MatTO: A Modular Framework for Material and Topology Optimization of Stimulus-Responsive Soft Materials

Ian Galloway[1]
Department of Mechanical Engineering,
South Dakota School of Mines and Technology,
Rapid City, SD 57701, USA
e-mail: ian.galloway@mines.sdsmt.edu

Prashant K. Jha[1]
Department of Mechanical Engineering,
South Dakota School of Mines and Technology,
Rapid City, SD 57701, USA
e-mail: pjha.sci@gmail.com;
prashant.jha@sdsmt.edu

Design of stimulus-responsive soft materials plays a fundamental role in applications in soft robotics, sensing, and actuation. Their design space can extend beyond structural topology to include material composition and orientation. For example, in hard-magnetic soft materials (hMSMs), magnetic particle volume fraction and remanent magnetization direction fields can be designed together with the topology field. Our prior work proposed a joint material–structural optimization framework for hMSMs and demonstrated that simultaneous optimization of material and topology can lead to non-intuitive designs while providing additional design freedom. In this work, we show that this approach can be made material-independent and applied to different classes of materials in which the optimization fields represent different physical variables. Specifically, we propose the open-source framework MatTO to provide this generalized design capability. Through eleven numerical examples involving hMSMs, liquid crystal elastomers, and anisotropic and isotropic magneto-active elastomers in two and three dimensions, we demonstrate the applicability of the framework across different material classes and design variables. We discuss the main components of the framework and provide details for defining new problems and extending it to additional material models. The open-source repository is available at `CEADpx/matto`*. A common energy-based constitutive structure is assumed, and the open-source FEniCSx software is used for finite-element analysis. Future extensions will consider plastic deformation, phase-field fracture, and time- and path-dependent models.*

Keywords: Material and topology optimization, soft materials, finite-strain mechanics, adjoint sensitivity, open-source software

1 Introduction

Topology optimization provides a systematic way to distribute material in a design domain so that a structure meets a prescribed objective and a set of resource or performance constraints [1]. The same idea can be extended beyond solid–void design. In stimulus-responsive soft materials, the design may also include the spatial distribution of an active phase, particle concentration, or local material orientation. The response of soft materials is commonly nonlinear, and an applied magnetic field, thermal change, or other stimulus may alter the stress or preferred deformation. A computational design problem must therefore coordinate the constitutive model, finite-element state problem, design-field regularization, sensitivity analysis, optimizer, and the applied loads and stimuli.

Material-specific topology and inverse-design formulations have been developed for several classes of stimulus-responsive soft structures. Examples include joint topology and magnetization design of hard-magnetic soft materials [2], inverse design of magneto-active metasurfaces and robots [3], and topology, director, and shape optimization of liquid crystal elastomers [4–6]. These studies demonstrate the value of optimizing both structural geometry and local material organization. Their computational workflows, however, are generally developed around a particular constitutive model, stimulus, and set of design variables. Reusing them for another material can therefore require changes to the state problem, sensitivity analysis, and optimization code in addition to the material law. This makes it harder to compare formulations, reproduce examples, and identify which parts of a study are general and which are material-specific.

Our prior work developed a joint material–structural optimization framework for hard-magnetic soft materials (hMSMs), in which structural density, magnetic particle volume fraction, and remanent magnetization direction can be optimized simultaneously [7]. That work demonstrated how material and structural variables can be treated together within one nonlinear optimization problem. The present work asks whether the same approach can be made material-independent and applied across stimulus-responsive material classes in which the design fields represent different physical variables.

FEniCSx provides a natural basis for this generalization because it supports variational formulations, symbolic differentiation, finite-element assembly, and parallel numerical solution through UFL, DOLFINx, Basix, and PETSc [8–11]. FEniTop builds on FEniCSx to provide a modular topology-optimization framework with filtering, sensitivity analysis, MMA updates, and parallel execution [12]. Its linear-elastic 2D and 3D implementation provides the software starting point for the present work.

MatTO extends this foundation from linear-elastic density optimization to nonlinear, energy-based material and topology optimization of stimulus-responsive soft structures. The central idea is to separate the material-specific free-energy model from the common state, sensitivity, and optimization machinery. A material model declares the design fields and stimuli that it requires and supplies a free-energy density, while the problem definition specifies the mesh, active or prescribed fields, loads, objective, constraints, and numeri-

[1]Corresponding Author.

cal settings. The same optimization procedure can therefore operate on different numbers and types of design fields without embedding material-specific equations in the optimization algorithm.

The examples included with MatTO use up to three scalar fields: structural density ρ, material composition or active-material distribution ϕ, and local orientation θ. Their physical meanings depend on the selected material model. In hMSMs [7], ϕ denotes magnetic particle volume fraction and θ the remanent-magnetization direction, restricted to a fixed plane in the three-dimensional example. In anisotropic magneto-active elastomers (MAEs) and liquid crystal elastomers (LCEs), θ instead represents particle-chain or mesogen orientation [6,13]. An isotropic magneto-active elastomer [14] uses no orientation field. Each field can be optimized or prescribed and can have its own bounds, initialization, filtering, projection, and fixed-region settings.

The hMSM formulation builds on our earlier joint material–structural optimization work [7]. The other implementations are based on the anisotropic magnetorheological-elastomer model of Akbari and Khajehsaeid [13], the isotropic magneto-active-polymer model of Garai and Haldar [14], and the liquid-crystal-elastomer model of Barrera et al. [6]. These models are used to demonstrate that the same optimization framework can operate across different free-energy functions, stimuli, and active design fields. The present work does not propose new constitutive models for these material classes.

The contributions of this paper are:

(i) a material-independent nonlinear finite-element and optimization framework for stimulus-responsive soft materials;
(ii) a generic representation of active and prescribed design fields whose physical meaning is determined by the material model rather than fixed by the optimization algorithm;
(iii) an energy-based material interface coupled to automatic stress construction, adjoint sensitivity analysis, design-field parameterization, and MMA updates; and
(iv) eleven numerical examples in two and three dimensions demonstrating the framework across different material classes, active design fields, load cases, and objectives.

The remainder of the paper is organized as follows. Section 2 defines the optimization problem and computational workflow. Section 3 describes the software architecture and implementation. Section 4 presents the material families and numerical examples. Section 5 explains how new problems and material models are introduced, and Section 6 describes installation, testing, and verification. Section 7 summarizes the present scope and directions for extension. The source is available at https://github.com/CEADpx/matto.

2 Problem definition and computational procedure

MatTO considers a design problem in which one or more spatial material fields are optimized subject to nonlinear equilibrium. The formulation therefore connects the physical material response, the optimizer-controlled design variables, and the mappings between them. This section defines these quantities and the resulting computational procedure. Software objects and implementation details are deferred to Section 3.

2.1 Modeling assumptions and kinematics.

All material families considered here are modeled under the following assumptions.

A1. Quasistatic, conservative response. Deformation is treated using finite-strain continuum mechanics. Inertia, viscosity, dissipation, and material history are not represented. The incremental loading used by the nonlinear solver is a numerical continuation procedure, not a discretization of physical time.

A2. Reference-configuration description. Free-energy density, body force, traction, and equilibrium are expressed with respect to the undeformed configuration. The resulting formulation is total Lagrangian and uses the first Piola–Kirchhoff stress.

A3. Prescribed mechanical loads and stimuli. Each load case prescribes its body force, boundary tractions, and scalar or vector stimuli. These quantities are treated as design-independent prescribed data. MatTO does not solve an additional electromagnetic, thermal, or optical field equation; any such stimulus is supplied to the constitutive free-energy model.

A4. Effective continuum representation. Each material is represented by a continuum free-energy density. Constituents and microstructure are not discretized as separate finite-element domains; their local influence enters through constitutive parameters and physical design fields.

The examples considered here use two- and three-dimensional reference domains. The two-dimensional examples use plane strain; when required by a constitutive model, the in-plane deformation gradient is embedded in three dimensions within the material model. Three-dimensional deformation and joint material–structural design are demonstrated for the hMSM family. Let $\Omega_0 \subset \mathbb{R}^d$, $d = 2, 3$, be the reference configuration. For load case $\ell = 1, \dots, N_\ell$, the motion $\boldsymbol{\chi}^{(\ell)} : \Omega_0 \to \Omega^{(\ell)}$ maps a material point $\boldsymbol{X}$ to its current position $\boldsymbol{x}^{(\ell)}$. The displacement and deformation gradient are

$$\boldsymbol{u}^{(\ell)}(\boldsymbol{X}) = \boldsymbol{\chi}^{(\ell)}(\boldsymbol{X}) - \boldsymbol{X}, \qquad \boldsymbol{F}^{(\ell)} = \nabla_{\boldsymbol{X}} \boldsymbol{\chi}^{(\ell)} = \boldsymbol{I} + \nabla_{\boldsymbol{X}} \boldsymbol{u}^{(\ell)}.$$

For each material model, the design-dependent free-energy density is

$$\widehat{W}^{(\ell)} = \widehat{W}\left(\boldsymbol{X}, \boldsymbol{F}^{(\ell)}, \boldsymbol{q}, \boldsymbol{s}^{(\ell)}; \boldsymbol{p}\right), \tag{1}$$

where $\boldsymbol{q} = (q_1, \dots, q_{N_d})$ contains the physical design fields, $\boldsymbol{s}^{(\ell)}$ contains the prescribed stimuli for case ℓ, and $\boldsymbol{p}$ denotes material parameters. A hat is used to emphasize design dependence. The first Piola–Kirchhoff stress is

$$\widehat{\boldsymbol{P}}^{(\ell)} = \frac{\partial \widehat{W}^{(\ell)}}{\partial \boldsymbol{F}^{(\ell)}}.$$

UFL [9] evaluates this derivative symbolically. The stress therefore need not be specified separately from the free energy. The framework does not prescribe a universal form of $\widehat{W}$: density, composition, orientation, and stimulus enter only through the selected material model. Table 1 summarizes the notation used in the common continuum and optimization formulation.

2.2 Strong and weak equilibrium problems.

Given the physical fields $\boldsymbol{q}$, prescribed stimuli $\boldsymbol{s}^{(\ell)}$, and mechanical loading, the strong problem for load case ℓ is to find $\boldsymbol{u}^{(\ell)}$ such that

$$\begin{aligned} \nabla_{\boldsymbol{X}} \cdot \widehat{\boldsymbol{P}}^{(\ell)} + \boldsymbol{b}^{(\ell)} &= \boldsymbol{0}, && \boldsymbol{X} \in \Omega_0, \\ \boldsymbol{u}^{(\ell)} &= \bar{\boldsymbol{u}}, && \boldsymbol{X} \in \Gamma_u, \\ \widehat{\boldsymbol{P}}^{(\ell)} \boldsymbol{n} &= \bar{\boldsymbol{t}}^{(\ell)}, && \boldsymbol{X} \in \Gamma_t := \partial\Omega_0 \setminus \Gamma_u. \end{aligned}$$

The prescribed traction is zero on portions of Γ_t on which no traction is specified. The trial and test spaces are

$$V_u^{(\ell)} = \left\{ \boldsymbol{u} \in [H^1(\Omega_0)]^d : \boldsymbol{u} = \bar{\boldsymbol{u}} \text{ on } \Gamma_u \right\},$$

$$V_v = \left\{ \delta\boldsymbol{u} \in [H^1(\Omega_0)]^d : \delta\boldsymbol{u} = \boldsymbol{0} \text{ on } \Gamma_u \right\}.$$

For $\boldsymbol{u}^{(\ell)} \in V_u^{(\ell)}$ and $\delta\boldsymbol{u} \in V_v$, define the residual functional

$$\widehat{\mathcal{R}}^{(\ell)}\left(\boldsymbol{u}^{(\ell)}, \delta\boldsymbol{u}; \boldsymbol{q}, \boldsymbol{s}^{(\ell)}, \boldsymbol{b}^{(\ell)}, \bar{\boldsymbol{t}}^{(\ell)}, \bar{\boldsymbol{u}}\right) :=$$

$$\int_{\Omega_0} \widehat{\boldsymbol{P}}^{(\ell)} : \nabla_{\boldsymbol{X}} \delta\boldsymbol{u} \, \mathrm{d}V - \int_{\Omega_0} \boldsymbol{b}^{(\ell)} \cdot \delta\boldsymbol{u} \, \mathrm{d}V - \int_{\Gamma_t} \bar{\boldsymbol{t}}^{(\ell)} \cdot \delta\boldsymbol{u} \, \mathrm{d}S. \tag{2}$$

Table 1 Notation used in the common continuum and optimization formulation.

Symbol	Description
$\Omega_0, \Omega^{(\ell)}$	Reference and current configurations for load case ℓ.
$\boldsymbol{X}, \boldsymbol{x}^{(\ell)}$	Material point in Ω_0 and its current position.
$\chi^{(\ell)}, \boldsymbol{u}^{(\ell)}$	Motion and displacement fields for load case ℓ.
$\boldsymbol{F}^{(\ell)}$	Deformation gradient, $\boldsymbol{I} + \nabla_{\boldsymbol{X}} \boldsymbol{u}^{(\ell)}$.
$\widehat{W}^{(\ell)}, \widehat{\boldsymbol{P}}^{(\ell)}$	Design-dependent free-energy density and first Piola–Kirchhoff stress.
$\boldsymbol{b}^{(\ell)}, \bar{\boldsymbol{t}}^{(\ell)}, \bar{\boldsymbol{u}}$	Prescribed body force, traction, and displacement; the present interface uses common displacement data for all cases.
$\Gamma_u, \Gamma_t, \boldsymbol{n}$	Displacement boundary, traction boundary, and outward unit normal in the reference configuration.
$\boldsymbol{s}^{(\ell)}, \boldsymbol{p}$	Prescribed stimulus data for case ℓ and material parameters.
$z_i, \widetilde{z}_i, q_i$	Raw, filtered, and physical representations of design field i.
Q_h, V_h^{HH}	Raw design space and continuous Helmholtz-filter space used by the included examples.
$\mathcal{H}_i, \mathcal{P}_i, \mathcal{T}_i$	Helmholtz filter, optional Heaviside projection, and complete raw-to-physical map.
r_i, β_i, η	Filter radius, projection sharpness, and projection threshold.
$\widehat{\mathcal{R}}^{(\ell)}, \widehat{\boldsymbol{r}}^{(\ell)}$	Weak and finite-dimensional equilibrium residuals.
$N_d, \mathcal{Z}_h$	Number of active design fields and their product raw-design space.
$f^{(\ell)}, w_\ell, f$	Load-case objective, case weight, and weighted aggregate objective.
C_j, N_j, c_j, g_j	Constrained measure, fixed normalization, admissible normalized limit, and dimensionless constraint residual.
$\boldsymbol{\lambda}^{(\ell)}$	Adjoint field for load case ℓ.

The weak problem is to find $\boldsymbol{u}^{(\ell)} \in V_u^{(\ell)}$ such that

$$\widehat{\mathcal{R}}^{(\ell)}\left(\boldsymbol{u}^{(\ell)}, \delta\boldsymbol{u}; \boldsymbol{q}, \boldsymbol{s}^{(\ell)}, \boldsymbol{b}^{(\ell)}, \bar{\boldsymbol{t}}^{(\ell)}, \bar{\boldsymbol{u}}\right) = 0, \qquad \forall \delta\boldsymbol{u} \in V_v. \tag{3}$$

This defines the continuum variational problem. Section 3.5 describes its finite-element discretization and solution by load incrementation and Newton iterations.

2.3 Design-field parameterization and energy interpolation. Topology and material optimization require a distinction between the optimizer-controlled variables and the physical fields entering the constitutive model. If every element value were used directly and independently, neighboring elements could alternate between bounds or develop variations on the scale of a single element. Such patterns can be difficult to interpret physically and may change with mesh refinement. MatTO therefore denotes the optimizer-controlled quantity as the *raw field* and optionally maps it through regularization operators to a *physical field*. The material energy, objective, and constraints are evaluated using the physical fields.

Let N_d be the number of active scalar design fields. For field i, z_i denotes its raw representation, $\widetilde{z}_i$ its filtered representation, and q_i its final physical representation. In the examples, the raw fields are piecewise constant, giving one bounded design coefficient per element before regularization. Let $\{\Omega_e\}_{e=1}^{N_e}$ be a finite-element partition of Ω_0. The corresponding piecewise-constant raw space is

$$Q_h = \left\{ z \in L^2(\Omega_0) : z|_{\Omega_e} \in \mathbb{P}_0(\Omega_e), \quad e = 1, \ldots, N_e \right\}. \tag{4}$$

For the examples considered here, the collection of active raw fields therefore belongs to the product space

$$\boldsymbol{z} = (z_1, \ldots, z_{N_d}) \in \mathcal{Z}_h := Q_h^{N_d}.$$

More generally, each field may use a different compatible raw and physical finite-element space. Equations (4)–(6) describe the filtered fields used in the examples. The 3D hMSM orientation field instead uses the same piecewise-constant space for its raw and physical representations, with the identity map.

The Helmholtz filter suppresses element-scale oscillations and introduces a characteristic spatial length scale controlled by r_i. For a raw field $z_i \in Q_h$ and radius $r_i > 0$, define the filter operator

$$\mathcal{H}_i(\,\cdot\,; r_i) : Q_h \to V_h^{\mathrm{HH}}, \qquad \widetilde{z}_i = \mathcal{H}_i(z_i; r_i),$$

where $\widetilde{z}_i \in V_h^{\mathrm{HH}}$ satisfies

$$\int_{\Omega_0} \left(r_i^2 \nabla\widetilde{z}_i \cdot \nabla v + \widetilde{z}_i v \right) \mathrm{d}V = \int_{\Omega_0} z_i v \, \mathrm{d}V, \qquad \forall v \in V_h^{\mathrm{HH}}. \tag{5}$$

Because the filter equation contains $\nabla\widetilde{z}_i$, its solution belongs to an H^1-conforming space. For the first-order Lagrange interpolation used here,

$$V_h^{\mathrm{HH}} = \left\{ v \in H^1(\Omega_0) : v|_{\Omega_e} \in \mathcal{L}_1(\Omega_e), \quad e = 1, \ldots, N_e \right\}. \tag{6}$$

No essential boundary condition is imposed on $\widetilde{z}_i$, giving the natural homogeneous Neumann condition $\nabla\widetilde{z}_i \cdot \boldsymbol{n} = 0$ on $\partial\Omega_0$ [15]. Filtering regularizes a field but does not force intermediate values toward either bound. When a nearly discrete void–solid density is desired, a smooth Heaviside projection can additionally sharpen the filtered field while retaining differentiability. The pointwise projection is

$$q_i = \mathcal{P}_i(\widetilde{z}_i; \beta_i, \eta) := \frac{\tanh(\beta_i\eta) + \tanh[\beta_i(\widetilde{z}_i - \eta)]}{\tanh(\beta_i\eta) + \tanh[\beta_i(1 - \eta)]}, \tag{7}$$

where $\eta = 0.5$ is fixed. In the above, larger β_i produces a sharper transition about the threshold η. A continuation schedule increases β_i gradually so that the early optimization remains smooth before a nearly binary density is promoted [16]. The complete raw-to-physical map for an active field is therefore

$$q_i = \mathcal{T}_i(z_i) = \begin{cases} \mathcal{P}_i(\mathcal{H}_i(z_i; r_i); \beta_i, \eta), & \text{filter and projection,} \\ \mathcal{H}_i(z_i; r_i), & \text{filter only,} \\ z_i, & \text{no operator.} \end{cases} \tag{8}$$

For structural density, this convention gives

$$\widetilde{\rho} = \mathcal{H}_\rho(\rho; r_\rho), \qquad \bar{\rho} = \mathcal{P}_\rho(\widetilde{\rho}; \beta_\rho, \eta),$$

where ρ, $\widetilde{\rho}$, and $\bar{\rho}$ are the raw, filtered, and physical density fields. Composition and orientation fields may use filtering without projection because intermediate composition and orientation values can be physically meaningful. Each field may have its own filter radius and bounds. An inactive field is assigned a prescribed physical value and is excluded from the optimization. Section 3.4 describes how the same operators are applied to fields with different physical roles.

Remark: Fixed regions and physical fields. When a field is fixed on a set of cells, their raw coefficients are prescribed and excluded from MMA. Filtering uses the full raw field, including these coefficients. After applying $\mathcal{T}_i$, the prescribed values are imposed on the physical degrees of freedom of the fixed cells, including shared nodes for continuous fields.

The fixed physical values are independent of the free raw coefficients. The sensitivities of the objective and constraints at the fixed physical entries are therefore set to zero before applying the transpose operators. MMA uses the resulting derivatives with respect to the free raw coefficients. Resource integrals include the fixed regions.

Remark: Constitutive interpolation. Filtering and projection should be distinguished from constitutive interpolation. The design-field parameterization maps raw fields to physical fields, but the framework does not impose a universal SIMP law or multiply the entire energy by density. Instead, any interpolation or penalization is defined within the material model through $\widehat{W}$. For example, the material energy may use

$$s_\rho(\bar{\rho}) = \varepsilon_\rho + (1-\varepsilon_\rho)\bar{\rho}^{p_\rho}, \qquad \varepsilon_\rho \geq 0, \tag{9}$$

to retain a small elastic stiffness floor while using $\bar{\rho}$ directly in a stimulus-coupling term for which no floor is desired. A material model may set $\varepsilon_\rho = 0$, omit density from selected terms, or use another interpolation entirely. This placement within $\widehat{W}$ is important for stimulus-responsive materials because different contributions to the same energy need not share the same interpolation or penalization. Section 3.5 illustrates this distinction using the hMSM material energy.

2.4 Optimization problem. The raw fields $\boldsymbol{z} = (z_1, \ldots, z_{N_d}) \in \mathcal{Z}_h$ constitute the optimization variables. After spatial discretization, their free coefficients form the finite-dimensional design vector supplied to MMA. We use $\boldsymbol{z}$ for both the collection of discrete fields and the corresponding coefficient vector when the distinction is clear from context. Their physical counterparts are obtained from

$$\boldsymbol{q} = \mathcal{T}(\boldsymbol{z}) := \big(\mathcal{T}_1(z_1), \ldots, \mathcal{T}_{N_d}(z_{N_d})\big) = \big(q_1, \ldots, q_{N_d}\big).$$

Here, $\mathcal{T}$ denotes the collection of field-wise maps defined in Eq. (8). Any inactive physical fields required by the material model are prescribed separately. Substitution of $\boldsymbol{q} = \mathcal{T}(\boldsymbol{z})$ into Eq. (1) defines the design-dependent stress and weak residual for every load case. After finite-element discretization, Eq. (3) gives

$$\widehat{\boldsymbol{r}}^{(\ell)}\Big(\boldsymbol{u}^{(\ell)}; \boldsymbol{q}, \boldsymbol{s}^{(\ell)}, \boldsymbol{b}^{(\ell)}, \bar{\boldsymbol{t}}^{(\ell)}, \bar{\boldsymbol{u}}\Big) = \boldsymbol{0}, \qquad \ell = 1, \ldots, N_\ell.$$

For N_g constraints, the resulting optimization problem is

$$\begin{aligned} \underset{\boldsymbol{z}}{\text{minimize}} \quad & f(\boldsymbol{z}) = \sum_{\ell=1}^{N_\ell} w_\ell f^{(\ell)}\Big(\boldsymbol{u}^{(\ell)}(\mathcal{T}(\boldsymbol{z})), \mathcal{T}(\boldsymbol{z})\Big), \\ \text{subject to} \quad & \begin{cases} \widehat{\boldsymbol{r}}^{(\ell)} = \boldsymbol{0}, & \ell = 1, \ldots, N_\ell, \\ g_j(\boldsymbol{q}) \leq 0, & j = 1, \ldots, N_g, \\ \boldsymbol{z}_{\min} \leq \boldsymbol{z} \leq \boldsymbol{z}_{\max}. \end{cases} \end{aligned} \tag{10}$$

Here $f^{(\ell)}$ is the objective for load case ℓ, and w_ℓ is its prescribed weight. Writing $\mathcal{T}(\boldsymbol{z})$ explicitly in Eq. (10) makes the dependence on the raw fields explicit: each field affects the objective through the parameterization map and may influence it both directly and indirectly through the equilibrium displacement. The same objective definition is evaluated for each load case under its prescribed loads and stimuli. The weighted objectives and their gradients are summed over the load cases before the design update. The formulation includes the single-case limit $N_\ell = 1$ and any active subset of the declared design fields; inactive fields remain prescribed and do not appear in $\boldsymbol{z}$.

Each physical design constraint begins with an inequality of the form

$$\frac{C_j(\boldsymbol{q})}{N_j} \leq c_j, \qquad N_j > 0, \quad c_j > 0, \tag{11}$$

where C_j is the constrained integral or other scalar measure, N_j is a fixed normalization with the same physical units, and c_j is the admissible normalized limit. The dimensionless constraint supplied to MMA is

$$g_j(\boldsymbol{q}) = \frac{C_j(\boldsymbol{q})}{N_j c_j} - 1 \leq 0. \tag{12}$$

For example, limiting the mean physical density to a prescribed volume fraction $v_{\max}$ corresponds to

$$C_\rho(\boldsymbol{q}) = \int_{\Omega_0} \bar{\rho}\ \mathrm{d}V, \qquad N_\rho = \int_{\Omega_0} 1\ \mathrm{d}V, \qquad c_\rho = v_{\max},$$

and hence

$$g_\rho(\boldsymbol{q}) = \frac{\int_{\Omega_0} \bar{\rho}\ \mathrm{d}V}{v_{\max} \int_{\Omega_0} 1\ \mathrm{d}V} - 1 \leq 0.$$

The same construction defines constraints on active-material fraction or other design-dependent scalar measures. The present formulation considers constraints that depend directly on the physical design fields but not on the equilibrium displacement. The normalization N_j is independent of both the displacement and the physical design fields. A displacement-dependent constraint would require a separate adjoint problem for its total derivative and is not considered in the present formulation.

Remark: Phase-fraction constraints. For the MAE bridge and LCE pusher examples, the active-phase constraint is imposed relative to the retained material. Define $\langle v \rangle = |\Omega_0|^{-1} \int_{\Omega_0} v\ \mathrm{d}V$. To limit the active phase within retained material to α, these examples impose

$$v_\phi = \big\langle \bar{\rho}\bar{\phi} + \alpha(1-\bar{\rho}) \big\rangle \leq \alpha,$$

$$g_\phi = v_\phi/\alpha - 1 \leq 0.$$

The corresponding density-weighted phase fraction is

$$a_\phi = \frac{\langle \bar{\rho}\bar{\phi} \rangle}{\langle \bar{\rho} \rangle} = \alpha + \frac{v_\phi - \alpha}{\langle \bar{\rho} \rangle}, \qquad \langle \bar{\rho} \rangle > 0. \tag{13}$$

Thus $v_\phi \leq \alpha$ is equivalent to $a_\phi \leq \alpha$, although the two quantities have different numerical values. The hMSM examples instead constrain $\langle \bar{\phi} \rangle$ directly and do not impose a constraint on $\langle \bar{\rho}\bar{\phi} \rangle$.

2.5 Adjoint sensitivities. MMA requires the derivative of the objective with respect to every design coefficient. A physical field q_i can affect a load-case objective directly and indirectly through the equilibrium displacement. For a converged load case, the total derivative is

$$\frac{\mathrm{d}f^{(\ell)}}{\mathrm{d}q_i} = \left.\frac{\partial f^{(\ell)}}{\partial q_i}\right|_{\boldsymbol{u}^{(\ell)}} + \left(\frac{\partial \boldsymbol{u}^{(\ell)}}{\partial q_i}\right)^T \frac{\partial f^{(\ell)}}{\partial \boldsymbol{u}^{(\ell)}}. \tag{14}$$

The first term is the direct dependence of the objective on q_i. The second contains the state sensitivity $\partial \boldsymbol{u}^{(\ell)}/\partial q_i$. Direct differentiation of the discrete equilibrium equation $\widehat{\boldsymbol{r}}^{(\ell)} = \boldsymbol{f}_{\text{int}}^{(\ell)} - \boldsymbol{f}_{\text{ext}}^{(\ell)} = \boldsymbol{0}$

gives, because the prescribed external loads are independent of displacement and design,

$$\boldsymbol{K}_T^{(\ell)} = \frac{\partial \widetilde{\boldsymbol{r}}^{(\ell)}}{\partial \boldsymbol{u}^{(\ell)}} = \frac{\partial \boldsymbol{f}_{\mathrm{int}}^{(\ell)}}{\partial \boldsymbol{u}^{(\ell)}}. \tag{15}$$

The displacement sensitivity then satisfies

$$\boldsymbol{K}_T^{(\ell)} \frac{\partial \boldsymbol{u}^{(\ell)}}{\partial q_i} = -\frac{\partial \boldsymbol{f}_{\mathrm{int}}^{(\ell)}}{\partial q_i}. \tag{16}$$

Solving Eq. (16) for every coefficient of every active field would require one linear solve per design coefficient. The adjoint method instead requires one transposed solve for the scalar load-case objective [1,12]:

$$\left(\boldsymbol{K}_T^{(\ell)}\right)^T \boldsymbol{\lambda}^{(\ell)} = -\frac{\partial f^{(\ell)}}{\partial \boldsymbol{u}^{(\ell)}}. \tag{17}$$

The adjoint values are zero at displacement degrees of freedom fixed by the Dirichlet conditions. Combining Eqs. (16) and (17) with Eq. (14) gives the physical-field sensitivity

$$\frac{\mathrm{d}f^{(\ell)}}{\mathrm{d}q_i} = \left.\frac{\partial f^{(\ell)}}{\partial q_i}\right|_{\boldsymbol{u}^{(\ell)}} + \left(\frac{\partial \boldsymbol{f}_{\mathrm{int}}^{(\ell)}}{\partial q_i}\right)^T \boldsymbol{\lambda}^{(\ell)}. \tag{18}$$

The first term is the direct effect of q_i on the objective, while the second accounts for the change in the equilibrium state. The same adjoint $\boldsymbol{\lambda}^{(\ell)}$ is used in Eq. (18) for every active physical field. MatTO therefore requires at most one objective-adjoint solve per load case rather than one state-sensitivity solve per design coefficient. If the objective does not depend on displacement, the right-hand side of Eq. (17) vanishes and the adjoint solve is unnecessary.

Because the constraints considered here are state independent, their sensitivities require only direct differentiation. A state-dependent constraint would require its own adjoint solve. The load-case gradients are weighted and summed according to Eq. (10). The resulting physical-field gradient is then mapped to the raw-field gradient through the corresponding parameterization:

$$\frac{\mathrm{d}f}{\mathrm{d}z_i} = \left(\frac{\partial \mathcal{T}_i(z_i)}{\partial z_i}\right)^T \sum_{\ell=1}^{N_\ell} w_\ell \frac{\mathrm{d}f^{(\ell)}}{\mathrm{d}q_i}. \tag{19}$$

For a filtered and projected field, this mapping applies the projection derivative followed by the adjoint of the Helmholtz filter. Section 3.6 describes the assembly and solution of Eqs. (15)–(18), and Section 3.4 describes the mapping in Eq. (19).

2.6 Computational procedure for one optimization iteration. With the equilibrium problem, optimization problem, and sensitivity calculation defined, one iteration consists of five material-independent steps:

(1) **Apply the design-field parameterization.** Each z_i is mapped through $\mathcal{T}_i$ to obtain the physical field q_i used by the energy, objective, and constraints. Any projection-continuation parameter is updated according to its prescribed schedule.
(2) **Solve the equilibrium problems.** For each load case, the mechanical loads and stimuli are increased from zero to their target values over a prescribed number of increments. Nonlinear equilibrium is solved after each increment. This load incrementation improves convergence and does not represent a discretization of physical time, consistent with assumption A1.
(3) **Evaluate the objective, constraints, and physical-field sensitivities.** At each converged target state, the load-case objective and its derivatives are evaluated using Eqs. (17) and (18). The state-independent constraints and their direct derivatives are evaluated at the same design.
(4) **Map sensitivities to the raw fields and aggregate the load cases.** Each physical-field gradient is mapped to its corresponding raw field using Eq. (19). The objective values and gradients are multiplied by w_ℓ and summed over the load cases.
(5) **Update the design.** MMA updates the free raw coefficients using the corresponding derivatives, bounds, normalized constraint residuals, and update history. These coefficients define the raw fields for the next iteration together with the fixed values.

The iterations terminate when the iteration limit is reached or the prescribed design-change criterion is satisfied after completion of all projection-continuation schedules. The final design is then mapped to the physical fields and every load case is solved once more so that the stored states and reported scalar quantities correspond to the final design.

Remark: Stopping criterion. The design change is $\max_j \frac{|z_j^{\mathrm{new}} - z_j|}{z_j^{\max} - z_j^{\min}}$, with the maximum taken over the free raw coefficients on all MPI ranks. The MMA move limit uses the same bound width. Feasibility is assessed separately on the final design.

Section 3 traces these five steps through the design, state, sensitivity, orchestration, and optimization modules.

3 Software architecture and implementation

MatTO separates the material model and problem definition from the finite-element, sensitivity, and optimization procedures shared across material families. The shared implementation resides in `src/matto`, the constitutive models in `src/matto/materials`, and individual studies in the `examples` directory. A material model defines the free-energy density and the fields and stimuli on which it depends, while an input script specifies the mesh, active or prescribed design fields, boundary conditions, load cases, objective, constraints, and numerical settings. The shared state, sensitivity, and optimization procedures therefore remain independent of the specific hMSM, MAE, or LCE constitutive equations.

Figure 1 summarizes the resulting computational workflow. The material and problem definitions are collected in a `problem` dictionary. The `OptimizationDriver` constructs the finite-element state and sensitivity problems from these definitions and coordinates the repeated parameterization, equilibrium, differentiation, and design-update steps. The raw design fields are first mapped to physical fields and used in the equilibrium solves and objective and constraint evaluations. The resulting physical-field sensitivities are then mapped to the raw design variables before the MMA update.

Table 2 summarizes the current package structure. Material-specific physics enters through `material.energy`, while the problem definition supplies the objective, constraints, loading, and prescribed stimuli. The remaining modules operate on these definitions without containing equations specific to a particular material family.

The following subsections trace a study from its problem definition through state solution, sensitivity evaluation, and design update. The listings illustrate the principal interfaces and their connection to the mathematical formulation in Section 2; they are not intended to reproduce the complete input files or package source.

3.1 Defining a study in an input script. An input script defines one optimization problem. It constructs the mesh and boundary markers, specifies the design variables, defines the load and stimulus cases, and supplies the objective and constraints. The restorative hMSM beam in `examples/hMSM/input_beam.py` is used here to illustrate the interface.

The design-field definitions correspond directly to the parameterization in Section 2.3. Each field is declared as active or prescribed and is assigned its initialization, bounds, raw and physical finite-element spaces, parameterization operators, and any fixed regions.

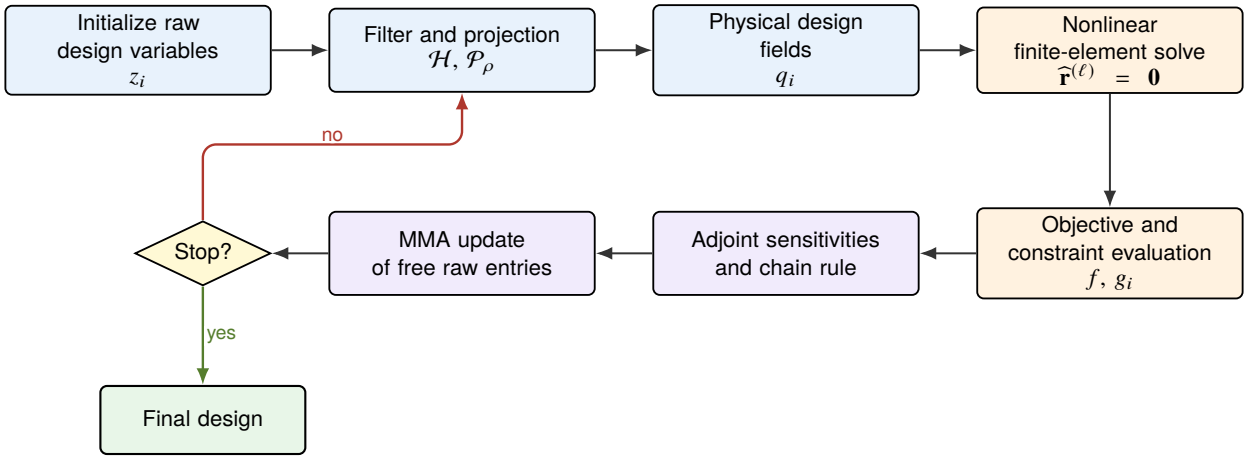


Fig. 1 MatTO optimization workflow. The material model and problem definition supply the physical and numerical information required by the shared design-field parameterization, nonlinear equilibrium, sensitivity evaluation, and MMA update procedures.

Table 2 Responsibilities of the principal modules in `src/matto`.

Module	Role
`materials/`	Defines the `Material` interface, constitutive free energies, shared kinematic operations, and material-consistency checks.
`driver.py`	Coordinates design-field parameterization, load-case solutions, sensitivity evaluation, MMA updates, stopping checks, and output.
`design.py`	Represents the design fields, applies the raw-to-physical maps, and maps physical-field sensitivities to the raw variables.
`state.py`	Defines the displacement problem, prescribed loads and stimuli, free energy, equilibrium residual, and nonlinear solution for each load case.
`sensitivity.py`	Constructs and assembles the required UFL derivatives, solves the objective adjoint problem, and evaluates sensitivities for the active fields.
`mma.py`	Updates the discrete design variables using the method of moving asymptotes.
`utility.py`	Provides nonlinear-solver configuration, MPI communication utilities, and distributed-field gathering.
`postprocess.py`	Provides iteration hooks, history output, field snapshots, and plotting without altering the optimization equations.

Listing 1 shows the distinction between the structural-density and particle-fraction fields in the beam problem. Density `rho` is filtered and projected because a nearly binary structural field is desired, whereas particle fraction `phi` is filtered without projection because intermediate concentrations are physically meaningful. The orientation field `theta` follows the latter construction with bounds $[-2\pi, 2\pi]$.

```
design_variables = {
    "rho": {
        "active": True,
        "initial": 0.50,
        "bounds": (0.05, 1.00),
        "prescribed_value": 1.00,
        "raw_space": ("DG", 0),
        "physical_space": ("CG", 1),
        "operators": [
            {"type": "density_filter", "radius": 1.0},
            {"type": "heaviside", "beta_initial": 1.0,
             "beta_update_interval": 25, "beta_max": 4.0},
        ],
        "fixed_regions": [],
    },
    "phi": {
        "active": True,
        "initial": 0.10,
        "bounds": (0.00, 0.30),
        "prescribed_value": 0.00,
        "raw_space": ("DG", 0),
        "physical_space": ("CG", 1),
        "operators": [
            {"type": "density_filter", "radius": 1.0},
        ],
        "fixed_regions": [],
    },
}
```

Listing 1: Representative design-field definitions for the restorative beam.

Boundary definitions identify the support and traction regions, and each load case assigns values to the prescribed mechanical loading and material stimuli. Stimulus names are treated generically by the shared workflow and interpreted only by the selected material model. For the hMSM beam, the material reads the applied magnetic flux density `B_app`. Another material may instead require a different scalar or vector stimulus. Listing 2 shows the two load cases used by the beam problem. Mechanical loads and stimuli are increased together over `load_steps`, and `weight` specifies w_ℓ in Eq. (10).

```
load_steps = 50
load_cases = [
    {"name": "traction_down_B_up", "weight": 1.0,
     "tractions": {"out_right": (0.0, -0.50)},
     "stimuli": {"B_app": (0.0, 25.0)}},
    {"name": "traction_up_B_down", "weight": 1.0,
     "tractions": {"out_right": (0.0, 0.50)},
     "stimuli": {"B_app": (0.0, -25.0)}},
]

def build_objective(u_field, external_work, dx):
    return external_work
```

Listing 2: Load cases and objective for the restorative beam.

For this example, minimizing external work gives the compliance objective under the prescribed traction. The same interface also permits displacement-based, tracking, or domain-integral objectives constructed from the state and other problem data.

Constraints are defined independently of the objective and follow the normalized form introduced in Eqs. (11) and (12). For each constraint, `form` defines $C_j(\boldsymbol{q})$, `normalize_by` defines N_j, and `upper_bound` defines c_j. For the restorative beam, the two

constraints are

$$\frac{1}{|\Omega_0|}\int_{\Omega_0}\bar{\rho}\ \mathrm{d}V \le 0.50, \qquad \frac{1}{|\Omega_0|}\int_{\Omega_0}\bar{\phi}\ \mathrm{d}V \le 0.10, \tag{20}$$

where the physical fields are used because they enter the constitutive model. The corresponding input definition follows the same C_j, N_j, and c_j structure for any design-only constraint.

Problem-specific derived quantities may also be defined for visualization or postprocessing. These quantities do not enter the optimization unless they are explicitly used in the objective or constraints. For example, the hMSM beam writes the effective particle fraction $\bar{\rho}\bar{\phi}$ and the corresponding in-plane magnetization direction in addition to displacement and the physical design fields.

Finally, the input script combines the mesh, design variables, material model, loading, objective, constraints, and numerical settings into the problem definition and passes it to the common optimization driver. Listing 3 shows the principal entries while omitting problem-specific output and solver details.

```python
from matto.driver import OptimizationDriver
from matto.materials import HardMagneticSoftMaterial

material = HardMagneticSoftMaterial(**material_parameters)

problem = {
    "mesh": mesh,
    "design_variables": design_variables,
    "boundary_conditions": boundary_conditions,
    "traction_boundaries": traction_boundaries,
    "load_steps": load_steps,
    "load_cases": load_cases,
    "material": material,
    "build_objective": build_objective,
    "build_constraints": build_constraints,
    "fem_options": fem_options,
    "optimization_options": optimization_options,
    "output_options": output_options,
}

if __name__ == "__main__":
    OptimizationDriver(problem).run()
```

Listing 3: Problem definition and common entry point.

A new study is therefore defined primarily through a new input script and, when necessary, a new material model. The shared finite-element, sensitivity, and optimization modules do not need to be modified when only the geometry, design fields, loading, objective, constraints, or constitutive free energy changes.

3.2 Defining a material through its free energy. The material interface isolates the constitutive model from the definition of a particular optimization problem. The problem input specifies the geometry, loading, design fields, objective, and constraints, whereas the material model defines how the physical design fields and prescribed stimuli enter the free-energy density. Several optimization problems can therefore share the same constitutive model without modifying the finite-element or optimization procedures.

A subclass of `Material` specifies the design fields entering the free energy, the prescribed stimuli, the material parameters, and a scalar free-energy density through `energy(F, fields, stimuli)`. `StateProblem` supplies the deformation gradient, physical design fields, and prescribed stimuli and differentiates the resulting UFL expression to obtain the stress. Listing 4 shows the hMSM implementation and its principal constitutive operations. Imports and parameter validation are omitted; the complete class is available in `src/matto/materials/hmsm.py`.

```python
class HardMagneticSoftMaterial(Material):
    fields = ("rho", "phi", "theta")
    parameters = {
        "G0": None, "mu0": None, "B_rem_mag": None,
        "magnetization_map": "F",
        "p_rho": 3.0, "eps_rho": 1.0e-6, "dim": 2,
    }

    def __init__(self, **parameters):
        super().__init__(**parameters)
        self.stimuli = {"B_app": (self.dim,)}

    def energy(self, F, fields, stimuli):
        rho = fields["rho"]
        phi = fields["phi"]
        theta = fields["theta"]
        B_app = stimuli["B_app"]
        C = F.T * F
        I1, J = ufl.tr(C), ufl.det(F)

        G_matrix = self.G0 * simp(
            rho, self.p_rho, self.eps_rho)
        reinforcement = ufl.exp(
            2.5 * phi / (1.0 - 1.35 * phi))
        mu = G_matrix * reinforcement
        K = 500.0 * G_matrix
        W_elastic = (
            (mu / 2.0) * (I1 - 3.0 - 2.0 * ufl.ln(J))
            + (K / 2.0) * (J - 1.0)**2
        )
        B_rem = self.B_rem_mag * director(
            theta, dim=self.dim)
        if self.magnetization_map == "R":
            B_mapped = polar_rotation(F) * B_rem
        else:
            B_mapped = F * B_rem
        W_magnetic = -(rho * phi / self.mu0) * (
            ufl.inner(B_mapped, B_app))
        return W_elastic + W_magnetic
```

Listing 4: hMSM free energy behind the declared material interface.

Before the variational forms are constructed, the material declaration is checked against the fields, stimuli, and parameter values supplied by the problem definition. Internal history variables are not supported in the present formulation because their evolution is not represented. The included material classes use the declared interface above; the earlier `build_free_energy` callback remains available for compatibility.

The listing also illustrates why constitutive interpolation is defined within the material energy rather than imposed by the optimization framework. The elastic contribution uses the penalized density factor in Eq. (9), including ε_ρ, whereas the magnetic coupling uses $\rho\phi$ without that stiffness floor. Other material models may use different fields, stimuli, invariants, and interpolation rules while supplying a scalar free-energy density through the same interface. The shared finite-element, sensitivity, and optimization procedures therefore depend on the material interface rather than on a particular constitutive equation or field name.

3.3 The `driver` module: Coordinating the optimization procedure. The `OptimizationDriver` coordinates the sequence defined in Section 2.6. From the problem definition, it constructs the design-field representations, state problem, objective and constraints, and sensitivity calculation. The free raw coefficients of the active fields form the MMA design vector. Prescribed fields supply physical values to the material model.

The same objects can also be used for forward analysis and sensitivity evaluation without taking an MMA step. This capability is used for derivative verification and for evaluating saved designs with the same state and sensitivity calculations used during optimization.

The repeated part of the algorithm follows the five steps introduced in Section 2.6. Listing 5 shows the corresponding operations in `OptimizationDriver`. Reporting, allocation, validation, and MPI ownership operations are omitted so that the listing emphasizes the coordination of design-field parameterization, state solution, sensitivity evaluation, and design update.

```python
# Operations in _iterate():
operators_updated = self.forward(iteration)
evaluation = self.evaluate(write_output=write_output)
self._mma_update(
    evaluation["objective_gradients"],
    evaluation["constraints"],
    evaluation["constraint_gradients"],
```

```
)

# Within evaluate(), one loop over load cases:
for load_case_index, load_case in enumerate(self.load_cases):
    self.solve_load_case(load_case)
    values, physical_gradients = self.sensitivity.evaluate()
    weight = float(load_case.get("weight", 1.0))
    objective_value_total += weight * values["objective"]
    use_constraints = load_case_index == 0

    for name, variable in active.items():
        vectors = [physical_gradients["objective"][name]]
        if use_constraints:
            vectors.extend(
                physical_gradients["constraints"][c][name]
                for c in constraint_names
            )
        raw = variable.backward(vectors)
        objective_gradients_total[name] += (
            weight * variable.free_part(
                _owned_gradient(raw[0], variable.raw)))
        # Copy first-case constraint gradients from raw[1:].

# _mma_update packs gradients and bounds, calls MMA, and
# unpacks the proposed vector into the free raw entries:
for name, variable in active.items():
    variable.set_values(
        updated_design_vector[self.design_slices[name]])
```

Listing 5: Principal operations in `OptimizationDriver`; allocation and packing are omitted.

The call to `forward` applies the design-field parameterization before the state solutions are evaluated. Each load case is then solved to its prescribed target, followed by objective and sensitivity evaluation. Load-case objective values and gradients are weighted according to Eq. (10). Because the constraints considered here depend only on the physical design fields, their values and gradients are evaluated once for each design.

After sensitivity evaluation, `variable.backward` maps the physical-field sensitivities to the corresponding raw design variables. The resulting gradients are assembled in the same field order as the design vector and supplied to MMA. The updated design vector then defines the raw fields for the next iteration. The load-incremented nonlinear state solution used for each load case is described in Section 3.5.

3.4 The `design` module: Design fields and operators. The `design` module implements the raw-to-physical parameterization introduced in Section 2.3. It defines a common `Operator` interface, the `HelmholtzFilter`, `HeavisideProjection`, and `Identity` operators, and the `DesignVariable` class that composes these operators for each design field.

Each operator defines a `forward` map and a `backward` action on sensitivity vectors. The latter applies the transpose of the linearized field map required by Eq. (19). An operator may also carry a continuation schedule through `update(iteration)` and `continuation_complete`. This common interface allows different parameterization sequences to be assigned to different design fields without introducing field-specific branches in the optimization procedure.

A `HelmholtzKernel` assembles the filter problem in Eq. (5) once during initialization. In coefficient form,

$$K_i^{\mathrm{H}} \widetilde{z}_i = M_i^{\mathrm{H}} z_i, \qquad (D\mathcal{H}_i)^T = (M_i^{\mathrm{H}})^T (K_i^{\mathrm{H}})^{-T}. \tag{21}$$

The Helmholtz matrix is symmetric. Design fields that use the same mesh, finite-element spaces, filter radius, and solver options share the assembled matrices and solver through a common kernel, while each `HelmholtzFilter` retains its own input and output fields. Listing 6 shows the forward filter solve and the corresponding transpose action on sensitivity vectors.

```
def forward(self):
    kernel = self.kernel
    kernel.T_mat.mult(self.input.x.petsc_vec, kernel.rhs)
    kernel.solver.solve(kernel.rhs, self.output_wrap)
    self.output.x.scatter_forward()

def backward(self, gradients):
    kernel = self.kernel
    # One input-side work vector is allocated per slot.
    values = []
    for slot, gradient in enumerate(gradients):
        if gradient is None:
            values.append(None)
            continue

        kernel.solver.solve(gradient, kernel.af_wrap)
        kernel.af.x.scatter_forward()
        result = self._results[slot]
        kernel.T_mat.multTranspose(
            kernel.af.x.petsc_vec, result)
        values.append(result)

    return values
```

Listing 6: Forward and transpose actions of the Helmholtz filter.

The `HeavisideProjection` implements the pointwise map in Eq. (7). During the forward operation it evaluates the projected field and stores the pointwise derivative `dphys`. The corresponding sensitivity mapping multiplies the incoming vector by this derivative, which applies the local chain rule. Listing 7 shows these two operations; construction of the operator and the β-continuation schedule are handled separately.

```
def forward(self):
    beta, eta = self.beta, self.eta
    x = self.input.x.petsc_vec.array
    denominator = (
        np.tanh(beta * eta) + np.tanh(beta * (1 - eta)))

    self.dphys = (
        beta * (1 - np.tanh(beta * (x - eta))**2)
        / denominator)

    self.output.x.petsc_vec.array[:] = (
        np.tanh(beta * eta) + np.tanh(beta * (x - eta))
    ) / denominator

    self.output.x.scatter_forward()

def backward(self, gradients):
    for gradient in gradients:
        if gradient is not None:
            gradient.array *= self.dphys

    return gradients
```

Listing 7: Heaviside projection and sensitivity mapping.

For each declared field, `DesignVariable` creates the raw and physical FEniCSx functions, stores the activity status, initialization, bounds, and prescribed value, constructs the requested operator chain, and identifies fixed regions. Consecutive operators must use compatible finite-element spaces, and an identity map requires the same space on both sides.

Listing 8 shows the forward and backward operations. The forward operation applies the operators, advances continuation when required, and imposes fixed physical values. The backward operation zeros the sensitivities of fixed physical entries and applies the transpose operators in reverse order, consistent with Eq. (19).

```
def forward(self, iteration):
    if not self.active:
        self._assign(self.phys, self.prescribed_value)
        return False

    updated = False
    for operator in self.operators:
        updated = operator.update(iteration) or updated
        operator.forward()

    self._hold(self.phys, self.held_phys)
    return updated

def backward(self, physical_gradients):
```

```
    # Active-field path; validation is omitted here.
    gradients = [g.copy() if g is not None else None
                 for g in physical_gradients]

    for gradient in gradients:
        if gradient is not None:
            for mask, _ in self.held_phys:
                gradient.array[mask] = 0.0

    for operator in reversed(self.operators):
        gradients = operator.backward(gradients)

    return [g.array.copy() if g is not None else None
            for g in gradients]
```

Listing 8: Forward design-field parameterization and reverse sensitivity mapping.

The MMA vector contains the free raw entries of each active field. The methods `get_values`, `get_lower_bounds`, `get_upper_bounds`, and `set_values` act on those entries.

3.5 The `state` module: Constructing the nonlinear state problem. The `StateProblem` class constructs the finite-element problem associated with the continuum formulation in Section 2.2. From the problem definition, it creates the first-order vector displacement space, displacement and adjoint fields, test function, Dirichlet conditions, traction measures, and mutable `Constant` objects for the prescribed mechanical loads and stimuli. This construction is performed once. For each load case and load increment, only the values of the corresponding constants are updated. The displacement approximation is first-order in the present implementation.

Listing 9 shows the principal operations used to construct the nonlinear equilibrium problem. `StateProblem` forms the deformation gradient, supplies the physical design fields and prescribed stimuli to the selected material model, evaluates the free-energy density, and differentiates it with UFL to obtain the first Piola–Kirchhoff stress. The internal and external virtual-work forms then define the nonlinear residual. The implementation uses `R = L - a`, which differs by a sign from Eq. (2) but has the same equilibrium solution.

```
F = ufl.variable(ufl.Identity(dim) + ufl.grad(u_field))
fields = {name: design_variables[name].phys
          for name in material.fields}
stimuli_used = {name: stimuli[name]
                for name in material.stimuli}

W = material.energy(F, fields, stimuli_used)
P = ufl.diff(W, F)

a = inner(grad(v), P) * dx
L = inner(v, body_force) * dx
for name, traction in traction_constants.items():
    L += inner(v, traction) * traction_measures[name]

R = L - a
nonlinear_problem = WrapNonlinearProblem(
    u_field, R, bcs, solver_options["state"])

f_int = ufl.derivative(W * dx, u_field, v)
```

Listing 9: Energy differentiation and nonlinear residual in `state.py`; local aliases are used.

The state problem also constructs the external-work form used by compliance-type objectives. The problem definition then supplies the objective and constraints through the corresponding functions, as shown in Listing 10. These calls create symbolic UFL forms whose values and derivatives are evaluated after each load case reaches equilibrium.

```
# StateProblem._build_external_work:
external_work = inner(u_field, body_force) * dx
for name, traction in traction_constants.items():
    external_work += (
        inner(u_field, traction) * traction_measures[name])

# OptimizationDriver._build_objective_and_constraints:
objective_form = problem["build_objective"](
    state.u_field, state.external_work_form, state.dx)

constraints = problem["build_constraints"](
    design_variables, state.dx)
```

Listing 10: Construction of the external-work, objective, and constraint forms.

The displacement field and all raw and physical design fields are available for output. Additional problem-specific expressions may be supplied through `build_output_fields`, as described in Section 3.1. The state object retains the fields, prescribed-data constants, residual and internal-force forms, integration measures, and solver information required by the sensitivity calculation and subsequent load-case solutions.

The nonlinear solution for each load case is carried out by `StateProblem.solve`. The displacement, mechanical loads, and stimuli are first reset, after which the prescribed target values are increased together from zero over the specified number of load increments. Newton's method is applied after each increment. Because the existing `Constant` objects are updated in place, the variational forms do not need to be reconstructed during load incrementation. Listing 11 shows this procedure.

```
zero_function(u_field)
set_constant(body_force, body_force_target, scale=0.0)

for traction in traction_constants.values():
    set_constant(traction, np.zeros_like(traction.value))

for stimulus in stimuli.values():
    set_constant(stimulus, np.zeros_like(stimulus.value))

for step in range(1, load_steps + 1):
    load_fraction = step / load_steps

    set_constant(
        body_force, body_force_target,
        scale=load_fraction)

    for name, traction in traction_constants.items():
        target = traction_targets.get(
            name, np.zeros_like(traction.value))
        set_constant(
            traction, target, scale=load_fraction)

    for name, stimulus in stimuli.items():
        target = stimulus_targets.get(
            name, np.zeros_like(stimulus.value))
        set_constant(
            stimulus, target, scale=load_fraction)

    self.nonlinear_problem.solve_fem()
```

Listing 11: Load and stimulus continuation for one load case.

3.6 The `sensitivity` module: Assembling design derivatives. The `Sensitivity` class implements the derivative calculation developed in Section 2.5. It receives the state problem, objective form, constraint definitions, and active design variables and constructs the required UFL derivative forms. The calculation is independent of specific field names such as `rho`, `phi`, or `theta`; derivatives are generated for whichever physical design fields are active in the problem.

For each active field q_i, the implementation constructs the direct objective derivative $\partial f/\partial q_i$ when that dependence is present, together with the derivative required for the adjoint contribution in Eq. (18). Direct derivatives are also constructed for every supported constraint and active physical field. If a functional is independent of a field, the corresponding derivative contribution is treated as zero rather than requiring a field-specific branch.

At a converged state, `evaluate` first assembles the objective and all direct physical-field derivatives. If the objective depends on displacement, `_solve_adjoint` then assembles the right-hand side $-\partial f/\partial \boldsymbol{u}$ and the state tangent and solves the transposed system in Eq. (17). Homogeneous values are imposed on the displacement

Dirichlet degrees of freedom. Listing 12 shows the principal operations.

```
self._assemble_into_vector(
    self.dobjective_du_vector,
    self.dobjective_du_form,
)

set_bc(
    self.dobjective_du_vector,
    self.bcs,
    alpha=0.0,
)

self.dobjective_du_vector.scale(-1.0)

self.dfdu_matrix.zeroEntries()
assemble_matrix(
    self.dfdu_matrix,
    self.dfdu_form,
    bcs=self.bcs,
)
self.dfdu_matrix.assemble()

self.adjoint_solver.solveTranspose(
    self.dobjective_du_vector,
    self.lambda_field.x.petsc_vec,
)
```

Listing 12: Assembly and solution of the adjoint problem.

The same adjoint field is then used for every active physical design field. Rather than assembling a rectangular state-to-design Jacobian, the implementation forms the scalar adjoint internal-work functional $\boldsymbol{\lambda}^T \boldsymbol{f}_{\text{int}}$ and differentiates it directly with respect to q_i. This produces the term $(\partial \boldsymbol{f}_{\text{int}}/\partial q_i)^T \boldsymbol{\lambda}$ in Eq. (18). Listing 13 shows this operation. The indirect contribution is added to the previously assembled direct derivative to obtain the total physical-field sensitivity.

```
# Construction of the derivative forms:
adjoint_work = ufl.action(
    self.internal_force_ufl, self.lambda_field)

for name, variable in self.active_design_variables.items():
    design_test = ufl.TestFunction(
        variable.phys.function_space)

    # Construct this form only when the dependence exists.
    self.adjoint_contribution_forms[name] = form(
        ufl.derivative(
            adjoint_work, variable.phys, design_test))

# Evaluation after the adjoint solve:
for name in self.active_design_variables:
    contribution = self.adjoint_contribution_forms[name]
    if contribution is None:
        continue

    vector = self.adjoint_contribution_vectors[name]
    self._assemble_into_vector(vector, contribution)
    objective_gradients[name].axpy(1.0, vector)
```

Listing 13: Direct assembly of the adjoint contribution for each physical field.

The state tangent is assembled for both the Newton and adjoint solves. What is avoided is the explicit construction of a state-to-design Jacobian for each active field. Consequently, a scalar state-dependent objective requires one adjoint solve per load case, irrespective of the number of active design fields. If the objective is independent of displacement, the adjoint solve is skipped.

The constraints considered here are state independent, so their derivatives are obtained directly with respect to the physical design fields. Each constraint is normalized according to Eq. (12), and the same normalization is applied to its derivative. A state-dependent constraint would require an additional adjoint solve and is not included in the present formulation.

The resulting objective and constraint sensitivities are returned field by field to `OptimizationDriver`. They are then mapped from the physical fields to the corresponding raw design variables through the operator chains described in Section 3.4.

3.7 The `mma` and `utility` modules. The `mma` module contains the numerical implementation of the method of moving asymptotes inherited from FEniTop [12]. The function `mma_optimizer` receives the current raw design variables, their bounds and update history, the objective gradient, the normalized constraint residuals, and the corresponding constraint gradients. It solves the MMA subproblem and returns the updated design together with the current asymptotes and design-change measure [17]. The module is independent of the material model, prescribed stimuli, UFL energy expressions, and load cases; the driver provides the finite-dimensional quantities required by MMA.

The `utility` module provides supporting numerical operations used by the state, sensitivity, and design calculations. Solver options for the state, adjoint, and filter systems are applied independently through PETSc. `WrapNonlinearProblem` constructs and reuses the nonlinear state solver, while `Communicator` gathers distributed field values when serial output is required. Additional utilities support matrix comparison and related numerical checks. Iteration histories, snapshots, and plotting are handled separately by `postprocess.py`.

Listing 14 shows the nonlinear state-problem wrapper. The DOLFINx nonlinear problem and Newton solver are constructed once, after which `solve_fem` is called at each load increment described in Section 3.5. Solver tolerances, convergence criterion, maximum Newton iterations, and PETSc options are supplied through the state-solver settings.

```
class WrapNonlinearProblem:
    def __init__(self, u, R, bcs=None, solver_options=None):
        if bcs is None:
            bcs = []
        if solver_options is None:
            solver_options = {}

        self.u = u
        self.bcs = bcs
        self.problem = DolfinxNonlinearProblem(R, u, bcs)
        self.solver = NewtonSolver(
            u.function_space.mesh.comm, self.problem
        )

        self.solver.atol = float(
            solver_options.get("atol",
      DEFAULT_STATE_SOLVER["atol"])
        )
        self.solver.rtol = float(
            solver_options.get("rtol",
      DEFAULT_STATE_SOLVER["rtol"])
        )
        self.solver.convergence_criterion = solver_options.get(
            "convergence_criterion", "incremental"
        )

        if "max_it" in solver_options:
            self.solver.max_it = int(solver_options["max_it"])

        apply_petsc_options(
            self.solver.krylov_solver,
            solver_options.get("petsc_options", {}),
            prefix=f"state_ksp_{id(self)}",
        )

    def solve_fem(self):
        num_iterations, converged = self.solver.solve(self.u)
        self.u.x.scatter_forward()

        assert converged, (
            f"Newton solver did not converge in {num_iterations}
      iterations"
        )
```

Listing 14: Reusable nonlinear state-problem wrapper.

3.8 Postprocessing and consistency of saved states. The `PostProcessor` interface records iteration histories, field snapshots, and plots without modifying the mechanics or optimization

equations. After the final MMA update, the driver remaps the raw design fields and re-solves every load case so that the stored final fields and reported scalar quantities correspond to the same design.

Iteration snapshots are written after the MMA update. In the current implementation, the raw design arrays in an intermediate snapshot therefore correspond to the updated design, whereas the physical fields, displacement, and objective correspond to the preceding state evaluation. The latter quantities remain mutually consistent and are used for the iteration figures. Final results reported in this paper are taken from the separate final re-solve.

Together, the modules implement a common path from the material and problem definitions to nonlinear equilibrium, adjoint sensitivities, and the MMA design update. The examples in Section 4 change the constitutive model, active design fields, prescribed stimuli, and objective while reusing this numerical workflow.

4 Implemented material families and representative capabilities

The examples in this section demonstrate the use of the common formulation across material families with different free energies, prescribed stimuli, active design fields, and optimization objectives. They are not intended as new constitutive studies; the cited source papers remain the basis for the material models. Any adaptation made in the MatTO implementation is stated explicitly.

An overbar denotes a physical design field after the parameterization described in Section 2.3. The field $\bar{\rho}$ denotes structural density, while the physical meanings of $\bar{\phi}$, $\bar{\theta}$, and the prescribed stimulus depend on the material family. Table 3 summarizes the eleven responsive-material examples.

All examples use first-order displacement interpolation and quadrature degree two. Active raw fields are cell-wise constants and, when filtered, are mapped to continuous first-order physical fields. Loads and prescribed stimuli are increased incrementally before each objective and adjoint evaluation. The example inputs give the meshes, filter radii, and load increments. The design-change tolerance is 10^{-3}. The two-dimensional examples use a move limit of 0.02 and an iteration limit of 400 (300 for the restorative beam). The three-dimensional morphing surface uses a move limit of 0.05 and an iteration limit of 1000. Director angles are bounded by $[-\pi, \pi]$ and remanent-magnetization angles by $[-2\pi, 2\pi]$.

Table 4 lists the first evaluated objective and the objective from the final re-solve for each example. These values quantify improvement within a given problem but are not comparable across examples because the objectives have different definitions, units, and normalizations.

4.1 Hard-magnetic soft materials: Coupled structure and actuation.

The hMSM implementation builds on the joint material–structural optimization formulation of Galloway and Jha [7] and the topology-optimization approach of Zhao and Zhang [2]. The physical field $\bar{\rho}$ denotes structural density, $\bar{\phi}$ the magnetic-particle volume fraction, and $\bar{\theta}$ the programmed remanent-magnetization direction. In three dimensions, the present orientation parameterization remains planar, with $\boldsymbol{B}^r = B^r(\cos\bar{\theta}, \sin\bar{\theta}, 0)^T$, while the displacement field is fully three-dimensional. The prescribed stimulus $\boldsymbol{B}_{\text{app}}$ is a spatially uniform applied magnetic flux density whose magnitude and direction may vary between load cases.

For this material family, $\bar{\phi}$ is a particle concentration rather than a material-selection variable and affects both the elastic reinforcement and magnetic coupling. The implemented energy has the form

$$\widehat{W}_{\text{hMSM}} = W_{\text{el}}(\boldsymbol{F}; \bar{\rho}, \bar{\phi}) + W_{\text{mag}}(\boldsymbol{F}; \bar{\rho}, \bar{\phi}, \bar{\theta}, \boldsymbol{B}_{\text{app}}).$$

The magnetic energy is

$$W_{\text{mag}} = -\frac{\bar{\rho}\bar{\phi}}{\mu_0}\left[\boldsymbol{M}(\boldsymbol{F})\boldsymbol{B}^r\right] \cdot \boldsymbol{B}_{\text{app}}.$$

Here $\boldsymbol{M}(\boldsymbol{F})$ maps the reference remanent flux density and μ_0 is the vacuum permeability. MatTO supports $\boldsymbol{M}(\boldsymbol{F}) = \boldsymbol{R}(\boldsymbol{F})$, where $\boldsymbol{R}$ is the rotation tensor in the right polar decomposition $\boldsymbol{F} = \boldsymbol{R}\boldsymbol{U}$. The examples here use $\boldsymbol{M}(\boldsymbol{F}) = \boldsymbol{F}$. The detailed constitutive expressions and their UFL implementation are given in Listing 4. The elastic contribution uses the density penalization $s_\rho(\bar{\rho})$, whereas the magnetic contribution uses $\bar{\rho}\bar{\phi}$ directly. Thus ϕ changes both elastic reinforcement and magnetic actuation, while ρ attenuates both contributions as material is removed.

Remark: Two-dimensional energy reference. For the two-dimensional hMSM examples, $I_1 = \text{tr}(\boldsymbol{F}^T\boldsymbol{F})$ is evaluated from the in-plane 2×2 deformation gradient. The resulting elastic energy differs from its plane-strain three-dimensional embedding by $\mu(\bar{\rho}, \bar{\phi})/2$. This difference is independent of displacement and therefore does not alter stress, equilibrium, or the displacement-based objectives considered here. It would, however, matter for an objective defined directly from the stored energy.

4.1.1 Rotational actuator. The rotational actuator in `examples/hMSM/input_wheel.py` demonstrates material programming within a prescribed solid geometry. The central hub is clamped and the structure is actuated by an upward applied field $\boldsymbol{B}_{\text{app}} = (0, 100)$ mT, as shown in Fig. 2(a). The fields ϕ and θ are optimized to increase counterclockwise tangential displacement in an annular region near the outer rim. Because MatTO minimizes the supplied functional, the objective is

$$f_{\text{wheel}} = -\int_{\Omega_0} w_r(\boldsymbol{X})\, \boldsymbol{u} \cdot \boldsymbol{e}_\theta \, \mathrm{d}V,$$

where w_r localizes the objective near the rim and $\boldsymbol{e}_\theta$ is the counterclockwise unit tangent.

The objective decreases from −244.99 to −436.05, while the final mean particle fraction is 0.25781 against the prescribed limit of 0.30. Figure 2(b) shows the optimized particle volume fraction and remanent-magnetization direction, and Fig. 2(c) shows the deformed configuration.

The resulting finite rotation is evaluated independently of the tangential-displacement objective. For reference and deformed rim positions $\boldsymbol{X}_i$ and $\boldsymbol{x}_i$ measured from the fixed hub, the signed rotation is

$$\vartheta_i = \text{atan2}\big(X_{i,1}x_{i,2} - X_{i,2}x_{i,1},\ \boldsymbol{X}_i \cdot \boldsymbol{x}_i\big).$$

At iteration 400, the mean rotation over rim nodes with $9.05 < r < 9.95$ and the best-fit rigid rotation both give 62.64°.

4.1.2 Translational actuator. The translational actuator in `examples/hMSM/input_scissor.py` uses a prescribed frame geometry and a localized displacement-tracking objective. The frame is clamped at the left and actuated by $\boldsymbol{B}_{\text{app}} = (125, 0)$ mT (Fig. 3(a)). A Gaussian weight w_s is centered near the output region, where the target displacement is $\boldsymbol{u}^\star = (3, 0)$. The objective is

$$f_{\text{scissor}} = \int_{\Omega_0} w_s(\boldsymbol{X}) \left[(u_x - 3)^2 + u_y^2\right] \mathrm{d}V.$$

The first term drives the output toward the prescribed horizontal translation, while the second suppresses vertical motion.

The particle fraction is held at zero in the output plate in both the raw and physical fields. The objective decreases from 1.4204 to 1.4383×10^{-4}, while the final mean particle fraction is 0.14386 against the prescribed limit of 0.15.

The displacement field gives the Gaussian-weighted output

$$\bar{\boldsymbol{u}}_{\text{out}} = \frac{\int_{\Omega_0} w_s \boldsymbol{u} \, \mathrm{d}V}{\int_{\Omega_0} w_s \, \mathrm{d}V} \approx (2.9982, -0.0005),$$

which is close to the target $(3, 0)$. Figure 3(b) shows the optimized particle volume fraction and remanent-magnetization direction, and Fig. 3(c) shows the deformed frame.

Table 3 Responsive-material examples and their active design fields. A field omitted from the active set is prescribed when required by the material model.

Family and input	Active fields	Optimization goal
hMSM, `input_beam.py`	ρ, ϕ, θ	Two-case mechanical compliance.
hMSM, `input_wheel.py`	ϕ, θ	Weighted tangential rim displacement.
hMSM, `input_scissor.py`	ϕ, θ	Localized displacement tracking.
hMSM, `input_morphing_dome_3d.py`	ρ, ϕ, θ	Loaded 3D surface-shape tracking.
Anisotropic MAE, `input_beam.py`	ϕ, θ	Field-on compliance in a fixed geometry.
Anisotropic MAE, `input_bridge.py`	ρ, ϕ, θ	Field-on compliance with topology design.
Isotropic MAE, `input_beam.py`	ϕ	Field-on compliance in a fixed geometry.
Isotropic MAE, `input_bridge.py`	ρ, ϕ	Field-on compliance with topology design.
LCE, `input_morphing_strip.py`	ϕ, θ	Distributed target-shape tracking.
LCE, `input_pusher.py`	ρ, ϕ, θ	Upward displacement against a downward load.
LCE, `input_vertical_extension.py`	ϕ, θ	Normalized top-edge extension.

Table 4 Numerical results for the eleven examples. f_0 and f_f are the objectives at the initial design and after solving equilibrium at the final design. Δ is the final normalized design change over the free raw coefficients. Wall times are rounded to the nearest minute. The dome cell count refers to the computed quadrant.

Problem	Cells	Iter./cap	f_0	f_f	Stop	Δ	Time (min)
hMSM restorative beam	4500	300/300	715.156	45.2989	Limit	0.0199	29
hMSM rotational actuator	5716	400/400	−244.991	−436.046	Limit	0.009195	42
hMSM translational actuator	4272	400/400	1.42041	1.438×10^{-4}	Limit	0.002224	16
hMSM 3D morphing surface	3456	752/1000	0.0904833	1.776×10^{-5}	Tolerance	9.924×10^{-4}	65
LCE morphing strip	1440	58/400	0.00523015	8.823×10^{-5}	Tolerance	9.254×10^{-4}	2
LCE pusher	3456	400/400	0.00448643	−0.0214981	Limit	0.004965	12
LCE vertical extension	2000	116/400	0.00285063	−0.0267257	Tolerance	7.871×10^{-4}	3
Anisotropic MAE beam	4500	370/400	30.081	18.5296	Tolerance	9.676×10^{-4}	45
Anisotropic MAE bridge	3840	400/400	2.02096	0.270734	Limit	0.001269	7
Isotropic MAE beam	4500	233/400	15.7966	11.1553	Tolerance	9.965×10^{-4}	8
Isotropic MAE bridge	3840	400/400	1.01134	0.143215	Limit	9.161×10^{-4}	6

4.1.3 Restorative beam with joint material–structural optimization. The restorative cantilever in `examples/hMSM/input_beam.py` activates all three design fields and introduces multiple load cases. One case combines a downward traction with an upward magnetic field, while the second reverses both directions. The two cases have equal weights and use the compliance objective

$$f_{\text{beam}} = \sum_{c=1}^{2} \int_{\Gamma_t} \bar{\boldsymbol{t}}_c \cdot \boldsymbol{u}_c \, \mathrm{d}S.$$

Domain-average constraints limit the structural density and particle concentration. This example therefore combines topology, particle concentration, remanent-magnetization direction, and multiple load cases within one optimization problem.

The beam occupies 100×20 mm and is clamped at the left edge. The right-edge tractions are $(0, -0.5)$ and $(0, 0.5)$ kPa, paired with applied magnetic fields $(0, 25)$ and $(0, -25)$ mT, as shown in Fig. 4(a, b). The mean density and particle-fraction limits are 0.50 and 0.10, respectively.

Figure 4(c, d) shows the resulting density, particle distribution, and remanent-magnetization direction. The optimized density is concentrated in the load-carrying regions, while the magnetic material is distributed primarily near the upper and lower portions of the beam. The same design is evaluated under both opposing load and field cases before each update. The objective of this design is 45.299, compared with 715.16 initially, and both volume constraints are satisfied. The beam reached its 300-iteration limit before meeting the design-change tolerance.

4.1.4 Load-bearing morphing surface in three dimensions. The three-dimensional example applies the same material model and three-field optimization framework to a load-bearing morphing surface. A $24 \times 24 \times 3$ mm slab is clamped on its four side faces. The top 0.5 mm layer is held solid and nonmagnetic by prescribing density one and particle fraction zero in the raw and physical fields. The remanent direction in the skin is held at its initial value. All three fields are optimized below the skin. Mirror symmetry about $X = 0$ and $Y = 0$ reduces the computation to one quadrant with $24 \times 24 \times 6$ hexahedral cells. The displacement component normal to each symmetry plane is fixed to zero there. The full design and displacement fields are obtained by reflection. Figure 5(a) shows the setup.

The applied magnetic flux density is 200 mT along z, and a downward traction of 0.50 kPa acts on the top surface. The target displacement is

$$w^\star(X, Y) = h_\star \left[1 - (X/12)^2\right]^2 \left[1 - (Y/12)^2\right]^2, \qquad h_\star = 1 \text{ mm}.$$

The objective is the normalized mean-square tracking error on the quadrant,

$$f_{\text{dome}} = \frac{1}{A_q h_\star^2} \int_{\Gamma_{\text{top},q}} (u_z - w^\star)^2 \, \mathrm{d}S, \qquad A_q = 12^2 \text{ mm}^2.$$

The reference remanent flux density has magnitude 200 mT and lies in the x–y plane. With the applied field along z, the magnetic coupling contains $\bar{\rho}\bar{\phi}(\cos\bar{\theta}\,\partial_X u_z + \sin\bar{\theta}\,\partial_Y u_z)$, so the orientation field selects the local slope direction driven by the field. The initial orientation follows the gradient of the target surface and is retained as a cell-wise unfiltered field to avoid averaging across the angle discontinuity. The mean constraints are $\langle\bar{\rho}\rangle \le 0.60$ and $\langle\bar{\phi}\rangle \le 0.10$, including the skin. The dome calculation uses a stiffness floor of $\varepsilon_\rho = 10^{-2}$ in the density interpolation.

Figure 5(b, c) shows the final design, and Fig. 5(d–f) compares the target and computed top-surface displacements. The objective decreases from 0.090483 to 1.7762×10^{-5}. The computed central

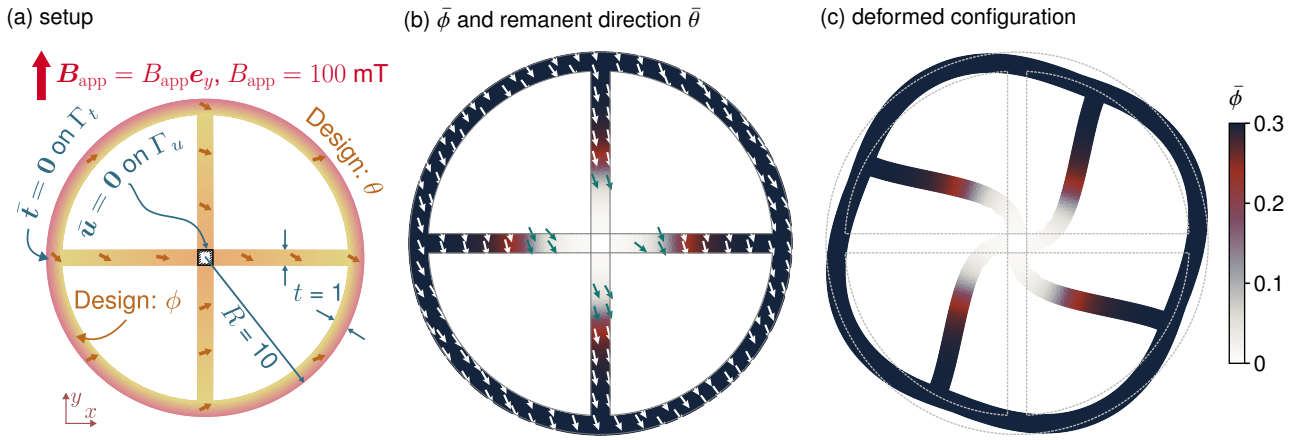


Fig. 2 Rotational hMSM actuator. (a) Wheel geometry with the clamped hub, the applied magnetic field, and the design fields. (b) Final particle volume fraction $\bar{\phi}$; arrows show the remanent-magnetization direction $\bar{\theta}$. (c) Deformed configuration colored by $\bar{\phi}$ over the dashed reference outline. The mean rim rotation is 62.64°.

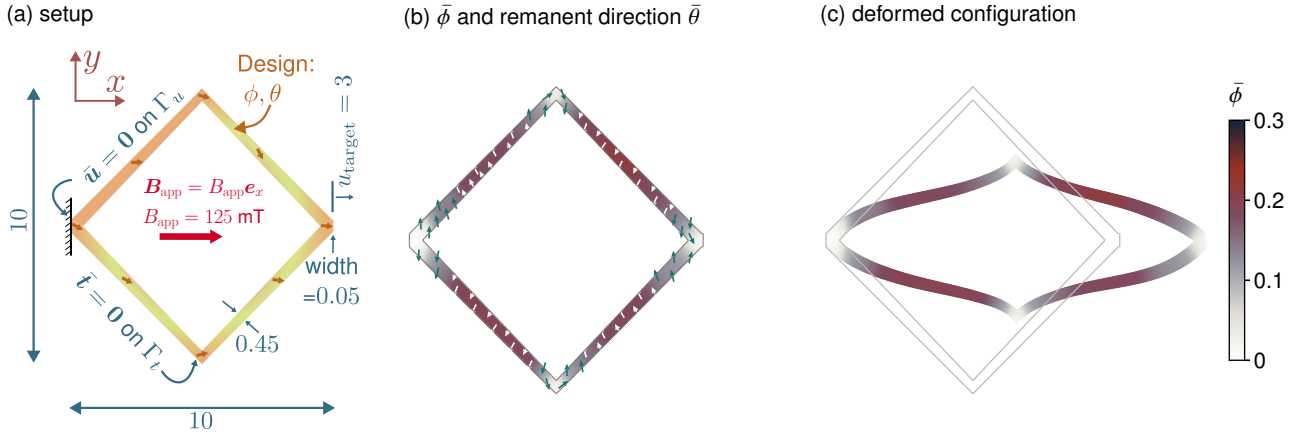


Fig. 3 Translational hMSM actuator. (a) Frame geometry with the left clamp, the applied magnetic field, and the target translation of the output. (b) Final particle volume fraction $\bar{\phi}$; arrows show the remanent-magnetization direction $\bar{\theta}$. (c) Deformed configuration colored by $\bar{\phi}$ over the reference outline.

displacement is 0.987 mm for a 1.00 mm target, and the root-mean-square error over the top surface is 0.0042 mm. Both volume constraints are satisfied. The design-change tolerance of 10^{-3} is met at iteration 752.

4.2 Magneto-active elastomers: Anisotropic and isotropic stiffening. The MAE implementations are based on the anisotropic magnetorheological-elastomer model of Akbari and Khajehsaeid [13] and the isotropic magneto-active-polymer model of Garai and Haldar [14]. The active endpoints use parameters reported for homogeneous composites containing 20% particles. To use these responses in material and topology optimization, MatTO interpolates between a particle-free silicone endpoint and the corresponding magneto-active endpoint.

For both material families, $\bar{\rho}$ denotes structural density and $\bar{\phi} \in [0, 1]$ is a material-selection field. The value $\bar{\phi} = 0$ denotes particle-free silicone and $\bar{\phi} = 1$ denotes the corresponding 20% magneto-active composite. Intermediate values interpolate between the two endpoint energies and should not be interpreted as particle volume fractions. The common interpolation is

$$\widehat{W}_{\text{MAE}} = s_\rho(\bar{\rho}) \left[s_\phi(\bar{\phi}) W_{\text{active}} + \left(1 - s_\phi(\bar{\phi})\right) W_{\text{passive}} \right], \qquad s_\phi(\bar{\phi}) = \bar{\phi}^{p_\phi}. \tag{22}$$

Thus s_ρ controls the presence of structural material, while s_ϕ interpolates between the passive and magneto-active responses. The density penalization s_ρ is defined in Eq. (9). Because both interpolations are defined within the free energy, their derivatives enter the same sensitivity calculation described in Section 2.5.

In both implementations, the prescribed scalar stimulus $h = \mu_0 \|\boldsymbol{H}\|$ is specified in tesla and represents the magnitude of a spatially uniform applied magnetic field. The anisotropic model additionally uses $\bar{\theta}$ to describe the in-plane particle-chain direction, with prescribed field direction $\theta_M = 0$. The isotropic model has no orientation field.

The two families differ in their active and passive endpoint energies. For the anisotropic MAE,

$$A^\star = s\left(1 - e^{-(h/2h_s)^2}\right) q(\bar{\theta}, \theta_M),$$

$$W^{\text{Ak}}_{\text{active}} = \frac{K_{\text{Ak}}}{2}(J-1)^2 + r(\bar{I}_4 - 1)^2 + (A_{\text{Ak}} + A^\star) g(\bar{I}_1; a_{\text{Ak}}, b_{\text{Ak}}),$$

$$W^{\text{Ak}}_{\text{passive}} = \frac{K_{\text{sil}}}{2}(J-1)^2 + A_{\text{sil}} g(\bar{I}_1; a_{\text{sil}}, b_{\text{sil}}).$$

For the isotropic MAE,

$$W^{\text{Ga}}_{\text{active}} = \frac{K}{2}(J-1)^2 + C^{\text{Ga}}_1(\bar{I}_1 - 3) + \left[C^{\text{Ga}}_2 + \widehat{C}_2(h)\right](\bar{I}_2 - 3),$$

$$W^{\text{Ga}}_{\text{passive}} = \frac{K}{2}(J-1)^2 + C^{\text{sil}}_1(\bar{I}_1 - 3) + C^{\text{sil}}_2(\bar{I}_2 - 3).$$

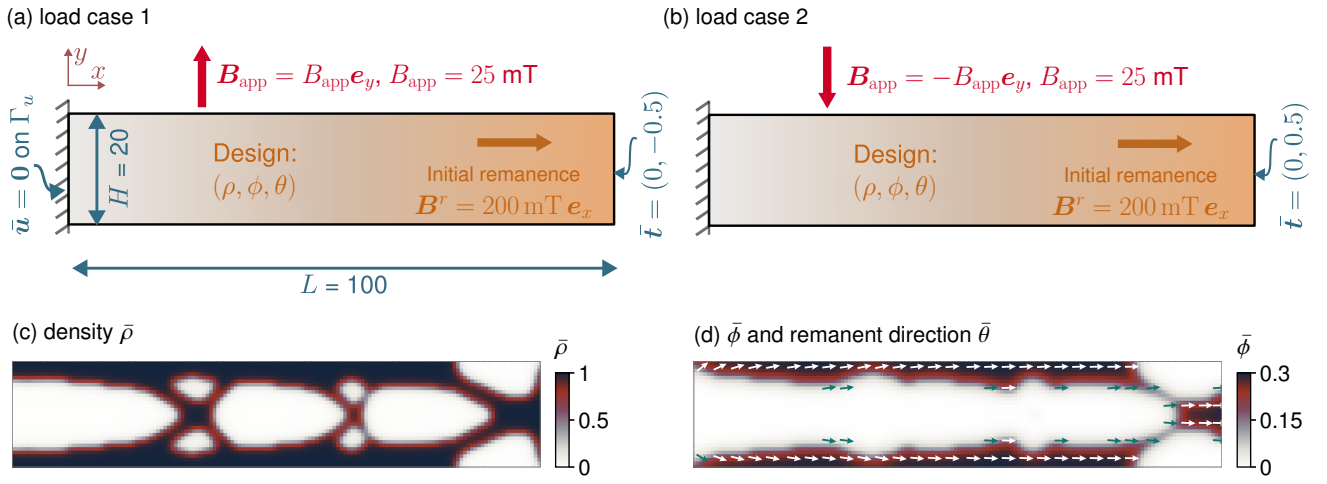


Fig. 4 Restorative hMSM beam. (a, b) Load cases 1 and 2: opposing right-edge tractions and applied magnetic fields on the beam clamped at the left edge. (c) Final physical density $\bar{\rho}$. (d) Final particle volume fraction $\bar{\phi}$; arrows show the remanent-magnetization direction $\bar{\theta}$.

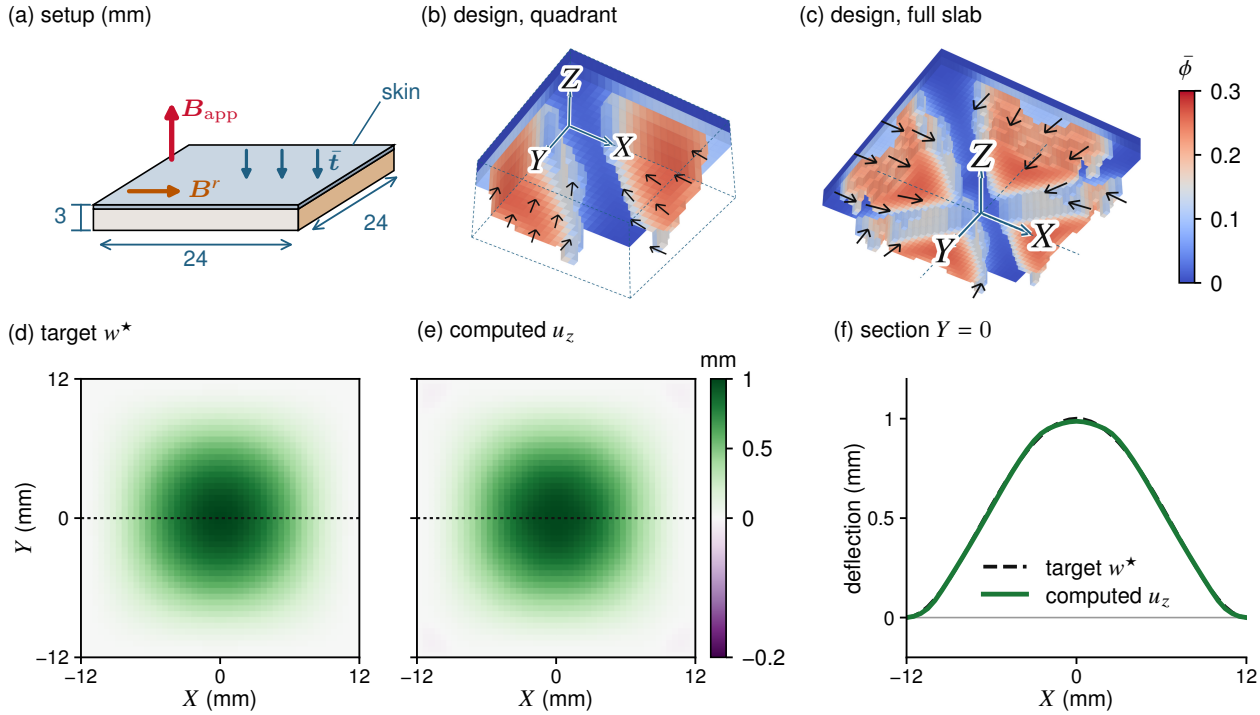


Fig. 5 Three-dimensional hMSM morphing surface. (a) Problem setup, with dimensions in millimeters. (b, c) Final design of the computed quadrant and the reflected full slab, viewed from below and including the skin. The dashed box outlines the quadrant, and the dashed lines in (c) mark the symmetry planes. Cells are displayed where $\bar{\rho} > 0.5$. Color denotes particle volume fraction, and arrows show remanent directions on bottom faces where $\bar{\phi} \geq 0.1$. (d, e) Target and computed top-surface displacements on a common scale in millimeters. (f) Profiles along $Y = 0$, indicated by dotted lines in (d, e).

The in-plane deformation gradient is embedded with $F_{33} = 1$ before constructing $\bar{C} = J^{-2/3} F^T F$, $\bar{I}_1 = \mathrm{tr}\,\bar{C}$, $\bar{I}_2 = [(\mathrm{tr}\,\bar{C})^2 - \mathrm{tr}(\bar{C}^2)]/2$, and $\bar{I}_4 = \widehat{\mathbf{N}} \cdot \bar{C}\widehat{\mathbf{N}}$, where $\widehat{\mathbf{N}} = (\cos\bar{\theta}, \sin\bar{\theta}, 0)^T$. The implemented anisotropic functions are

$$g(I; a, b) = \frac{e^{a(I-3)} - 1}{a} + b(I-2)[1 - \ln(I-2)] - b,$$

$$q(\bar{\theta}, \theta_M) = \frac{\sqrt{\cos^2(\bar{\theta} - \theta_M) + \delta_\theta^2} - \delta_\theta}{\sqrt{1 + \delta_\theta^2} - \delta_\theta}, \tag{23}$$

while the field-dependent coefficient in the isotropic model is

$$\widehat{C}_2(h) = b_1 \ln(1 + b_2 H) \arctan\left[a_1 (e^{a_2 H} - 1) H\right], \quad H = h/\mu_0.$$

The isotropic implementation therefore evaluates H in A/m, with $a_2 = 9.147\mu_0$ and $b_2 = 13.024\mu_0$, consistent with the source calibration [14].

Remark: Adaptation of the anisotropic MAE model. The anisotropic endpoint is an adaptation of Akbari and Khajehsaeid [13], not the complete calibrated magnetization model. The implementation omits the additive kI_7 magnetization-coupling term

and replaces the source model's signed chain-field alignment with the smoothed director-symmetric factor q in Eq. (23). Reversing the chain direction therefore leaves the implemented response unchanged. Both MAE implementations use isochoric invariants together with a volumetric penalty in the displacement-based finite-element formulation rather than exact incompressibility.

Table 5 lists the endpoint parameters used in the examples. The active-endpoint coefficients are taken from the cited composite calibrations, while the silicone coefficients are prescribed model parameters. The interpolation in Eq. (22) is a design interpolation between constitutive endpoints rather than a calibrated continuous law in particle volume fraction.

4.2.1 Material optimization in a prescribed beam geometry. The beam examples provide fixed-geometry demonstrations of the two MAE implementations. Structural density is prescribed as $\rho = 1$, while the material-selection field ϕ is optimized to reduce compliance under a downward traction and applied magnetic field. The anisotropic example additionally optimizes the particle-chain direction θ. It also evaluates a field-off load case with zero objective weight so that the corresponding response can be recorded without affecting the optimized objective. Both models use the 100×20 cantilever in Fig. 6(a), clamped at the left edge, with a downward right-edge traction of 0.50 kPa and $h = 0.45$ T.

The anisotropic beam objective decreases from 30.081 to 18.530 in 370 iterations, while the isotropic beam objective decreases from 15.797 to 11.155 in 233 iterations. Both runs meet the design-change tolerance, and in both the mean phase fraction reaches its bound of 0.30. Each reduction is measured relative to the initial design for that material model. Figure 6(b, c) shows the final phase fields. In both designs, the magneto-active phase forms bands in the upper and lower parts of the beam that extend from the clamp toward the loaded end. In the anisotropic design, the chain directions run nearly along the beam axis.

4.2.2 Joint topology and material optimization of a bridge. The bridge examples activate structural density in addition to the material-selection field. Both use the same bridge geometry and mechanical loading and minimize the compliance

$$f_{\text{bridge}} = \int_{\Gamma_t} \bar{\boldsymbol{t}} \cdot \boldsymbol{u} \, \mathrm{d}S, \qquad h = 0.45 \text{ T}.$$

The 120×50 domain in Fig. 7(a) is clamped along two lower support segments of width 12, and a downward traction of 1.0 kPa acts on a centered segment of width 10 on the upper edge. The two 12×4 support pads and the 10×4 load pad are held at $\rho = 1$ and $\phi = 0$ in the raw and physical fields. Their contributions remain included in the density and phase constraints. Both optimize ρ and ϕ, constrain the mean structural density to 0.45, and limit the magneto-active phase within the retained material. The anisotropic model additionally optimizes θ, whereas the isotropic model requires no orientation field. These two problems therefore use different design vectors while retaining the same state, sensitivity, and MMA procedures.

The final compliance is 0.27073 for the anisotropic MAE and 0.14322 for the isotropic MAE, compared with initial values of 2.0210 and 1.0113, respectively. In both cases, the optimized density forms two inclined load paths between the lower supports and the loaded region, as shown in Fig. 7(b, c). Figure 7(d, e) shows the phase fields, and the anisotropic design in Fig. 7(d) also develops a spatially varying chain direction within these material regions.

Both designs satisfy the density and retained magnetic-phase constraints.

4.3 Liquid crystal elastomers: Activation and shape programming. The LCE implementation follows the prescribed-order-parameter model used by Barrera et al. [6]. The physical field $\bar{\rho}$ denotes structural density, $\bar{\phi} \in [0, 1]$ interpolates between passive and programmed LCE, and $\bar{\theta}$ denotes the in-plane mesogen-director angle. Thus, $\bar{\phi} = 0$ corresponds to passive material and $\bar{\phi} = 1$ to programmed material whose activated response follows $\bar{\theta}$.

The prescribed scalar stimulus $a \in [0, 1]$, represented by `activation` in the input files, controls the reduction in nematic order. The value $a = 0$ gives the reference order parameter $S = S_0$, whereas $a = 1$ gives the fully activated value $S = 0$. The stimulus is prescribed and increased incrementally during the quasistatic solution; the thermal or optical process producing this change is not modeled.

The LCE material uses the same structural-density and material-selection interpolation introduced in Eq. (22), with passive and programmed endpoints

$$W^{\text{LCE}}_{\text{passive}}(\boldsymbol{F}) = \frac{\lambda}{2} \operatorname{tr}(\boldsymbol{E})^2 + \mu \, \boldsymbol{E} : \boldsymbol{E},$$

$$W^{\text{LCE}}_{\text{active}}(\boldsymbol{F}; \bar{\theta}, a) = W^{\text{LCE}}_{\text{passive}}(\boldsymbol{F}) - \frac{\beta}{2} S_0 a \, \mathbf{Q}(\bar{\theta}) : \boldsymbol{E},$$

where $\boldsymbol{E} = \frac{1}{2}(\boldsymbol{F}^T\boldsymbol{F} - \boldsymbol{I})$ is the Green–Lagrange strain, $\mathbf{Q}(\bar{\theta}) = 3\mathbf{n} \otimes \mathbf{n} - \boldsymbol{I}$, and $\mathbf{n} = (\cos\bar{\theta}, \sin\bar{\theta})^T$. The Lamé coefficients λ and μ are obtained from the prescribed Young's modulus and Poisson ratio. In the resulting design-dependent energy, ρ controls structural topology while ϕ determines how strongly the retained material follows the programmed response.

The examples use $E_{\text{LCE}} = 9340$ kPa, $\nu = 0.48$, $\beta = 575$ kPa, $S_0 = 0.40$, $p_\rho = p_\phi = 3$, and $\varepsilon_\rho = 10^{-6}$. The implementation uses $\boldsymbol{E} = (\boldsymbol{F}^T\boldsymbol{F} - \boldsymbol{I})/2$ throughout. Under plane strain, $E_{33} = 0$, and the planar director produces the same in-plane contraction as the corresponding plane-strain embedding.

4.3.1 Vertical-extension actuator. The vertical-extension example provides the simplest LCE actuation problem. The strip is 2×10, fixed along its lower edge; structural density is prescribed as $\rho = 1$, and the fields ϕ and θ are optimized under full activation (Fig. 8(a)). The objective

$$f_{\text{extension}} = -\frac{1}{w_{\text{top}} H} \int_{\Gamma_{\text{top}}} u_y \, \mathrm{d}S$$

is the negative mean upward displacement of the top edge normalized by the strip height $H = 10$, where $w_{\text{top}} = 2$ is the top-edge width. This problem isolates active-material placement and director programming before structural topology is introduced.

The final objective is −0.02673, corresponding to a mean rise of 0.2673 at the top edge, or 2.67% of the initial height (Fig. 8(c)). The final mean active fraction is 0.99881, below the prescribed limit of one, and Fig. 8(b) shows the final active-material and director fields. Starting from $\phi = 0.5$ and $\theta = 15°$, the optimization produces a nearly fully active strip. The director is nearly vertical away from the clamp. At cell centers more than one unit above the clamp, $\bar{\theta}$ ranges from 88.3° to 91.7°. Within one unit of the clamp, it ranges from 55° to 124°.

4.3.2 Load-bearing pusher. The pusher activates all three design fields. The two bottom attachment regions are clamped. In the support and central output pads, the density is held at one and the active fraction at zero in both the raw and physical fields. A downward traction is applied to the top of the central output region (Fig. 9(a)). The objective

$$f_{\text{pusher}} = -\frac{1}{w_{\text{out}} H} \int_{\Gamma_{\text{out}}} u_y \, \mathrm{d}S$$

maximizes the normalized upward displacement of this loaded boundary, where $w_{\text{out}} = 2$ is the output width and $H = 8$ is the domain height. The activated LCE must therefore generate upward

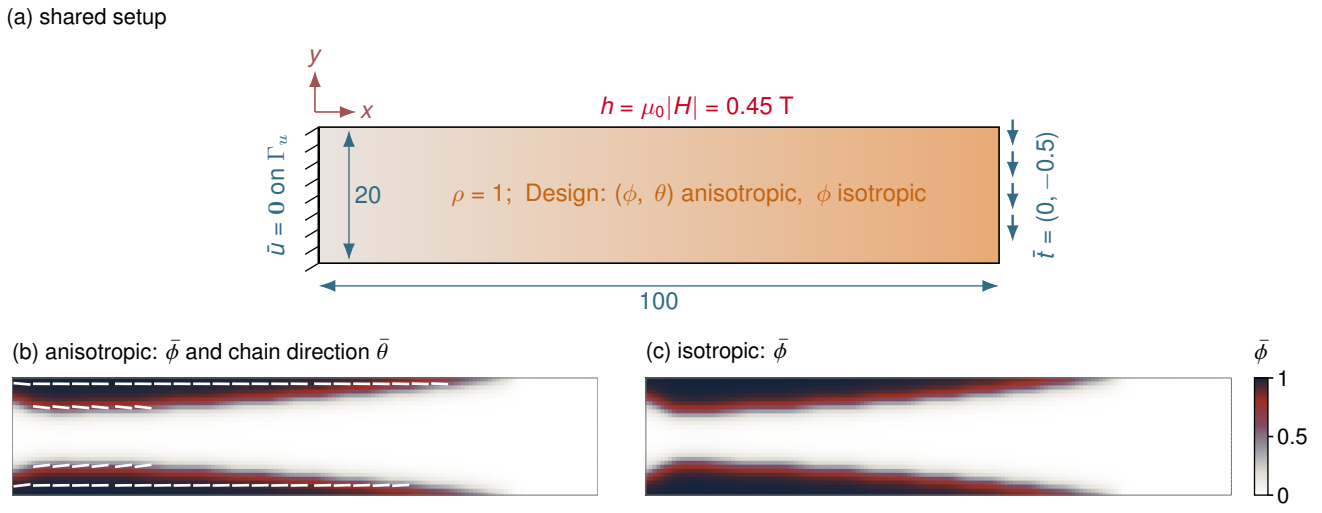


Fig. 6 MAE cantilevers. (a) Shared setup: a 100 × 20 cantilever with $\rho = 1$, clamped at the left edge, under a downward right-edge traction of 0.50 kPa and $h = 0.45$ T. (b) Final physical phase field $\bar{\phi}$ of the anisotropic MAE, with headless segments for the chain direction $\bar{\theta}$ where $\bar{\phi} > 0.5$. (c) Final $\bar{\phi}$ of the isotropic MAE. Panels (b) and (c) use the same scale.

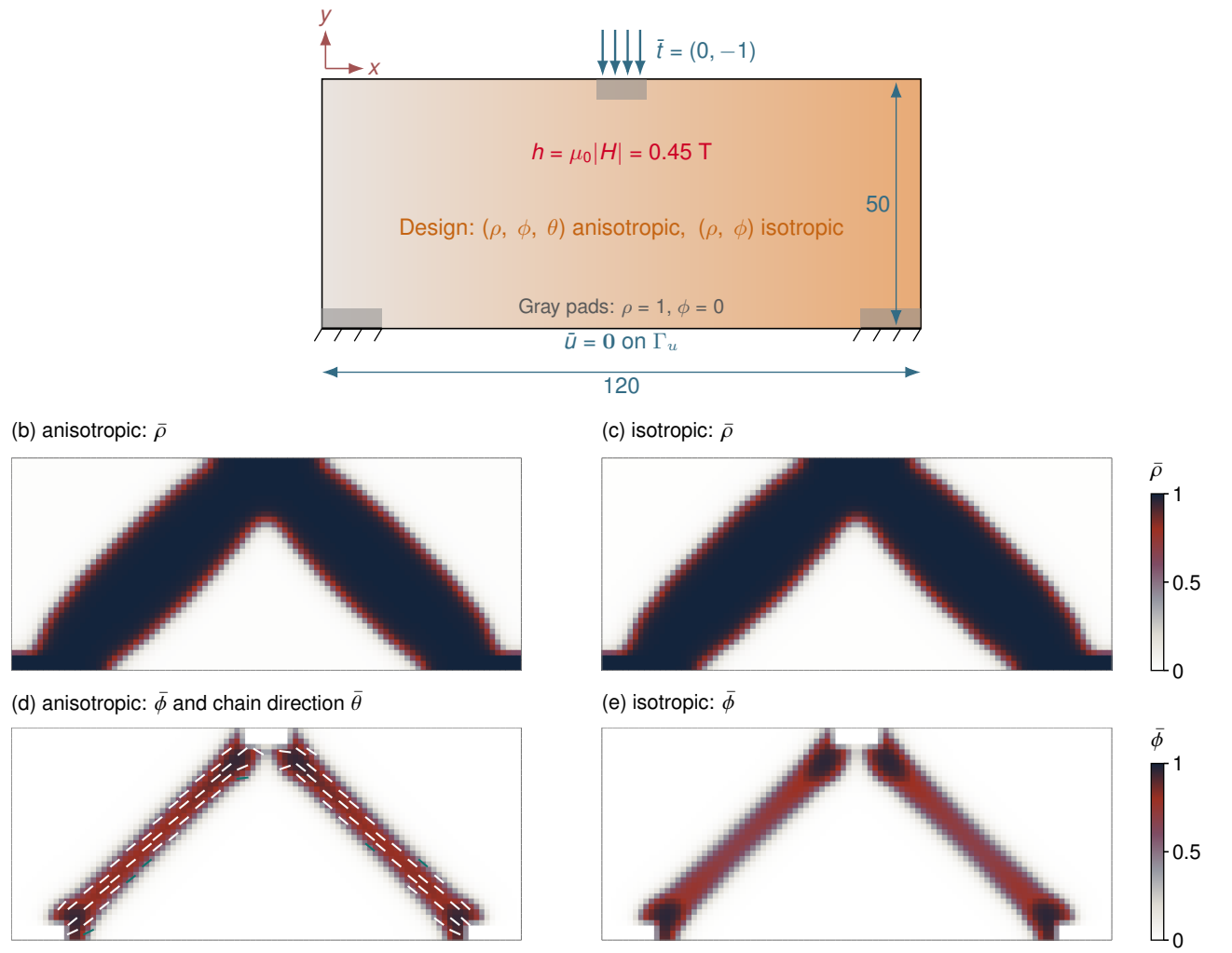


Fig. 7 MAE bridges, anisotropic (left) and isotropic (right). (a) Shared setup: the design domain, the clamped lower supports, the centered downward traction, and the three pads held at $\rho = 1$ and $\phi = 0$ (gray). (b, c) Final physical density $\bar{\rho}$. (d, e) Final physical phase field $\bar{\phi}$; the segments in (d) show the anisotropic chain direction $\bar{\theta}$ where $\bar{\rho}\bar{\phi}$ exceeds half its maximum. Each field uses the same scale in both models.

Table 5 Parameters prescribed in the MAE examples. Stress-like coefficients are in kPa; h and h_s are in tesla. Both families use $p_\rho = 3$, $\varepsilon_\rho = 10^{-6}$, and $p_\phi = 1$.

Implementation	Active endpoint	Silicone and common data
Anisotropic MAE	$A_{\mathrm{Ak}} = 326.75$, $a_{\mathrm{Ak}} = 2.785$, $b_{\mathrm{Ak}} = 3.40$, $r = 64.02$, $s = 318.16$, $h_s = 0.43$, $K_{\mathrm{Ak}} = 16000$	$A_{\mathrm{sil}} = 122$, $a_{\mathrm{sil}} = 0.28$, $b_{\mathrm{sil}} = 0.33$, $K_{\mathrm{sil}} = 6000$, $\theta_M = 0$, $\delta_\theta = 10^{-6}$
Isotropic MAE	$C_1^{\mathrm{Ga}} = 108$, $C_2^{\mathrm{Ga}} = 137$, $a_1 = 4.97$ m/A, $a_2 = 9.147\mu_0$, $b_1 = 174.685$, $b_2 = 13.024\mu_0$	$C_1^{\mathrm{sil}} = 270$, $C_2^{\mathrm{sil}} = 10.8$, $K = 12000$, $\mu_0 = 4\pi \times 10^{-7}$ T m/A

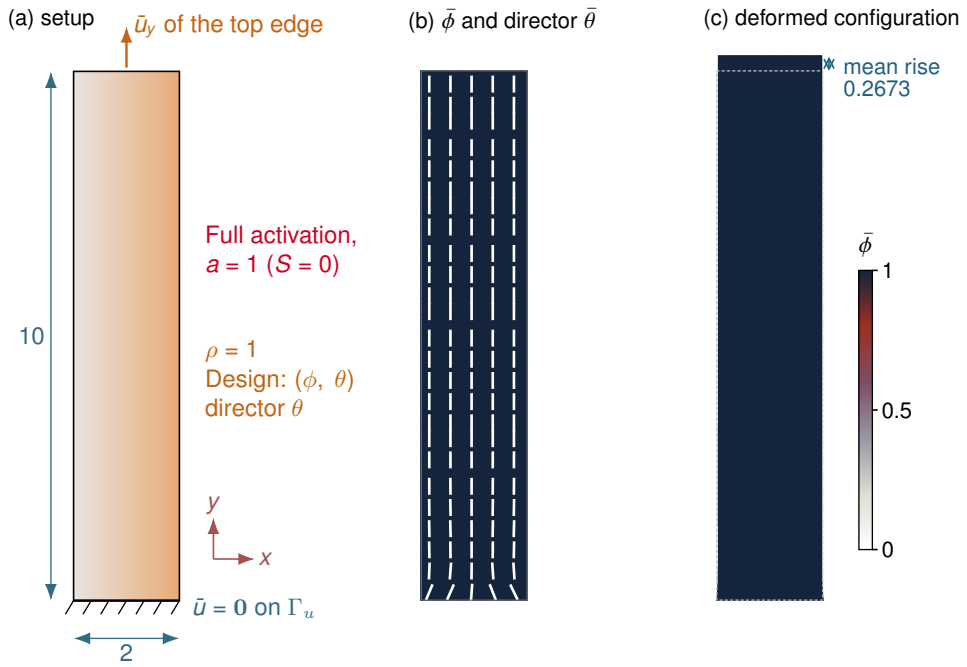


Fig. 8 LCE vertical-extension actuator. (a) Setup: a 2×10 strip with $\rho = 1$, clamped along its lower edge, under full activation $a = 1$; the design maximizes the mean rise of the top edge. (b) Final active-material field $\bar{\phi}$ with director segments $\bar{\theta}$; $\bar{\phi}$ lies between 0.9982 and 0.9990, with mean 0.99881. (c) Deformed configuration at true scale, colored by $\bar{\phi}$, over the dashed reference outline; the arrow marks the mean rise of 0.2673.

motion against the applied traction while the optimization simultaneously determines structural topology, active-material placement, and director orientation.

The domain is 12×8, with an output segment of width two subjected to a downward traction of 25 kPa. The objective decreases from 0.0044864 to −0.021498, corresponding to a final mean upward output displacement of 0.1720. The mean density is 0.49988, and the active fraction within the retained material is 0.49994. Both satisfy their limits of 0.50. Figure 9(b–d) shows the optimized structure, active-material field with director segments, and deformed configuration.

4.3.3 Distributed shape programming. The morphing strip considers distributed shape programming rather than a single output displacement. Structural density is prescribed as $\rho = 1$, and a small region at the center of the lower edge is fixed. The target displacement maps the two free arms to symmetric constant-curvature arcs (Fig. 10(a)). Listing 15 shows the corresponding displacement-tracking objective.

Equivalently, the fields ϕ and θ minimize

$$f_{\mathrm{strip}} = \frac{1}{L^2|\Omega_0|} \int_{\Omega_0} \left\| \boldsymbol{u} - \boldsymbol{u}^\star(\boldsymbol{X}) \right\|^2 \, \mathrm{d}V. \tag{24}$$

Here $L = 12$ is the strip length. Because the objective tracks displacement throughout the domain rather than only at the free ends, it promotes the prescribed curvature along the full strip.

```python
def build_objective(
    u_field,
    external_work,
    dx,
):
    """Minimize the mean squared error from the prescribed U
      shape."""
    target_displacement = prescribed_target_displacement()
    displacement_error = u_field - target_displacement
    domain_area = length * height

    # The L^2 and area factors make the objective dimensionless.
    # Discretization error remains mesh dependent.
    return (
        ufl.inner(displacement_error, displacement_error)
        / (length**2 * domain_area)
        * dx
    )
```

Listing 15: Distributed displacement-tracking objective for the LCE morphing strip.

The strip is 12×0.6, with a central support of width 0.6 and free-arm length 5.7. The target corresponds to circular arcs with 45° end rotations. The final tracking objective is 8.8230×10^{-5}, compared with 0.0052302 initially, while the final mean active fraction is 0.79958 against the prescribed limit of 0.80.

Figure 10(b) shows the final active-material and director fields, and Fig. 10(c) shows the deformed strip. Its end rotations are 27.0°

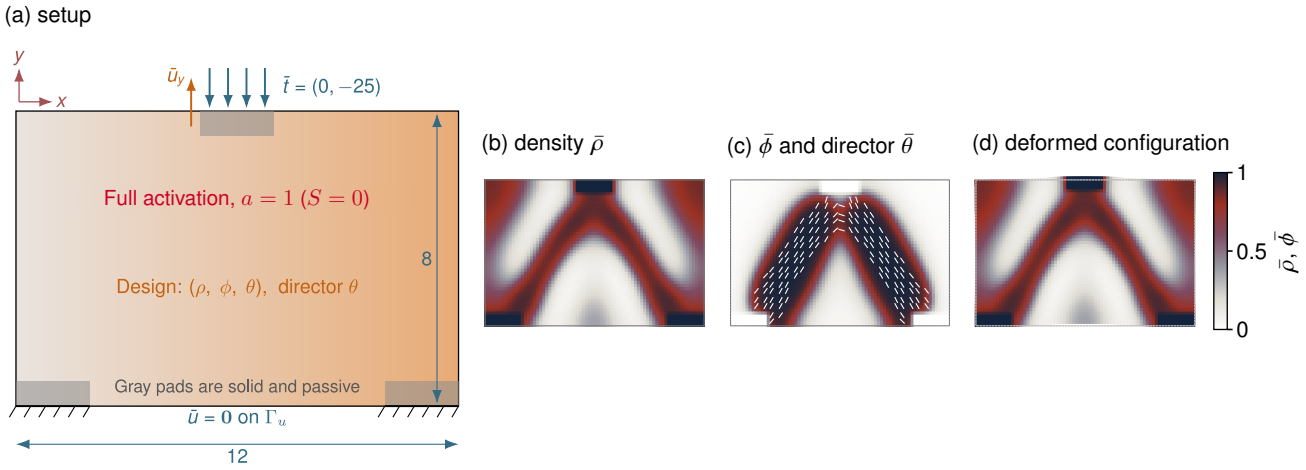


Fig. 9 LCE pusher. (a) Setup: a 12 × 8 domain with clamped bottom pads, a central output pad under a downward traction of 25 kPa, and full activation $a = 1$; the gray pads are held solid and passive. (b) Final physical density $\bar{\rho}$. (c) Final active-material field $\bar{\phi}$ with director segments where $\bar{\rho}\bar{\phi}$ exceeds half its maximum. (d) Deformed configuration colored by $\bar{\rho}$ over the dashed reference outline. The mean upward output displacement is 0.1720.

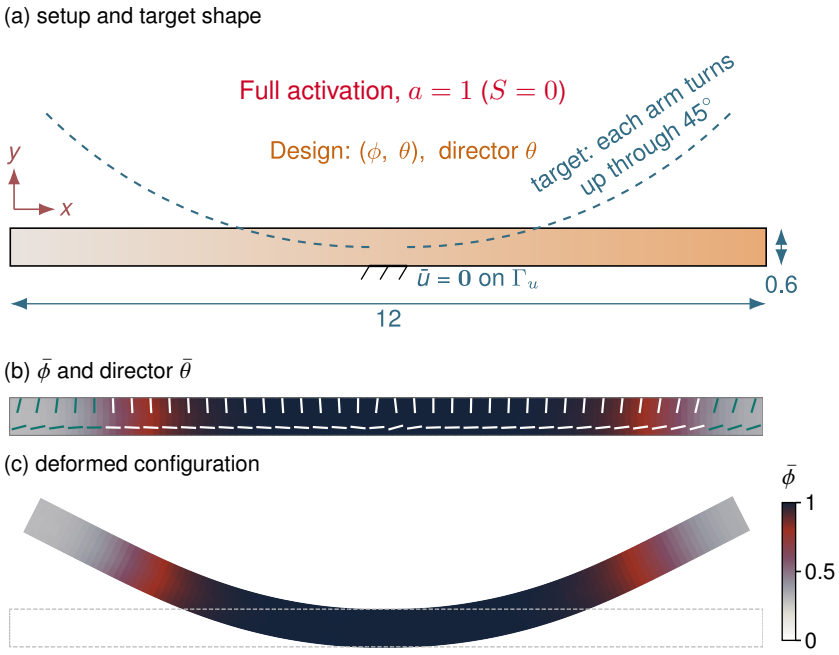


Fig. 10 LCE morphing strip. (a) Reference strip under full activation, the central clamp, and the target shape, whose arms turn up through 45°. (b) Final active-material field $\bar{\phi}$ with director segments $\bar{\theta}$. (c) Deformed strip colored by $\bar{\phi}$ over the reference outline. Panels (a)–(c) share one scale.

and 27.1° in magnitude, compared with the 45° target. The active fraction approaches its bound, and the design-change tolerance is met at iteration 58.

The hMSM, MAE, and LCE examples use different constitutive energies, physical interpretations of ϕ and θ, prescribed stimuli, and optimization objectives while retaining the same state, sensitivity, design-field, and MMA procedures. The examples therefore demonstrate reuse of the common formulation across distinct material models rather than physical equivalence between the material families.

4.4 Termination, feasibility, and scope of the examples. Five examples meet the design-change tolerance before their iteration limits, and six reach their limits. The isotropic MAE bridge meets the tolerance on its last allowed iteration. All eleven final designs satisfy their constraints after the final equilibrium solve.

Table 4 reports elapsed wall times on a 22-core Intel Xeon E5-2699 v4 workstation. Times include setup, optimization, the final equilibrium solve, and writing results. The two-dimensional runs used four MPI ranks and the dome used eight. The timing study ran one two-dimensional and one three-dimensional job at a time in parallel. The restorative beam was timed alone.

The examples demonstrate that the same formulation and software interfaces can accommodate changes in material energy, active design fields, objectives, load cases, and spatial dimension. They are not intended to provide experimental validation, a controlled comparison among material families, or performance predictions for extracted binary designs. Constitutive validation, discretization studies, and application-specific verification remain separate requirements.

5 Using and extending MatTO

MatTO can be extended in two ways: by defining a new optimization problem using an existing material model, or by adding a new material model together with a compatible problem definition. In either case, the closest supplied example provides a starting point. No changes to the shared modules in `src/matto` are required when the new study fits the formulation in Section 2.

5.1 Defining a new problem with an included material. A new problem can be created by first running the closest existing example unchanged. Listing 16 shows how to run the restorative beam example.

```
python examples/hMSM/input_beam.py
```

Listing 16: Running an existing MatTO example.

The corresponding input can then be copied and modified in the following order:

(1) construct the mesh and identify the displacement and traction boundaries;
(2) declare each design field required by the material model, specifying whether it is optimized or prescribed together with its bounds, initialization, parameterization operators, and any fixed regions;
(3) define the load cases, including their weights, mechanical loads, and prescribed stimuli;
(4) define the objective, design-only constraints, and any additional output fields; and
(5) specify the state, adjoint, filter, MMA, and output settings, assemble the `problem` dictionary, and call `OptimizationDriver(problem).run()`.

Every field required by the selected material model must be declared in `design_variables`. Setting `active=False` prescribes the physical field and removes it from the MMA design vector. Likewise, the names and scalar or vector shapes of the prescribed stimuli must agree with those required by the material model. Sections 3.1 and 3.4 describe the input structure and available design-field operators in detail.

5.2 Adding a material model. A new material family is introduced by defining a subclass of `matto.materials.Material` and at least one compatible input problem. The material class declares the design fields, prescribed stimuli, and material parameters that it requires and supplies the scalar free-energy density through

$$\texttt{energy(F, fields, stimuli)} \longrightarrow W.$$

Here W is a scalar UFL free-energy density and $\boldsymbol{F}$ is the deformation-gradient variable constructed by `StateProblem`. The method operates on the physical representations of the declared design fields and prescribed stimuli. Constitutive interpolation, density penalization, residual stiffness, and the physical interpretation of the design fields are therefore defined within the material model. Section 3.2 describes this interface in detail.

The accompanying input declares the required fields and stimuli and defines an objective and constraints consistent with their physical meanings. Before optimization, a new material implementation should be checked on a simple problem with a known response, including the passive or zero-stimulus limit when applicable. The supplied `check_material` helper evaluates the stress-free reference state at zero stimulus, the energy response under a prescribed stretch, and frame indifference under a sampled rigid rotation with vector stimuli rotated consistently. These checks provide implementation-level consistency tests and do not replace constitutive validation. The resulting objective and constraint sensitivities should also be checked against centered finite-difference directional derivatives on a coarse problem, as in Section 6.1.

6 Installation, verification, and reproducibility

MatTO is available at https://github.com/CEADpx/matto. The commands in Listing 17 create the environment from `environment.yml` and make the source available to Python.

```
git clone --branch v0.3.1 https://github.com/CEADpx/matto.git
cd matto
conda env create -f environment.yml
conda activate matto
export PYTHONPATH="$PWD/src"
```

Listing 17: Preparing the environment for MatTO.

The default test suite checks the MMA interface, design-field operators and their transpose actions, adjoint and constraint sensitivities, repeatability of sensitivity evaluation, material definitions, postprocessing, and stored regression values. A separate marked suite runs each supplied example for one optimization iteration and can be executed with different MPI process counts. Listing 18 gives the commands for both suites.

```
python -m pytest
mpirun -n 1 python -m pytest -m examples
mpirun -n 2 python -m pytest -m examples
mpirun -n 4 python -m pytest -m examples
```

Listing 18: Running the serial and MPI example tests.

These tests exercise the implementation and numerical consistency of the supplied examples; constitutive and discretization verification remain specific to the application being studied.

6.1 Directional-derivative verification. The finite-difference test included with MatTO provides a coarse regression check of the adjoint sensitivities. Additional centered-difference calculations were performed to verify the complete derivative path from the raw design variables through the parameterization, nonlinear equilibrium solution, and adjoint calculation. For a scalar functional $F(z)$ and an admissible raw-field direction $\boldsymbol{d}$, we compare

$$D_h F = \frac{F(z + h\boldsymbol{d}) - F(z - h\boldsymbol{d})}{2h}, \qquad D_{\mathrm{adj}} F = \nabla F(z) \cdot \boldsymbol{d},$$

using the scaled difference

$$e_h = \frac{|D_h F - D_{\mathrm{adj}} F|}{\max(|D_h F|, 10^{-10})}.$$

The derivative checks use the restorative beam, morphing surface, and anisotropic bridge setups in Sections 4.1.3, 4.1.4, and 4.2.2, respectively. The beam and bridge use the same meshes as the optimization runs. The morphing surface uses an $8 \times 8 \times 6$ quadrant mesh with one cell through the skin. For each centered difference, we perturb only the free raw design variables, hold the parameterization settings fixed, and re-solve the equilibrium problem. Table 6 gives the resulting directional-derivative errors.

For the restorative beam, the relative difference between the objective gradients computed on one and two MPI ranks was at most 1.4×10^{-10} across the three design fields. Repeating the evaluation at the same design on one rank reproduced the gradients exactly.

7 Conclusions and outlook

MatTO provides a material-independent framework for material and topology optimization of stimulus-responsive soft structures. The framework separates the problem definition and constitutive free-energy model from the finite-element, sensitivity, design-field, and optimization procedures. A material model specifies the physical meaning of the design fields and prescribed stimuli through a scalar UFL free-energy density, while the same numerical procedures construct equilibrium, evaluate adjoint sensitivities, map those

Table 6 Representative directional-derivative errors for three optimization examples.

Direction	h	e_h
2D hMSM beam: two-case objective		
ρ	10^{-4}	6.7×10^{-10}
ϕ	10^{-4}	3.1×10^{-10}
θ	10^{-2}	6.1×10^{-9}
Combined ρ, ϕ, θ	10^{-4}	7.8×10^{-10}
3D hMSM morphing surface: surface-tracking objective		
ρ	10^{-3}	1.4×10^{-9}
ϕ	10^{-4}	5.9×10^{-11}
θ	10^{-3}	7.2×10^{-9}
Anisotropic MAE bridge: compliance		
ρ	10^{-4}	2.6×10^{-9}
ϕ	10^{-3}	5.8×10^{-10}
Shifted phase residual		
ρ	10^{-3}	1.1×10^{-9}
ϕ	10^{-3}	1.3×10^{-10}

sensitivities to the raw design variables, and update the design with MMA. The number and physical meaning of the design fields can therefore change without introducing material-specific equations into the optimization procedure.

The eleven examples demonstrate this formulation for hard-magnetic soft materials, anisotropic and isotropic magneto-active elastomers, and liquid crystal elastomers. The problems include prescribed and optimized structural density, material composition, and orientation fields; single and multiple load cases; field-driven actuation and stiffening; displacement tracking; and a three-dimensional load-bearing morphing surface. These examples demonstrate reuse of the same formulation across different constitutive energies, stimuli, active field sets, and objectives rather than a controlled physical comparison among the material families.

The implementation is supported by automated regression, operator, sensitivity, and MPI tests. Additional centered-difference calculations verify the complete derivative path for multiple design fields, weighted load cases, the three-dimensional surface-tracking objective, and a density-dependent phase constraint. These checks establish consistency of the implemented sensitivities for the tested problems, while constitutive and discretization accuracy remain specific to the material model and application.

The present formulation considers displacement-based, quasistatic conservative response with prescribed stimuli and design-only constraints. The shared state formulation and hMSM material support both two and three dimensions, while the current LCE and MAE implementations are two-dimensional. Extensions to coupled field equations, state-dependent constraints, internal-variable evolution, fracture, plastic deformation, or time- and path-dependent response will require corresponding extensions of the state and sensitivity formulations. Within its present scope, MatTO provides a common computational basis for developing and comparing material and structural optimization problems without reconstructing the surrounding finite-element, sensitivity, and optimization procedures.

Acknowledgments

PKJ acknowledges support from the National Science Foundation through the Engineering Research Initiation (ERI) program under award No. 2502279, and from the South Dakota Board of Regents Competitive Research Grant (SDBOR CRG) program. These awards supported IG's work on this project at different stages. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies.

Conflict of Interest

No, there is no conflict of interest

Data Availability

The data that support the findings of this study are openly available in matto at https://github.com/CEADpx/matto, reference number v0.3.1.

References

[1] Bendsøe, M. P. and Sigmund, O., 2003, *Topology Optimization: Theory, Methods, and Applications*, Springer, Berlin, Heidelberg.

[2] Zhao, Z. and Zhang, X. S., 2022, "Topology Optimization of Hard-Magnetic Soft Materials," Journal of the Mechanics and Physics of Solids, **158**, p. 104628.

[3] Wang, C., Zhao, Z., and Zhang, X. S., 2023, "Inverse Design of Magneto-Active Metasurfaces and Robots: Theory, Computation, and Experimental Validation," Computer Methods in Applied Mechanics and Engineering, **413**, p. 116065.

[4] Fuchi, K., Ware, T. H., Buskohl, P. R., Reich, G. W., Vaia, R. A., White, T. J., and Joo, J. J., 2015, "Topology Optimization for the Design of Folding Liquid Crystal Elastomer Actuators," Soft Matter, **11**(37), pp. 7288–7295.

[5] Li, W. and Zhang, X. S., 2023, "Arbitrary Curvature Programming of Thermo-Active Liquid Crystal Elastomer via Topology Optimization," Computer Methods in Applied Mechanics and Engineering, **417**, p. 116393.

[6] Barrera, J. L., Cook, C., Lee, E., Swartz, K., and Tortorelli, D. A., 2024, "Liquid Crystal Orientation and Shape Optimization for the Active Response of Liquid Crystal Elastomers," Polymers, **16**, p. 1425.

[7] Galloway, I. and Jha, P. K., 2026, "Model-Informed Joint Material-Structural Optimization of Hard-Magnetic Soft Materials," arXiv preprint arXiv:2607.14397.

[8] Scroggs, M. W., Baratta, I. A., Richardson, C. N., and Wells, G. N., 2022, "Basix: A Runtime Finite Element Basis Evaluation Library," Journal of Open Source Software, **7**(73), p. 3982.

[9] Alnæs, M. S., Logg, A., Ølgaard, K. B., Rognes, M. E., and Wells, G. N., 2014, "Unified Form Language: A Domain-Specific Language for Weak Formulations of Partial Differential Equations," ACM Transactions on Mathematical Software, **40**(2), pp. 9:1–9:37.

[10] Baratta, I. A., Dean, J. P., Dokken, J. S., Habera, M., Hale, J. S., Richardson, C. N., Rognes, M. E., Scroggs, M. W., Sime, N., and Wells, G. N., 2023, "DOLFINx: The Next Generation FEniCS Problem Solving Environment," Zenodo, doi: 10.5281/zenodo.10447666.

[11] Balay, S. et al., 2024, "PETSc/TAO Users Manual," Argonne National Laboratory, Tech. Rep. ANL-21/39, Revision 3.22.

[12] Jia, Y., Wang, C., and Zhang, X. S., 2024, "FEniTop: A Simple FEniCSx Implementation for 2D and 3D Topology Optimization Supporting Parallel Computing," Structural and Multidisciplinary Optimization, **67**, p. 140.

[13] Akbari, E. and Khajehsaeid, H., 2021, "A Continuum Magneto-Mechanical Model for Magnetorheological Elastomers," Smart Materials and Structures, **30**, p. 015008.

[14] Garai, A. and Haldar, K., 2025, "Experiments and Modeling of Magneto-Stiffening Effects for Magnetoactive Polymer," International Journal of Mechanical Sciences, **286**, p. 109860.

[15] Lazarov, B. S. and Sigmund, O., 2011, "Filters in Topology Optimization Based on Helmholtz-Type Differential Equations," International Journal for Numerical Methods in Engineering, **86**(6), pp. 765–781.

[16] Wang, F., Lazarov, B. S., and Sigmund, O., 2011, "On Projection Methods, Convergence and Robust Formulations in Topology Optimization," Structural and Multidisciplinary Optimization, **43**, pp. 767–784.

[17] Svanberg, K., 1987, "The Method of Moving Asymptotes—A New Method for Structural Optimization," International Journal for Numerical Methods in Engineering, **24**(2), pp. 359–373.

Figure Captions and Table Headings

Figure 1. MatTO optimization workflow. The material model and problem definition supply the physical and numerical information required by the shared design-field parameterization, nonlinear equilibrium, sensitivity evaluation, and MMA update procedures.

Figure 2. Rotational hMSM actuator. (a) Wheel geometry with the clamped hub, the applied magnetic field, and the design fields. (b) Final particle volume fraction $\bar{\phi}$; arrows show the remanent-magnetization direction $\bar{\theta}$. (c) Deformed configuration colored by $\bar{\phi}$ over the dashed reference outline. The mean rim rotation is 62.64°.

Figure 3. Translational hMSM actuator. (a) Frame geometry with the left clamp, the applied magnetic field, and the target translation of the output. (b) Final particle volume fraction $\bar{\phi}$; arrows show the remanent-magnetization direction $\bar{\theta}$. (c) Deformed configuration colored by $\bar{\phi}$ over the reference outline.

Figure 4. Restorative hMSM beam. (a, b) Load cases 1 and 2: opposing right-edge tractions and applied magnetic fields on the beam clamped at the left edge. (c) Final physical density $\bar{\rho}$. (d) Final particle volume fraction $\bar{\phi}$; arrows show the remanent-magnetization direction $\bar{\theta}$.

Figure 5. Three-dimensional hMSM morphing surface. (a) Problem setup, with dimensions in millimeters. (b, c) Final design of the computed quadrant and the reflected full slab, viewed from below and including the skin. The dashed box outlines the quadrant, and the dashed lines in (c) mark the symmetry planes. Cells are displayed where $\bar{\rho} > 0.5$. Color denotes particle volume fraction, and arrows show remanent directions on bottom faces where $\bar{\phi} \geq 0.1$. (d, e) Target and computed top-surface displacements on a common scale in millimeters. (f) Profiles along $Y = 0$, indicated by dotted lines in (d, e).

Figure 6. MAE cantilevers. (a) Shared setup: a 100×20 cantilever with $\rho = 1$, clamped at the left edge, under a downward right-edge traction of 0.50 kPa and $h = 0.45$ T. (b) Final physical phase field $\bar{\phi}$ of the anisotropic MAE, with headless segments for the chain direction $\bar{\theta}$ where $\bar{\phi} > 0.5$. (c) Final $\bar{\phi}$ of the isotropic MAE. Panels (b) and (c) use the same scale.

Figure 7. MAE bridges, anisotropic (left) and isotropic (right). (a) Shared setup: the design domain, the clamped lower supports, the centered downward traction, and the three pads held at $\rho = 1$ and $\phi = 0$ (gray). (b, c) Final physical density $\bar{\rho}$. (d, e) Final physical phase field $\bar{\phi}$; the segments in (d) show the anisotropic chain direction $\bar{\theta}$ where $\bar{\rho}\bar{\phi}$ exceeds half its maximum. Each field uses the same scale in both models.

Figure 8. LCE vertical-extension actuator. (a) Setup: a 2×10 strip with $\rho = 1$, clamped along its lower edge, under full activation $a = 1$; the design maximizes the mean rise of the top edge. (b) Final active-material field $\bar{\phi}$ with director segments $\bar{\theta}$; $\bar{\phi}$ lies between 0.9982 and 0.9990, with mean 0.99881. (c) Deformed configuration at true scale, colored by $\bar{\phi}$, over the dashed reference outline; the arrow marks the mean rise of 0.2673.

Figure 9. LCE pusher. (a) Setup: a 12×8 domain with clamped bottom pads, a central output pad under a downward traction of 25 kPa, and full activation $a = 1$; the gray pads are held solid and passive. (b) Final physical density $\bar{\rho}$. (c) Final active-material field $\bar{\phi}$ with director segments where $\bar{\rho}\bar{\phi}$ exceeds half its maximum. (d) Deformed configuration colored by $\bar{\rho}$ over the dashed reference outline. The mean upward output displacement is 0.1720.

Figure 10. LCE morphing strip. (a) Reference strip under full activation, the central clamp, and the target shape, whose arms turn up through 45°. (b) Final active-material field $\bar{\phi}$ with director segments $\bar{\theta}$. (c) Deformed strip colored by $\bar{\phi}$ over the reference outline. Panels (a)–(c) share one scale.

Table 1. Notation used in the common continuum and optimization formulation.

Table 2. Responsibilities of the principal modules in `src/matto`.

Table 3. Responsive-material examples and their active design fields. A field omitted from the active set is prescribed when required by the material model.

Table 4. Numerical results for the eleven examples. f_0 and f_f are the objectives at the initial design and after solving equilibrium at the final design. Δ is the final normalized design change over the free raw coefficients. Wall times are rounded to the nearest minute. The dome cell count refers to the computed quadrant.

Table 5. Parameters prescribed in the MAE examples. Stress-like coefficients are in kPa; h and h_s are in tesla. Both families use $p_\rho = 3$, $\varepsilon_\rho = 10^{-6}$, and $p_\phi = 1$.

Table 6. Representative directional-derivative errors for three optimization examples.