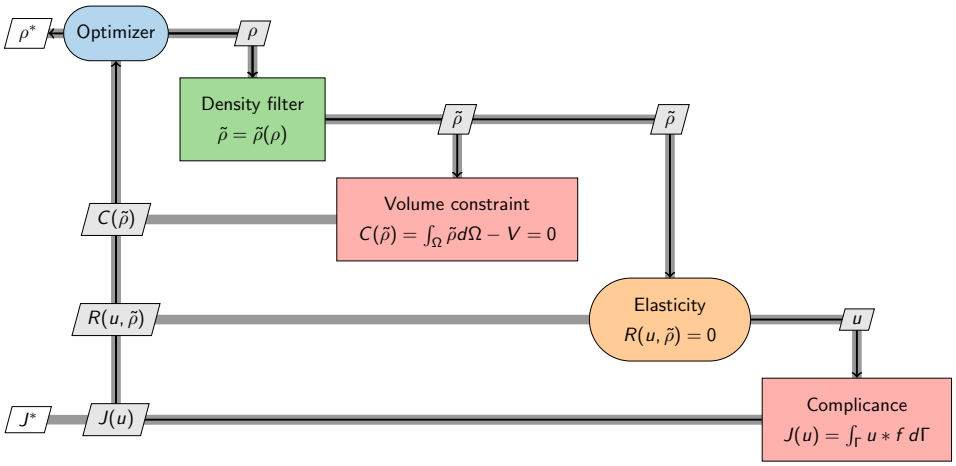


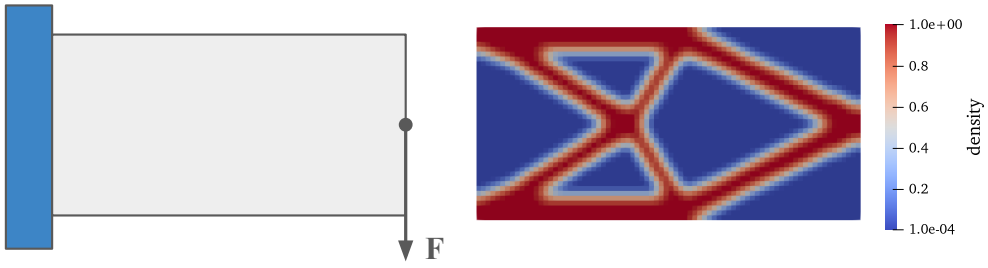
Fig. 6: The MDO process for topology optimization of a cantilever beam.

4.2.2 Numerical results

The physical setting of the boundary value problem is shown in Fig. 7a. The compliance of a cantilever beam is minimized. The beam has a clamped boundary condition on the left edge, and a downward traction force applied in the middle of the right edge. The beam has a length of 160 and a height of 80. It is discretized into a structured quadrilateral mesh, with user-defined number of elements on each side.

Linear elasticity is assumed in this example, but it can be extended to nonlinear cases as discussed in [25]. The Young's modulus is set to 1.0 and the Helmholtz ratio is set to 0.3. The traction force, denoted as F, has a magnitude of 0.25 and is directed downwards. ρ_{min} is set to 10^{-4} as the lower bound of the design variable. Note that all of the parameters are dimensionless.

The optimization terminates when the gradient drops under 10^{-8}, which occurs within 338 major iterations. Fig. 7b illustrates the results of the optimization for the penalized density $\tilde{\rho}$. The density distribution exhibits a tapered shape as anticipated in prior studies [23,25]. Furthermore, due to the linear nature of this problem, the density distribution displays a symmetric profile.



(a) (b)

Fig. 7: Topology optimization of a cantilever beam with linear elasticity. (a) the problem setting; (b) the optimized density distribution $\tilde{\rho}$ with a 80×40 mesh.

4.3 Electric motor design

In this section, we tackle a practical problem including the shape optimization of an electric motor. This problem is an essential part for the system-level modeling and optimization of eVTOL aircraft. Using the presented tool, we are able to build a high-fidelity model that can accurately capture the precise mesh movement near the motor components and their impact on the electromagnetic performance of the motor. This is achieved by formulating the relevant processes as coupled PDEs and connecting them with an external solver for shape parametrization. Specifically, we focus on studying a permanent magnet synchronous motor (PMSM), and analyze the behavior of a single static phase within its periodic operation.

4.3.1 Optimization problem definition

The optimization problem is formulated shown in (11), where $\boldsymbol{x}$ represents a vector of design variables that define the shape of the motor. Ω_{bc} is a vector of boundary movements on the finite element mesh. Ω denotes the original mesh and $\tilde{\Omega}$ represents the moved mesh. The objective function $J(\cdot)$ depends on the deformed mesh $\tilde{\Omega}$ and the solution of Maxwell's equations A_z—the z-component of magnetic potential vector A.

The problem is governed by two nonlinear PDEs: one for mesh movement (hyperelasticity) and another for electromagnetic modeling (Maxwell's equations). These PDEs are coupled in a way that the solution of the former—the nodal deformation—is incorporated in the PDE residual of the latter. Hence the nodal deformation can have an impact on the final solution of the magnetic flux density. The corresponding MDO process is abstracted in Fig. 8.

$$
\begin{aligned}
\text{minimize} \quad & J(A_z, \tilde{\Omega}) \\
\text{with respect to} \quad & \boldsymbol{x} && \text{– design variables} \\
\text{such that} \quad & S(\boldsymbol{x}) = \Omega_{bc} && \text{– shape parametrization} \\
& M(\tilde{\Omega}, \Omega, \Omega_{bc}) = \mathbf{0} && \text{– mesh movement} \\
& R(A_z, \tilde{\Omega}) = \mathbf{0} && \text{– Maxwell's equations} \\
& \boldsymbol{x_{min}} \leq \boldsymbol{x} \leq \boldsymbol{x_{max}}
\end{aligned}
\tag{11}
$$

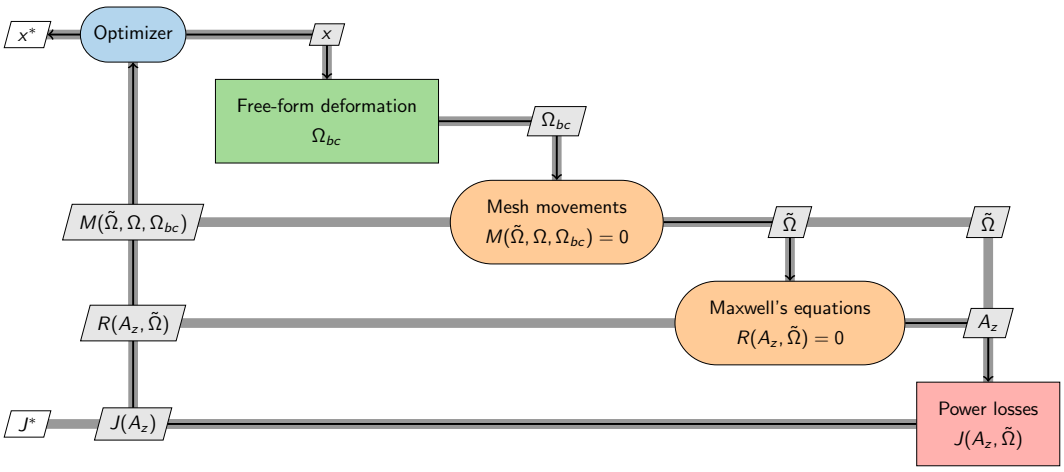


Fig. 8: The MDO process for shape optimization of an electric motor.

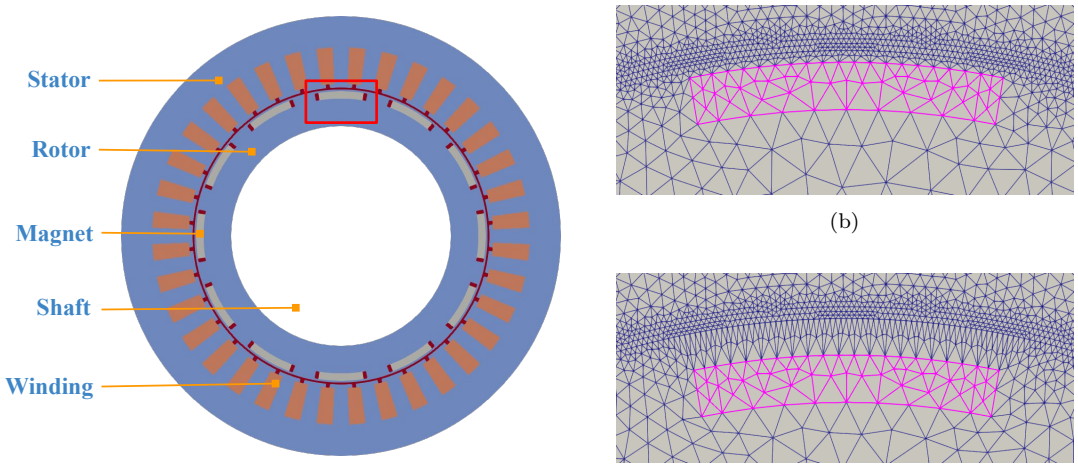


Fig. 9: (a) The baseline geometry of the permanent magnet synchronous motor model. The red box highlights one of the twelve magnets. (b) The zoomed view of the red-boxed region with the undeformed mesh. The magnet is highlighted in fuchsia color. (c) The zoomed view of the red-boxed region with the deformed mesh. The magnet highlighted in fuchsia has been moved inwards along the radial direction.

4.3.2 Shape parametrization with free-form deformation

We start with an initial configuration of the motor geometry as shown in Fig. 9a. We use the vector of design variables $\boldsymbol{x}$ to describe the changes in the geometric parameters of the motor, such as the inner radius of the rotor and the positions of the magnets.

To incorporate these changes into the physical domain, we employ free-form deformation (FFD), which is a technique commonly used for modeling the deformation of rigid objects and is analogous to parametrization with B-splines. FFD allows us to map the geometric changes $\boldsymbol{x}$ to the movements of a specific group of boundary-defining points in the physical domain. These points are denoted as Ω_{bc}.

It is important to note that this problem involves numerous modeling specifications and steps in the preprocessing and postprocessing stages. However, for the purpose of this paper, these details are omitted, and the focus is primarily on the PDE-related aspects of the formulation.

4.3.3 Automated mesh movement

For the sake of notation convenience, we introduce $\boldsymbol{u_{bc}}$ to represent the boundary movement Ω_{bc} on the finite element domain. Additionally, we denote $\boldsymbol{u}$ as the displacements for all the mesh vertices in the entire domain:

$$\boldsymbol{u}(\boldsymbol{X}) = \phi(\boldsymbol{X}) - \boldsymbol{X} \ , \tag{12}$$

where ϕ: $\boldsymbol{X} \rightarrow \tilde{\boldsymbol{X}}$ represents the mapping from the original domain Ω to the deformed domain $\tilde{\Omega}$. This approach for updating mesh based on the mesh movement is analogous to the approach used in fluid-structure interaction applications [28].

To simulate the mesh movement, we formulate a BVP where the displacements of the boundary vertices $\boldsymbol{u_{bc}}$ are imposed as Dirichlet boundary conditions. The BVP is governed by a hyperelastic equation. By solving this BVP, we obtain the smooth movements $\boldsymbol{u}$ throughout the entire domain, including both the on-boundary and off-boundary vertices. The strong form of the hyperelasticity problem is as follows

$$\begin{aligned} -\nabla \cdot \boldsymbol{P}(\boldsymbol{u}) &= \boldsymbol{f} \quad \text{in } \Omega \ , \\ \boldsymbol{u} &= \boldsymbol{u_{bc}} \quad \text{on } \Gamma_{bc} \ , \end{aligned} \tag{13}$$

where $\boldsymbol{f}$ represents the body force, which is assumed to be zero in our case. The first Piola-Kirchhoff stress $\boldsymbol{P}$ is given by

$$\boldsymbol{P} := J\sigma \boldsymbol{F}^{-T} \ , \tag{14}$$

where σ represents the Cauthy stress, $\boldsymbol{F}$ is the deformation gradient, and J is its determinant. $\boldsymbol{F}$ can be computed by

$$\boldsymbol{F} = \frac{\partial \phi}{\partial \boldsymbol{X}} = \nabla \boldsymbol{u} + \boldsymbol{I} \ , \tag{15}$$

which can be conveniently implemented in UFL as follows:

```
def F(u):
    return ufl.grad(u)+ufl.Identity(2) # 2D problem
```

For the material model, we adopt the modified St. Venant–Kirchhoff constitutive model, where the second Piola-Kirchhoff stress is defined as:

$$\boldsymbol{S}(\boldsymbol{F}) = 2\mu(\boldsymbol{E} - \frac{tr(\boldsymbol{E})}{3}\boldsymbol{I}) + K\mathrm{tr}(\boldsymbol{E})\boldsymbol{I} \ . \tag{16}$$

Here, $\boldsymbol{E} = \frac{1}{2}(\boldsymbol{F}^T\boldsymbol{F} - I)$ is the Green-Lagrange strain. μ and K are the material parameters that can be chosen based on the deformation to introduce artificial stiffness in regions of the mesh that experience crushing. In our numerical tests, we set $\mu = K = J^{-p}$, where p is a positive integer (set to 3).

Then the first Piola-Kirchhoff stress can be computed by

$$\boldsymbol{P}(\boldsymbol{F}) = \boldsymbol{F} \cdot \boldsymbol{S}(\boldsymbol{F}) \ . \tag{17}$$

In the finite element formulation, the boundary condition is weakly enforced to the weak form of (13) using the symmetric Nitsche's method applied on the subdomains. To specify the relevant facets for the boundary condition, we define an integral operator using UFL, such as:

```
dss = ufl.Measure('ds', domain=mesh, subdomain_data=boundary_data)
```

Here, `ds` represents the integral operator on facets in the finite element domain that are not on the outer boundary, and `dss` is a user-specified integral operator on the selected facets based on the provided boundary data, which contains the indices of the desired facet entities. The group of those facets corresponds to Γ_{bc} in (13), which is a subset of the total boundary Γ.

Fig. 9 illustrates how the mesh deforms when moving a magnet inwards along the radial direction. Fig. 9b represents the undeformed mesh. Nonzero displacements are assigned to the vertices located on the outer edges of the magnet, while the vertices on the unmoved boundaries, such as the air gap (the finely discretized area above the magnet), are restrained to remain unmoved. Fig. 9c shows the deformed mesh obtained by warping the domain using the solution of the hyperelastic problem. The deformed mesh captures the movements of the vertices both inside and surrounding the magnet, resulting in a smooth mesh of the new configuration.

We note that the mesh is not actually moved during the optimization process. Instead, the mesh movement is incorporated into Maxwell's equations, allowing us to compute the electromagnetic solution as if it were on the deformed mesh. This coupling method will be further explained in the next subsection.

An important advantage of this approach is that the number of vertices and their connectivity relationship remain constant throughout the optimization process. By computing the displacements of the vertices and incorporating them into the subsequent steps, we eliminate the need to regenerate or move the mesh, resulting in reduced computational cost.

To enhance the robustness of this algorithm for handling large deformation during the optimization process, we employ a customized nonlinear solver with load-stepping to incrementally apply the subdomain movements. The number of steps is dynamically determined based on the mesh size and the maximum expected deformation. This adaptive approach helps prevent inverted elements, which can cause the subsequent electromagnetic solver to fail.

4.3.4 Magnetostatic problem on the deformed mesh

Maxwell's equations are

$$\begin{aligned} \nabla \times \boldsymbol{H} &= \boldsymbol{J}_w + \boldsymbol{J}_m \ , \\ \nabla \cdot \boldsymbol{B} &= 0 \ , \end{aligned} \tag{18}$$

where $\boldsymbol{J}_w$ and $\boldsymbol{J}_m$ are source terms representing the effective current densities within the stator windings and the magnets respectively. $\boldsymbol{B}$ represents the magnetic flux density, and $\boldsymbol{H}$ represents the magnetizing field.

The relationships between $\boldsymbol{B}$, $\boldsymbol{H}$, and $\boldsymbol{A}$ (the magnetic vector potential) are given by

$$\begin{aligned} \boldsymbol{B} &= \mu \boldsymbol{H} \ , \\ \boldsymbol{B} &= \nabla \times \boldsymbol{A} \ , \end{aligned} \tag{19}$$

where μ is the magnetic permeability, which is a nonlinear function of B and depends on the local material properties.

Rewriting Maxwell's equations in (18) by using $\boldsymbol{A}$ as the solution gives us

$$\nabla \times (\mu^{-1} \nabla \times \boldsymbol{A}) = \boldsymbol{J}_w + \boldsymbol{J}_m \ . \tag{20}$$

This can be reduced by considering only the z-component of $\boldsymbol{A}$ as the other two components are both zero

$$\begin{aligned} -\nabla \cdot (\mu^{-1} \nabla A_z) &= \boldsymbol{J}_z \quad \text{in} \quad \Omega \ , \\ A_z &= g \quad \text{on} \quad \Gamma \ . \end{aligned} \tag{21}$$

We choose $g = 0$ for a homogeneous boundary condition of A_z. Then $\boldsymbol{B}$ can be obtained from A_z in the postprocessing steps by

$$\boldsymbol{B}(x, y) = \left(\frac{\partial A_z}{\partial y}, -\frac{\partial A_z}{\partial x} \right) \ . \tag{22}$$

The weak form of (21) can be formulated as

$$\int_\Omega \mu^{-1} \nabla A_z \cdot \nabla v \, d\Omega = \int_\Omega J_z v \, d\Omega \ , \tag{23}$$

where v is a test function of A_z. By moving the right-hand side term to the left and applying the Dirichlet boundary condition with Nitsche's method, we obtain the PDE residual in variational form on the undeformed mesh:

$$\begin{aligned} R(A_z, \Omega) = &\int_\Omega \mu^{-1} \nabla A_z \cdot \nabla v \, d\Omega \ - \int_\Omega J_z v \, d\Omega \\ &- \int_\Gamma \mu^{-1} (\nabla A_z \cdot \boldsymbol{n} \cdot v \ \mp \nabla v \cdot \boldsymbol{n} \cdot (A_z - g)) \, d\Gamma \\ &+ \int_\Gamma \frac{C_{pen}}{h} \mu^{-1} v (A_z - g) \, d\Gamma \ . \end{aligned} \tag{24}$$

To obtain the variational form on the deformed mesh, we adopt a seamless coupling method for hyperelasticity and electromagnetism by incorporating $\boldsymbol{u}$ into the weak form in (24). This is inspired by the body-fitted method for Fluid-Structure Interaction (FSI) problems [29], where the integral and differential operators are modified by Nanson's formula into the weak forms. For example,

$$\begin{aligned} d\Omega &\rightarrow J d\Omega \\ \boldsymbol{n} d\Gamma &\rightarrow J \boldsymbol{F}^{-T} \boldsymbol{n} d\Gamma \\ d\Gamma &\rightarrow |J \boldsymbol{F}^{-T} \boldsymbol{n}| d\Gamma \\ \nabla v &\rightarrow \nabla v \cdot \boldsymbol{F}^{-1} \\ &\ldots \end{aligned} \tag{25}$$

Here, J represents the determinant of $\boldsymbol{F}$, which is computed by $J = det(\boldsymbol{F})$. These modifications can be easily handled by UFL, i.e., $\nabla v \cdot \boldsymbol{F}^{-1}$ can be conveniently implemented as

```
def gradx(f,u):
    return ufl.dot(ufl.grad(f),ufl.inv(F(u)))
```

where `F(u)` is the predefined function as described in Section 4.3.3.

4.3.5 Numerical results

For the numerical tests, we set up a simplified optimization problem with a single design variable—the change of magnet position in the radial direction. The objective is to minimize the natural core losses due to the magnets during no-load operation. In the simplified setting, the total core loss P_L are given by the sum of the eddy current loss P_{ec} and the hysteresis loss P_h, i.e., $P_L = P_{ec} + P_h$. The losses are computed by

$$P_{ec} = 2\pi^2 f^2 l K_{ec} \int_{\Omega_{ec}} |\boldsymbol{B}|^2 d\Omega \quad \text{and} \quad P_h = 2\pi f l K_h \int_{\Omega_h} |\boldsymbol{B}|^\beta d\Omega \ , \tag{26}$$

where f is the frequency of the input signal in Hertz, l is the motor length, and they are both set to unit values. K_{ec} is the eddy current coefficient, and K_h is the Steinmetz hysteresis coefficient. Ω_{ec} and Ω_h represent the subdomains where the power loss effects are observed (typically the rotor and the stator). β is the Steinmetz exponent that depends on material, ranging from 1.5 to 2.5. β is set to 1.76835 for the chosen Hiperco 50 the alloy material, which is the material used for the rotor and the stator. Again, we need to convert the integral operators in (26) to the deformed mesh using the substitutions given in (25).

As shown in Fig. 10, this single-variable single-objective optimization converges successfully. The objective gradient, which is the total derivative of the objective with respect to the design variable, drops below 10^{-8} within 8 iterations.

The magnitude of the magnetic flux density $\boldsymbol{B}$ is plotted on both sides of the motor in Fig. 11. The left side represents the magnetic flux solution field on the undeformed mesh, while the right half shows the results obtained on the deformed mesh.

Although it is not obvious to notice the difference of the two sides as they use the same color scale, it is worth mentioning that the maximum magnitude of $\boldsymbol{B}$ on the right is 2.1, which is 8.7% smaller than the maximum value of 2.3 on the left as in the color legend. Also, in the optimized configuration, a lighter color of $\boldsymbol{B}$ magnitude can be observed in the stator area between the stator teeth. This indicates a reduction in the power loss in this region, leading to improved performance. The decrease in peak magnetic flux density can be explained by the enlarged area of the rotor between the magnets and the air gap. The movement of the magnets increases the area for flux to permeate through this tight zone, which decreases the effective flux density. The movement of the magnets also decreases the flux that permeates from the rotor to the stator, reducing the flux density magnitudes in the stator teeth.

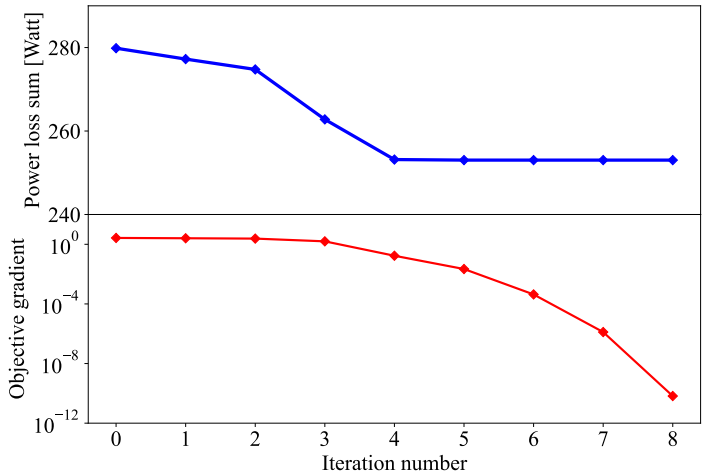


Fig. 10: Convergence of the objective function (total core loss) and the objective gradient during optimization iterations.

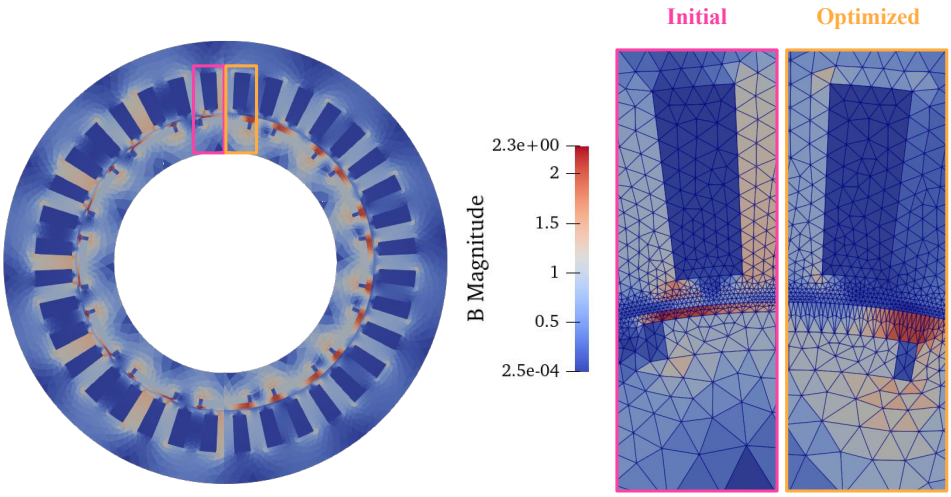


Fig. 11: The magnetic flux density on the initial configuration (pink) and the optimized configuration (orange).

4.4 Aeroelastic analysis on an eVTOL wing

This application employs aeroelastic coupling of a high-fidelity structural solver based on Reissner–Mindlin shell theory and a low-fidelity aerodynamics solver: the vortex lattice method (VLM). A solver-independent method [30] is used for coupling these two solvers. Numerical tests include a static aeroelastic analysis with analytical derivative computation and a dynamic case for gust response on an eVTOL wing. We use the Uber eCRM-001 model in the following test cases, which is an electric common reference model (eCRM) for electric air taxi.

4.4.1 Aerodynamic solver with vortex lattice method

VLM is a widely used approach for analyzing aerodynamic performance, such as force or pressure distribution on lifting surfaces [21]. VLM makes a potential flow assumption, which means the flow can be characterized by an irrotational velocity field given by

$$\nabla^2 \Phi = 0 \quad , \tag{27}$$

where Φ is the fluid potential.

Other assumptions of VLM include that 1) the fluid is incompressible; and 2) the lifting surfaces are thin and have a small angle of attack and sideslip.

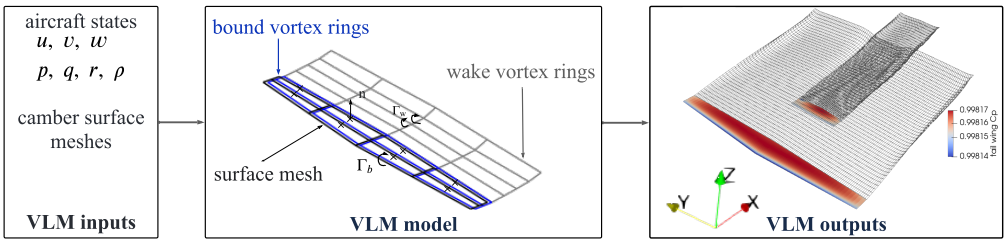


Fig. 12: VLM workflow for aerodynamic analysis.

The VLM model is built in CSDL to enable automated derivative computation. It is implemented in an inertial frame with a constant freestream velocity and assumes steady-state conditions during each mission segment. Fig. 12 demonstrates the inputs, model, and outputs of the VLM model. The inputs of the VLM model are the meshes and the states of the lifting surfaces. Then the model creates structured quadrilateral panels and uses vortex rings located at the quarter-chord of each panel for computation. The system of equations to be solved in (28) corresponds to the flow-tangency condition meaning zero normal velocity on the lifting surface panels.

$$\boldsymbol{A\Gamma}_{\mathrm{b}} + \boldsymbol{B\Gamma}_{\mathrm{w}} + \boldsymbol{v} \cdot \boldsymbol{n} = 0 \ , \tag{28}$$

where $\boldsymbol{v}$ is the kinematic velocity, and $\boldsymbol{n}$ is the normal vector of the lifting surface panels. $\boldsymbol{\Gamma}_{\mathrm{b}}$ and $\boldsymbol{\Gamma}_{\mathrm{w}}$ represent the bound and the wake vortex circulation strengths, respectively. $\boldsymbol{A}$ and $\boldsymbol{B}$ are the aerodynamic influence coefficient matrices.

The outputs of the VLM model are the traction forces on each panel given by the panel forces divided by the areas. The panel forces are calculated using the Kutta–Joukowski theorem [31] as velow

$$\boldsymbol{F} = \rho \boldsymbol{\Gamma}_{\mathrm{b}} \boldsymbol{v}_{\mathrm{ind}} \times \boldsymbol{s} \ , \tag{29}$$

where ρ is the density of air, $\boldsymbol{v}_{\mathrm{ind}}$ is the induced velocity evaluated at the bound vortex collocation points, and $\boldsymbol{s}$ represents the bound vector.

4.4.2 Thin-shell structural solver with Reissner–Mindlin plate theory

We implement the Reissner–Mindlin shell formulation in FEniCSx[3], based on the geometrically-linear, quadrilateral-element variant of the fully nonlinear unconventional shell formulation developed by Campello, et al. [32]. This formulation considers six-parameter solution fields (three for displacements and three for rotations).

There are other openly-available implementations of this shell formulation in the FEniCS community, such as the pedagogical example by Bleyer [33], with slight difference in the formulation of the drilling stabilization term. Moreover, the existing implementation is implemented in legacy FEniCS with limited support for quadrilateral elements, while our shell solver is implemented in FEniCSx, which provides robust support for quadrilateral elements.

While our shell solver can deal with both triangular and quadrilateral elements, it currently cannot handle meshes containing multiple types of elements, i.e., meshes containing both triangular and quadrilateral elements, due to a limitation in the FEniCSx mesh importing module.

4.4.3 Work-conservative aeroelastic coupling

The VLM aerodynamic solver is loosely coupled with the FEniCSx structural solver by linearly mapping the aerodynamic loads and structural displacements between both solvers. We use the approach presented in [30] to construct these linear load and displacement maps. In this approach, we start by expressing the aeroelastic work with each solver. This is achieved by defining a matrix M for each solver, such that the aeroelastic work can be computed as the product $\boldsymbol{u}^T M \boldsymbol{F}$. Here, $\boldsymbol{u}$ and $\boldsymbol{F}$ are the vectors containing the degrees of freedom of the displacements and the loads respectively. The product $\boldsymbol{u}^T M \boldsymbol{F}$ encodes all solver-specific information used in computing aeroelastic work.

Next, we define a linear map to transfer the displacements from the structural solver to the VLM solver. Then we construct the linear map that transfers aerodynamic loads from the VLM solver to the structural solver following the expressions of aeroelastic work and the displacement map. The resulting load map ensures that the load-displacement mapping pair conserves aeroelastic work and can also be designed to conserve the total aerodynamic load in each principal direction. In this application, both the displacement map and the load map are linear, which allows them to be used in gradient-based optimization algorithms.

[3] The source code of this FEniCSx-based shell solver is available on Github at `https://github.com/RuruX/shell_analysis_fenicsx` with numerous examples and the associated mesh files.

4.4.4 Static aeroelastic simulation with constant wind velocity

We start with a simple static case of aeroelasticity, where a constant wind velocity $V_\infty = 50m/s$ is applied. We consider an angle of attack $AoA = 6°$, and the air density is set to the International Standard Atmosphere air density at sea level, $\rho = 1.225$ kg/m^3. The aerodynamic solver initiates with an original VLM mesh and outputs the aerodynamic loads, which are then projected onto the structural domain. The structural solver computes the displacements, which in turn lead to updates in the VLM mesh. This iterative process continues until the updates in the VLM mesh coordinates between the previous and the current iterations fall below 10^{-6}. We use an embeded nonlinear block CG solver for this iterative coupling relationship. The results of the von Mises stress and the aerodynamic load are shown in Fig. 13.

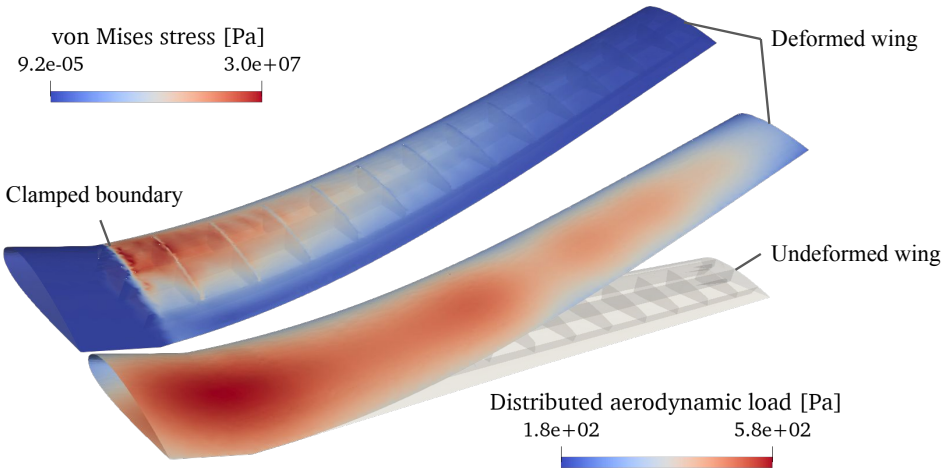


Fig. 13: Static aeroelastic analysis of the eCRM-001 wing model with constant wind velocity and clamped boundary condition at the wing root. The deformation of the wing structure is scaled by a factor of 40 for visualization.

4.4.5 Dynamic aeroelastic simulation of wind gust response

For the dynamic scenario, we study the response of the eVTOL wing subjected to gust excitation. The eVTOL aircraft is initially cruising at a constant speed of $v_0 = 50$ m/s under steady wind in the x direction, $V_\infty = 50$ m/s. Then we apply a gust wind in the z-direction with a sinusoidal profile given by $V_{gust} = v_0 \times (1 - cos(2\pi(t - T_0)/T_1))$, where T_0 is the time when the gust is activated and T_1 is the duration of the gust application.

We simulate the aeroelastic response of the wing under this gust excitation, considering three phases: the steady state, the gust application period, and the oscillation after gust removal. We choose the implicit midpoint rule in FEniCSx as the time integration method for the dynamic structural solver, and the steady VLM solver.

In Fig. 14, the gust velocity and the tip displacement of the eVTOL wing are shown over time. The peak tip displacement occurs slightly after the maximum gust velocity. This aligns with our expectation since the velocity of wing tip is not zero yet at the maximum gust velocity, leading to a continued upward movement of the wing until the velocity of the wing itself decreases to zero. The amplitude of the tip displacement is constant during the oscillation phase because there is no damping in the elastic model. In the plot of tip displacements, a convergence study of time step size is performed and the solution converges when $\Delta t = 0.001$ (Nsteps=50).

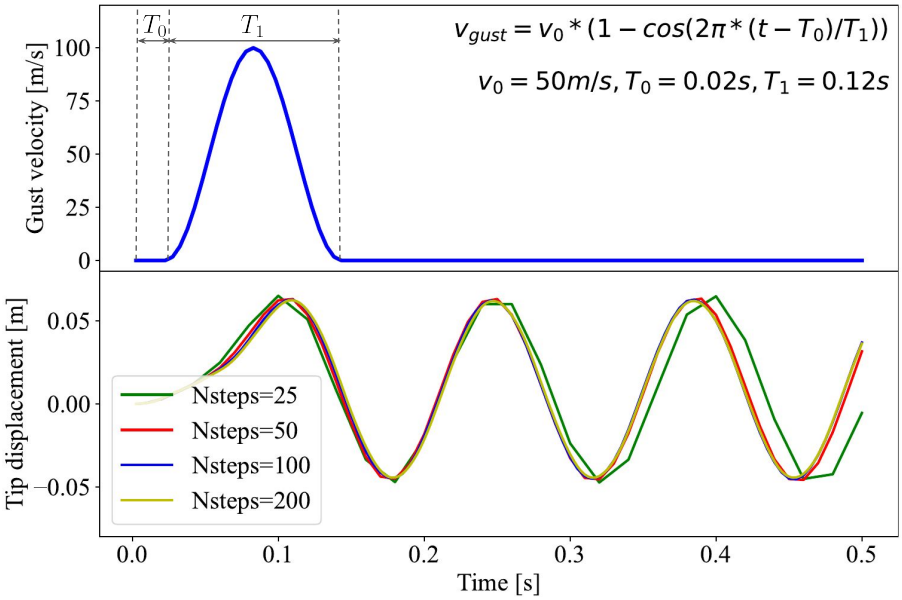


Fig. 14: Numerical results for wind gust response of the eCRM-001 wing model under a sinusoidal gust excitement. The plot on top shows the velocity of the gust over time. The plot on the bottom shows the corresponding aeroelastic response of the wing measured by the tip displacement with different time step sizes.

4.4.6 Sensitivity analysis of compliance with respect to the thickness

The aeroelastic solver is fully integrated with FEMO, allowing us to compute the total derivatives of any specified outputs with respect to design variables automatically using CSDL. The MDO process is shown in Fig. 15.

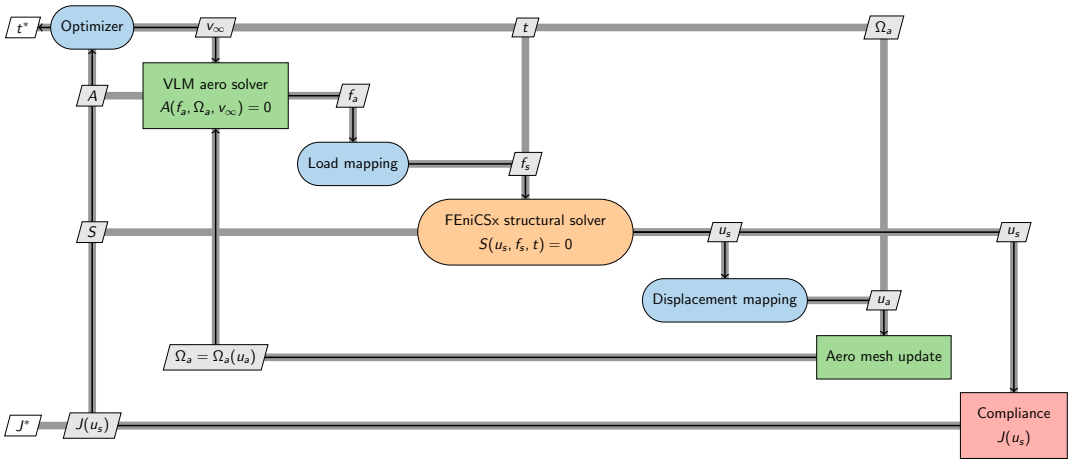


Fig. 15: The MDO process for thickness sensitivity analysis of the eCRM-001 wing structure under aerodynamic loads.

These derivatives are the key to solve any optimization problems involving aeroelasticity efficiently. Here, we demonstrate the capability of automatic adjoint computation for aeroe-

lasticity by performing sensitivity analysis of structural compliance with respect to thickness distribution in a static scenario. It is worth mentioning that while compliance and thickness are structural parameters, the aerodynamic solver is also involved in this derivative computation. This is because the aerodynamic mesh will be affected by structural deformation, leading to changes in the aerodynamic loads, which in turn impact the compliance, computed from the displacements.

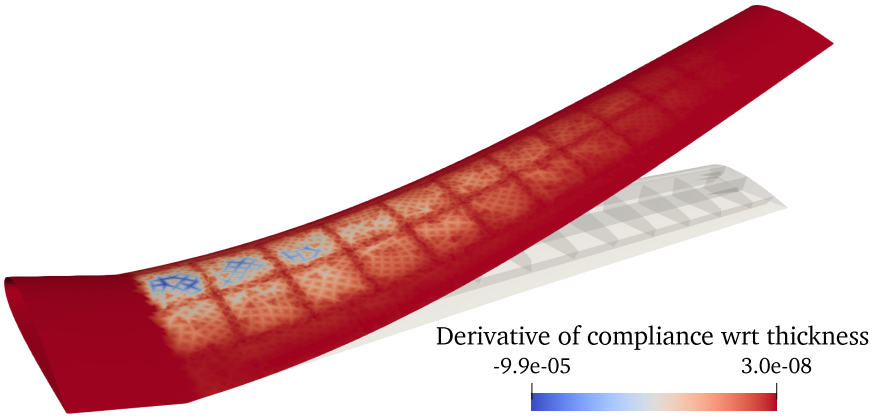


Fig. 16: The gradients of compliance with respect to the thickness distribution represented by first-order continuous function.

5 Conclusion

In this work, we have presented a generalized approach for modeling PDE-constrained MDO problems, and developed the computational tool FEMO, with the goal to minimize the time, effort, and expertise required for development. FEMO leverages the powerful finite element solver FEniCSx to effectively and accurately solve PDEs, and integrates with CSDL, a domain-specific language for MDO modeling, to enable coupling with non-PDE components and automatic adjoint sensitivity analysis.

The effectiveness of this approach has been demonstrated through verification using benchmarking problems, including the optimal control of a nonlinear elliptic PDE and topology optimization of a cantilever beam, both of which have known analytical solutions. Additionally, its capability to tackle real-world problems, such as the design and optimization of electric vertical take-off and landing (eVTOL) aircraft, which motivated this research, has been established.

The shape optimization of an electric motor has converged successfully, which couples two nonlinear PDEs—the hyperelastic equation and Maxwell's equations—to simulate the magnetic flux density field under geometry deformation, and then coupled with non-PDE postprocessing model to compute the power loss as the objective function.

We applied this method to large-scale aeroelastic analysis of an eVTOL wing, where there are thousands of design variables corresponding to node-wise structural thicknesses. Here, the FEniCSx-based structural solver is loosely coupled with the CSDL-based VLM solver through FEMO, and both static and dynamic cases have been discussed. For the static case, we also performed sensitivity analysis for structural compliance of the wing with respect to thickness distribution, which provided qualitatively reasonable results for the gradients. In the dynamic

case, the structural response of the wing under gust excitation was studied, and the wing tip displacement profile achieved convergence through the reduction of the time step size.

While the presented methodology and implementation have shown promising results, there are still some challenges to overcome. Expertise is required for applying case-dependent Dirichlet boundary conditions weakly with Nitsche's method, including formulation, tuning, and testing. Additionally, the efficiency needs to be improved for large-scale problems with a large number of design variables. Potential solutions could be reduced-order modeling, parallel computing, or tuning for nonlinear/linear solvers. For future work, we plan to address the above-mentioned challenges, and extend FEMO to include additional features and applications, such as sensitivity analysis for dynamic aeroelastic models. Another goal could be to incorporate other finite element modules based on the legacy FEniCS to fulfill specific analysis requirements, such as using PENGoLINS for isogeometric analysis with non-matching shells [34].

Acknowledgements

The work presented in this paper is supported by National Aeronautics and Space Administration award number 80NSSC21M0070.

Declarations

Replication of results The source code of FEMO, and the example problems discussed in this work are provided in our open-source GitHub repository (`http://github.com/RuruX/femo`).

Conflict of interest The authors declare that they have no conflict of interest.

References

1. P. A. Nelson, K. Gallagher, I. D. Bloom, and D. W. Dees, "Modeling the performance and cost of lithium-ion batteries for electric-drive vehicles.," Tech. Rep. ANL-11/32, Argonne National Laboratory, 2011.
2. D. Kamensky, "Open-source immersogeometric analysis of fluid–structure interaction using fenics and tigar," *Computers and Mathematics with Applications*, vol. 81, pp. 634–648, 2021.
3. C. Cotter and J. Shipton, "Mixed finite elements for numerical weather prediction," *Journal of Computational Physics*, vol. 231, no. 21, pp. 7076–7091, 2012.
4. Siemens, "NX Nastran 10: Optimization User's Guide." `https://docs.plm.automation.siemens.com/data_services/resources/nxnastran/10/help/en_US/tdocExt/pdf/optimization.pdf`, 2014.
5. G. J. Kennedy and J. R. Martins, "A parallel finite-element framework for large-scale gradient-based design optimization of high-performance structures," *Finite Elements in Analysis and Design*, vol. 87, pp. 56–73, 2014.
6. A. Logg, K.-A. Mardal, G. N. Wells, *et al.*, *Automated Solution of Differential Equations by the Finite Element Method*. Springer, 2012.
7. M. S. Alnæs, A. Logg, K. B. Ølgaard, M. E. Rognes, and G. N. Wells, "Unified form language: A domain-specific language for weak formulations of partial differential equations," *ACM Trans. Math. Softw.*, vol. 40, mar 2014.
8. K. B. Ølgaard and G. N. Wells, "Optimizations for quadrature representations of finite element tensors through automated code generation," *ACM Trans. Math. Softw.*, vol. 37, jan 2010.
9. A. Logg and G. N. Wells, "Dolfin: Automated finite element computing," *ACM Trans. Math. Softw.*, vol. 37, apr 2010.
10. S. K. Mitusch, S. W. Funke, and J. S. Dokken, "dolfin-adjoint 2018.1: automated adjoints for fenics and firedrake," *Journal of Open Source Software*, vol. 4, no. 38, p. 1292, 2019.
11. F. Rathgeber, D. A. Ham, L. Mitchell, M. Lange, F. Luporini, A. T. T. Mcrae, G.-T. Bercea, G. R. Markall, and P. H. J. Kelly, "Firedrake: Automating the finite element method by composing abstractions," *ACM Trans. Math. Softw.*, vol. 43, dec 2016.
12. V. Gandarillas, A. J. Joshy, M. Z. Sperry, A. K. Ivanov, and J. T. Hwang, "A graph-based methodology for constructing computational models that automates adjoint-based sensitivity analysis," *Structural and Multidisciplinary Optimization*, 2022.
13. M. W. Scroggs, J. S. Dokken, C. N. Richardson, and G. N. Wells, "Construction of arbitrary order finite element degree-of-freedom maps on polygonal and polyhedral cell meshes," *ACM Trans. Math. Softw.*, vol. 48, may 2022.

14. F. Pörner and D. Wachsmuth, "Tikhonov regularization of optimal control problems governed by semi-linear partial differential equations," 2017.
15. J. Nitsche, "Über ein Variationsprinzip zur Lösung von Dirichlet-Problemen bei Verwendung von Teilräumen, die keinen Randbedingungen unterworfen sind," *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, vol. 36, pp. 9–15, 1971.
16. C. Annavarapu, M. Hautefeuille, and J. E. Dolbow, "A robust nitsche's formulation for interface problems," *Computer Methods in Applied Mechanics and Engineering*, vol. 225-228, pp. 44–54, 2012.
17. W. Jiang, C. Annavarapu, J. Dolbow, and I. Harari, "A robust nitsche's formulation for interface problems with spline-based finite elements," *International Journal for Numerical Methods in Engineering*, vol. 104, 02 2015.
18. D. Schillinger, I. Harari, M.-C. Hsu, D. Kamensky, S. K. Stoter, Y. Yu, and Y. Zhao, "The non-symmetric nitsche method for the parameter-free imposition of weak boundary and coupling conditions in immersed finite elements," *Computer Methods in Applied Mechanics and Engineering*, vol. 309, pp. 625–652, 2016.
19. M. S. Alnæs, A. Logg, K. B. Ølgaard, M. E. Rognes, and G. N. Wells, "Unified form language: A domain-specific language for weak formulations of partial differential equations," *ACM Trans. Math. Softw.*, vol. 40, mar 2014.
20. P. E. Gill, W. Murray, and M. A. Saunders, "Snopt: An sqp algorithm for large-scale constrained optimization," *SIAM Rev.*, vol. 47, p. 99–131, jan 2005.
21. J. D. Anderson Jr, *Fundamentals of aerodynamics*. Tata McGraw-Hill Education, 2010.
22. N. Arada, E. Casas, and F. Tröltzsch, "Error estimates for the numerical approximation of a semilinear elliptic control problem," *Computational Optimization and Applications*, vol. 23, pp. 201–229, 11 2002.
23. H. Chung, J. T. Hwang, J. S. Gray, and H. A. Kim, "Topology optimization in openmdao," *Structural and multidisciplinary optimization*, vol. 59, no. 4, pp. 1385–1400, 2019.
24. J. S. Gray, J. T. Hwang, J. R. R. A. Martins, K. T. Moore, and B. A. Naylor, "OpenMDAO: An Open-Source Framework for Multidisciplinary Design, Analysis, and Optimization," *Structural and Multidisciplinary Optimization*, vol. 59, pp. 1075–1104, 2019.
25. J. Yan, R. Xiang, D. Kamensky, M. T. Tolley, and J. T. Hwang, "Topology optimization with automated derivative computation for multidisciplinary design problems," *Structural and Multidisciplinary Optimization*, vol. 65, p. 151, Apr 2022.
26. K. Svanberg and H. Svärd, "Density filters for topology optimization based on the pythagorean means," *Structural and Multidisciplinary Optimization*, vol. 48, pp. 859–875, Nov 2013.
27. M. P. Bendsøe and N. Kikuchi, "Generating optimal topologies in structural design using a homogenization method," *Computer Methods in Applied Mechanics and Engineering*, vol. 71, no. 2, pp. 197–224, 1988.
28. G. E. Neighbor, H. Zhao, M. Saraeian, M.-C. Hsu, and D. Kamensky, "Leveraging code generation for transparent immersogeometric fluid–structure interaction analysis on deforming domains," *Engineering with Computers*, vol. 39, pp. 1019–1040, Apr 2023.
29. Y. Bazilevs, V. M. Calo, T. J. R. Hughes, and Y. Zhang, "Isogeometric fluid-structure interaction: theory, algorithms, and computations," *Computational Mechanics*, vol. 43, pp. 3–37, Dec 2008.
30. S. P. C. Van Schie, H. Zhao, Y. Jiayao, X. Ru, J. T. Hwang, and D. Kamensky, "Solver-independent aeroelastic coupling for large-scale multidisciplinary design optimization," in *AIAA Scitech 2023 Forum*, 2023.
31. J. Katz and A. Plotkin, *Low-speed aerodynamics*, vol. 13. Cambridge university press, 2001.
32. E. M. B. Campello, P. M. Pimenta, and P. Wriggers, "A triangular finite shell element based on a fully nonlinear shell formulation," *Computational Mechanics*, vol. 31, pp. 505–518, Aug 2003.
33. J. Bleyer, *Numerical Tours of Computational Mechanics with FEniCS*, 2018.
34. H. Zhao, X. Liu, A. H. Fletcher, R. Xiang, J. T. Hwang, and D. Kamensky, "An open-source framework for coupling non-matching isogeometric shells with application to aerospace structures," *Computers & Mathematics with Applications*, vol. 111, pp. 109–123, 2022.