Structural and Multidisciplinary Optimization (2025) 68:153
https://doi.org/10.1007/s00158-025-04055-2

EDUCATIONAL PAPER

PolyPlas: a Python implementation of a topology optimization framework for plasticity with unstructured polygonal finite elements

Emily Alcazar[1] · Jonathan B. Russ[1] · Glaucio H. Paulino[1,2]

Received: 19 February 2025 / Revised: 24 April 2025 / Accepted: 3 June 2025

Abstract
We present PolyPlas, a Python implementation for a structural topology optimization framework considering von Mises plasticity with unstructured polygonal finite element meshes. The modular structure of this code is inspired by PolyTop—an early educational code for compliance minimization for linear elastic material. For the purpose of open-source access and extensibility, PolyPlas is fully realized in Python. The nonlinear forward problem is solved via a Newton Raphson procedure with backtracking line search for improved convergence stability. The path-dependent sensitivity analysis is conducted using the adjoint method and a detailed discussion on the path-dependent algorithm and implementation of the sensitivity analysis is included herein. Finally, several numerical examples are presented to illustrate the capabilities of PolyPlas in solving topology optimization problems considering von Mises plasticity, resulting in structures with high energy absorption. PolyPlas is wholly intended for educational purposes and to motivate further advancement in the field of topology optimization considering energy-dissipative phenomena.

Keywords Topology optimization · Elastoplasticity · Educational code · Open-source

1 Introduction

Since its inception in the work of Bendsøe and Kikuchi (1988), topology optimization has become a powerful engineering design method to optimize placement of material to extremize a specified design objective. Although the field is extensive with many significant advancements, the majority of the work in the literature is limited to linear or nonlinear elasticity (Sigmund and Maute 2013). Nonetheless, there have been several contributions investigating material inelasticity in conjunction with topology optimization frameworks that handle the history-dependent deformation and energy-dissipative phenomenon during the design process. One of the first efforts in considering elastoplasticity in structural optimization was done by Yuge and Kikuchi (1995), who presented a framework that computed the elastoplastic analysis based on the homogenization method to inform the design of a frame structure based on linear Timoshenko beam theory. Later Maute et al. (1998) proposed a topology optimization formulation to maximize structural ductility considering von Mises material with linear, isotropic hardening; however, they simplify the sensitivity analysis by neglecting some terms. Additional work by Kato et al. (2015) proposed a framework for the design of composite materials undergoing elastoplastic deformation making some assumptions which simplify the sensitivity analysis. Bogomolny and Amir (2012) approached the design of steel-reinforced concrete structures by incorporating Drucker–Prager and von Mises yield criteria within the topology optimization formulation. Their sensitivity analysis was derived analytically using the adjoint method following the framework established by Michaleris et al. (1994); Vidal and Haber (1993), thus representing the first complete analytical derivation of the sensitivity analysis for topology optimization considering elastoplasticity. The adjoint method has since become the standard approach in most of the topology optimization frameworks that consider elastoplasticity and is further elaborated upon in a unified framework for nonlinear path-dependent sensitivity analysis by Alberdi et al. (2018).

Responsible Editor: Junji Kato.

The open-source python code for PolyPlas may be found in the git repository: https://github.com/emilyalcazar1/PolyPlas.gi

✉ Glaucio H. Paulino
gpaulino@princeton.edu

[1] Department of Civil and Environmental Engineering, Princeton University, Princeton, NJ 08544, USA

[2] Princeton Materials Institute (PMI), Princeton University, Princeton, NJ 08544, USA

Published online: 19 August 2025

More recent works of elastoplastic topology optimization frameworks have considered cyclical loading with kinematic hardening (Li et al. 2017a) and anisotropic material behavior (Zhang et al. 2017). Additional advancements to these elastoplastic frameworks include the consideration of failure criteria. For example, Li et al. (2017b) presented a novel framework to consider elastoplasticity coupled with the Lemaitre damage model to account for local failure constraints during the design. Other works that considered elastoplasticity with failure criteria include the work by Alberdi and Khandelwal (2017) wherein shear loading constraints are considered in the optimization framework using an aggregation function. Kuci and Jansen (2022) later presented a framework to consider elastoplastic materials that handle local stress constraints in a level-set framework, adapting the formulation presented by Senhora et al. (2020) via the Augmented Lagrangian approach. In contrast, Amir (2017) proposed a method to indirectly generate stress-constrained designs for linear elastic structures without explicitly imposing stress constraints by utilizing the elastoplastic model subject to a single global constraint on the accumulated plastic strain. In the work by Russ and Waisman (2021) both local ductile failure constraints and buckling are incorporated. Elastoplastic topology optimization frameworks have also been developed to design two-dimensional periodic materials with high energy absorption as in the work by Alberdi and Khandelwal (2019) where computational homogenization is used to characterize the appropriate anisotropic plasticity behavior for the representative unit cell, and in the work by Abueidda et al. (2021) which informed the design of a three-dimensional periodic elastoplastic architected material. Additional frameworks have also been presented to approximate the elastoplastic material behavior based on path-independent nonlinear elasticity (Zhao et al. 2019, 2020) and on deformation plasticity (Li et al. 2024); thus significantly reducing the computational effort. However, these models are strictly limited to proportional loading and do not capture the dissipative physics. Several works have also incorporated finite deformation kinematics into elastoplastic topology optimization frameworks (Wallin et al. 2016; Ivarsson et al. 2021; Han et al. 2024).

In addition to these developments in topology optimization frameworks, there has been concurrent progress in the broad dissemination of topology optimization via open-source, educational programs (Wang et al. 2021), beginning with the first contribution `top99` which introduced a 99-line Matlab code for compliance minimization topology optimization for linear elasticity (Sigmund 2001). Subsequent works included `Top3D`, a 3D topology optimization program for compliance minimization which incorporated additional implementation details for heat conduction and compliant mechanism problems considering linear elasticity (Liu and Tovar 2014). Recently there have also been extensions to speed-up and improve the performance of these programs by means of vectorization and array preallocation (Andreassen et al. 2011), parallel programming (Aage et al. 2015) and other acceleration techniques (Ferrari and Sigmund 2020). Extensions to more complex physics like linear buckling analysis (Ferrari et al. 2021) and homogenization theory (Xia and Breitkopf 2015) have also been provided. The presented educational program in this work is inspired by `PolyTop`, a compliance minimization framework for linear elasticity considering unstructured polygonal elements (Talischi et al. 2012b). `PolyTop` ignited a series of topology optimization programs including `PolyTopFluid` which was developed to handle Stokes equation where the polygonal elements were demonstrated to have improved stability for incompressible viscous flow using a low-order mixed finite element formulation (Pereira et al. 2016). Later `PolyMat` was presented to handle multiple linear elastic materials (Sanders et al. 2018). `PolyTop3D` included the extension to 3D topology optimization for compliance minimization using the virtual element method approach (Chi et al. 2020). More recently, `PolyStress` provided the extension to local stress constraints in the topology optimization framework, also with the option to include nonlinear elasticity (Giraldo-Londoño and Paulino 2021b). Finally, `PolyDyna` extended the `PolyTop` suite to structural dynamics using an HHT-α time integration scheme (Giraldo-Londoño and Paulino 2021a).

The entirety of the aforementioned educational topology optimization programs are limited to linear or nonlinear elasticity. This work provides an original attempt at an open-source, educational topology optimization framework considering elastoplasticity. In this work, we present an open-source code for topology optimization considering von Mises elastoplasticity with a linear hardening rule. Here we present two major contributions: (1) a thorough explanation of the path-dependent sensitivity analysis for von Mises plasticity together with a detailed discussion on the implementation of the history-dependent algorithm into the code and (2) providing an open-source topology optimization program that considers energy-dissipative phenomenon in an extendable, modular framework. The program is implemented in Python, making it accessible for academic and industry users alike. With its modular structure, this framework aims to promote further developments and extensions.

The remainder of the paper is organized as follows: Sect. 2 begins by describing the nonlinear finite element analysis. In Sect. 3, the density-based topology optimization formulation is described. Section 4 discusses the path-dependent

sensitivity analysis in detail, followed by a discussion of the organization of PolyPlas in Sect. 5. Finally, three numerical examples are presented to demonstrate the capabilities of the software in Sect. 6. Concluding remarks are made in Sect. 7 with additional information provided in the appendices.

2 Nonlinear finite element analysis framework

Consistent with the former PolyTop literature, we begin by describing the classical continuum topology optimization problem within a general framework. The goal of topology optimization is to optimize the shape of a structure, $\omega \subseteq \mathbb{R}^d$ in d space dimensions where $d = 2$ in this work. This may be mathematically expressed by:

$$\begin{aligned} &\inf_{\omega \in \mathcal{O}} \quad f(\omega, \boldsymbol{u}) \\ &\text{s.t.} \quad g_i(\omega, \boldsymbol{u}) \leq 0, i = 1, .., K \end{aligned} \tag{1}$$

where f is the objective function to be minimized subject to K number of constraints, g_i. The symbol $\mathcal{O}$ represents the set of admissible shapes for ω. Each evaluation of the objective and constraint functions is dependent on the solution of the boundary value problem, $\boldsymbol{u}$. The strong form of the initial boundary value problem for a solid body satisfying static equilibrium in absence of body forces is shown below:

$$\begin{aligned} &\boldsymbol{\nabla} \cdot \boldsymbol{\sigma} = 0 \quad \text{in} \ \ \Omega \\ &\boldsymbol{u} = \bar{\boldsymbol{u}}_D \quad \text{on} \ \ \tilde{\Gamma}_D \subseteq \Gamma \\ &\boldsymbol{u} = \boldsymbol{0} \quad \text{on} \ \ \Gamma_D \subseteq \Gamma \end{aligned} \tag{2}$$

Here the shape ω is within the closed domain Ω with boundary Γ, $\boldsymbol{\sigma}$ is the Cauchy stress tensor and $\boldsymbol{\nabla}\cdot$ is the divergence operator. The Dirichlet boundary conditions are separated into zero displacements imposed on the partition of the boundary $\Gamma_D \subseteq \Gamma$ and nonzero prescribed displacements imposed on $\tilde{\Gamma}_D \subseteq \Gamma$ with $\tilde{\Gamma}_D \cap \Gamma_D = \emptyset$ (see Fig. 1).

The Cauchy stress tensor may be decomposed into volumetric and deviatoric components such that $\boldsymbol{\sigma} = p\boldsymbol{I} + \boldsymbol{s}$ where $p = \frac{1}{3}\boldsymbol{I} : \boldsymbol{\sigma}$ is the pressure stress and $\boldsymbol{s} = \mathbb{P}_{dev} : \boldsymbol{\sigma}$ is the deviatoric stress. Note that $\boldsymbol{I}$ is the second order identity tensor and $\mathbb{P}^{dev} \equiv \mathbb{I}^s - \frac{1}{3}\boldsymbol{I} \otimes \boldsymbol{I}$ is the deviatoric projection tensor constructed using the fourth order symmetric identity tensor, $\mathbb{I}^s_{ijkl} = \frac{1}{2}\left(\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk}\right)$.

2.1 Weak form

The solution of the boundary value problem $\boldsymbol{u} \in \mathcal{V}$ is determined via the finite element method beginning with the weak form of the initial boundary value problem where the space of admissible displacements

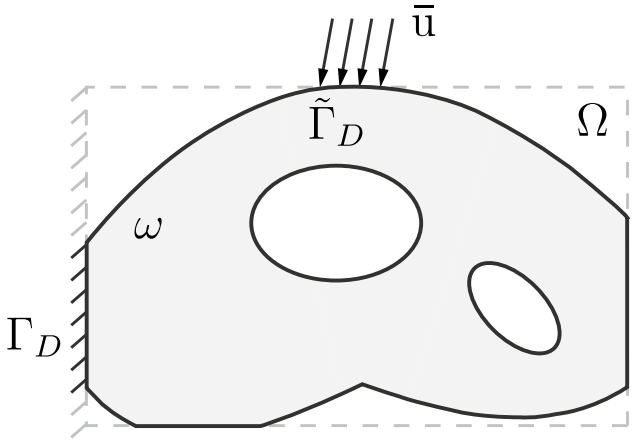


Fig. 1 Design domain and boundary conditions of the solid body [adapted from Talischi et al. (2012b)]

$\mathcal{V} = \left\{\boldsymbol{u} \in H^1\left(\omega, \mathbb{R}^2\right) : \boldsymbol{u}|_{\Gamma_D} = \boldsymbol{0},\ \boldsymbol{u}|_{\tilde{\Gamma}_D} = \bar{\boldsymbol{u}}_D\right\}$. We start by writing the total potential energy as a three-field variational principle in anticipation of employing the mean dilatation formulation (Nagtegaal et al. 1974; Simo et al. 1985) to avoid potential volumetric locking. The terms with the subscript $(\cdot)_{n+1}$ are associated with the pseudo-time from the standard backward Euler time integration:

$$\Pi(\boldsymbol{u}_{i+1}, \Theta_{i+1}, p_{i+1}) = \int_\Omega \left[\frac{1}{2}\kappa\Theta^2_{i+1} + G\boldsymbol{\varepsilon}^{e,dev}_{i+1} : \boldsymbol{\varepsilon}^{e,dev}_{i+1} + p_{i+1}(\boldsymbol{\nabla}\cdot\boldsymbol{u}_{i+1})\right]. \tag{3}$$

The independent field variables at the next time increment include the displacement field $\boldsymbol{u}_{i+1}$, a volume-like field variable Θ_{i+1}, and the pressure p_{i+1}. Here κ represents the bulk modulus, G represents the shear modulus, and $\boldsymbol{\varepsilon}^{e,dev}_{i+1}$ is the deviatoric component of the elastic strain tensor,

$$\boldsymbol{\varepsilon}^{e,dev}_{i+1} = \mathbb{P}^{dev} : (\boldsymbol{\varepsilon}_{i+1} - \boldsymbol{\varepsilon}^p_{i+1}) \tag{4}$$

with

$$\boldsymbol{\varepsilon}_{i+1} = \boldsymbol{\nabla}^s\boldsymbol{u}_{i+1} \tag{5}$$

representing the total strain tensor under the assumption of small deformation. The Euler Lagrange equations associated with the minimization of this potential energy are then expressed by

$$\delta\Pi_{\boldsymbol{u}_{i+1}} = \int_\Omega \left(\boldsymbol{s}_{i+1} : \boldsymbol{\nabla}\delta\boldsymbol{u} + p_{i+1}\boldsymbol{\nabla}\cdot\delta\boldsymbol{u}\right)dV = 0, \quad \forall\delta\boldsymbol{u} \in V_0 \tag{6}$$

$$\delta\Pi_{\Theta_{i+1}} = \int_\Omega \delta\Theta\left(\kappa\Theta_{i+1} - p_{i+1}\right)dV = 0, \quad \forall\delta\Theta \in \mathcal{W} \tag{7}$$

$$\delta\Pi_{p_{i+1}} = \int_\Omega \delta p\big(\boldsymbol{\nabla}\cdot\boldsymbol{u}_{i+1} - \Theta_{i+1}\big)dV = 0, \quad \forall \delta p \in \mathcal{X} \tag{8}$$

where $\delta\boldsymbol{u} \in V_0 \equiv \left\{\boldsymbol{u} \in H^1(\omega,\mathbb{R}^2),\ \delta\boldsymbol{u}|_{\bar{\Gamma}_D \bigcup \Gamma_D} \equiv \boldsymbol{0}\right\}$ represents the virtual displacement, $\delta\Theta \in \mathcal{W} \equiv \left\{\Theta \in L^2(\omega,\mathbb{R})\right\}$ represents the virtual dilatation, and $\delta p \in \mathcal{X} \equiv \left\{p \in L^2(\omega,\mathbb{R})\right\}$ denotes the virtual pressure.

2.2 Finite element discretization

An approximate solution is then obtained using the Galerkin finite element method in the usual manner, searching for a solution in a finite dimensional subspace of the infinite dimensional counterparts. The original domain is partitioned into N_{elem} non-overlapping elements which span the discretized domain $\bar{\Omega} \subseteq \Omega$ such that $\cup_{l=1}^{N_{elem}} \bar{\Omega}_\ell = \bar{\Omega}$ and satisfy $\bar{\Omega}_k \cap \bar{\Omega}_\ell = \emptyset\ \forall k \neq \ell$.

A standard Lagrange finite element interpolation is used for the displacement field approximation, $\boldsymbol{u}^{(h)}$, constructed using piecewise linear shape functions defined over each element, forming the basis for the underlying finite dimensional subspace $\mathcal{V}_h \subseteq \mathcal{V}$. The approximations for the dilatation field, $\Theta^{(h)} \in \mathcal{W}_h \subseteq \mathcal{W}$, and pressure field $p^{(h)} \in \mathcal{X}_h \subseteq \mathcal{X}$, are interpolated in a piecewise constant manner over each finite element, consistent with the mean dilatation formulation. Following the Galerkin procedure, the virtual displacement, virtual dilatation, and virtual pressure fields are interpolated over each finite element using the same shape functions as their non-virtual counterparts. The linear mapping from the vector of nodal displacements, $\bar{\boldsymbol{u}}_{i+1}$, to the displacement approximation in a given finite element is represented via an array of shape function values, $\boldsymbol{N}^u$, evaluated at the relevant spatial position. The corresponding gradient and divergence of the displacement approximation is also constructed in a similar manner via the arrays of shape function derivatives, $\boldsymbol{B}^u$ and $\boldsymbol{B}^u_{div}$, respectively. That is,

$$\begin{aligned} \boldsymbol{u}^{(h)}_{i+1} &\approx \boldsymbol{N}^u\bar{\boldsymbol{u}}_{i+1} \\ \boldsymbol{\nabla}^s\boldsymbol{u}^{(h)}_{i+1} &\approx \boldsymbol{B}^u\bar{\boldsymbol{u}}_{i+1} \\ \boldsymbol{\nabla}\cdot\boldsymbol{u}^{(h)}_{i+1} &\approx \boldsymbol{B}^u_{div}\bar{\boldsymbol{u}}_{i+1} \end{aligned} \tag{9}$$

for the trial displacement solution, and

$$\begin{aligned} \delta\boldsymbol{u}^{(h)} &\approx \boldsymbol{N}^u\delta\bar{\boldsymbol{u}} \\ \boldsymbol{\nabla}^s\delta\boldsymbol{u}^{(h)} &\approx \boldsymbol{B}^u\delta\bar{\boldsymbol{u}} \\ \boldsymbol{\nabla}\cdot\delta\boldsymbol{u}^{(h)} &\approx \boldsymbol{B}^u_{div}\delta\bar{\boldsymbol{u}} \end{aligned} \tag{10}$$

for the corresponding virtual displacements. Substituting the finite element approximations into Eq. (6) and invoking the standard procedure with the nodal virtual displacements, $\boldsymbol{\delta\bar{u}}$, we arrive at the global residual contribution from a single element,

$$\boxed{\boldsymbol{R}^e_{i+1} = \int_{\Omega_e} \boldsymbol{B}^{u^T} : \boldsymbol{s}_{i+1}dV + p_{i+1}\int_{\Omega_e}\boldsymbol{B}^{u^T}_{div}dV.} \tag{11}$$

A similar process for Eqs. (7) and (8) results in the expressions for the constant dilatation and pressure interpolation over each element. Namely,

$$\boxed{\Theta_{i+1} = \frac{1}{V_e}\int_{\Omega_e}\boldsymbol{B}^u_{div}\bar{\boldsymbol{u}}_{i+1}dV} \tag{12}$$

and

$$\boxed{p_{i+1} = \kappa\Theta_{i+1}} \tag{13}$$

which may be substituted into Eq. (11), condensing the dilatation and pressure degrees of freedom out of the global system of equations. The element Jacobian matrix may then be computed via

$$\boxed{\boldsymbol{J}^e_{i+1} = \frac{\partial\boldsymbol{R}^e_{i+1}}{\partial\bar{\boldsymbol{u}}_{i+1}} = \int_{\Omega_e}\boldsymbol{B}^{u^T} : \mathbb{C}^{dev}_{i+1} : \boldsymbol{B}^u dV + \frac{\kappa}{V_e}\left(\int_{\Omega_e}\boldsymbol{B}^{u^T}_{div}dV\right)\left(\int_{\Omega_e}\boldsymbol{B}^u_{div}dV\right)} \tag{14}$$

with $\mathbb{C}^{dev}_{i+1}$ representing the deviatoric component of the consistent tangent tensor (Simo and Hughes 2006), provided in Appendix A. The element residual vectors and Jacobian matrices are assembled into their global counterparts via the standard finite element assembly operations:

$$\boldsymbol{R}_{i+1} = \mathop{\mathcal{A}}_{e=1}^{N_{elem}}\boldsymbol{R}^e_{i+1}, \quad \boldsymbol{J}_{i+1} = \mathop{\mathcal{A}}_{e=1}^{N_{elem}}\boldsymbol{J}^e_{i+1} \tag{15}$$

The nonlinear system of equations is solved using the Newton Raphson method, updating the nodal displacement vector by

$$\boxed{\bar{\boldsymbol{u}}^{m+1}_{i+1} = \bar{\boldsymbol{u}}^m_{i+1} - \boldsymbol{J}^{m^{-1}}_{i+1}\boldsymbol{R}^m_{i+1}} \tag{16}$$

where m represents the Newton iteration number. The iteration continues until the relative ℓ_2 norm of the global residual vector falls below a specified tolerance.

2.3 Elastoplastic constitutive model

In this subsection, we describe the main pieces of the classical von Mises plasticity model used in this work. Under the assumption of small deformation, the total strain may be additively decomposed into its elastic and plastic components by,

$$\boldsymbol{\varepsilon} = \boldsymbol{\varepsilon}^e + \boldsymbol{\varepsilon}^p. \tag{17}$$

The yield function depends only on the deviatoric stress and the accumulated plastic strain, α. It is defined by the following,

$$\Phi(\boldsymbol{s}, \alpha) = \sqrt{\frac{3}{2}\boldsymbol{s} : \boldsymbol{s}} - \sigma_y(\alpha) \tag{18}$$

where the first term on the right corresponds to the von Mises stress and $\sigma_y(\alpha)$ is the current yield stress which may evolve throughout the deformation history. Here we assume a linear hardening rule,

$$\sigma_y(\alpha) = \sigma_y^0 + H\alpha \tag{19}$$

where H is the material hardening modulus and σ_y^0 is the initial value of the yield stress. The elastic response is assumed to be linear and isotropic, with the deviatoric stress related to the elastic strain in the classical manner, $\boldsymbol{s} = 2G\mathbb{P}^{dev} : (\boldsymbol{\varepsilon} - \boldsymbol{\varepsilon}^p)$. In this setting the shear modulus and bulk modulus may be defined in terms of the elastic modulus E and Poisson's ratio ν by $G = E/2(1+\nu)$ and $\kappa = E/3(1-2\nu)$, respectively.

An associative flow rule is assumed which governs the evolution of the plastic strain,

$$\dot{\boldsymbol{\varepsilon}}^p = \dot{\gamma}\boldsymbol{N} = \dot{\gamma}\sqrt{\frac{3}{2}}\frac{\boldsymbol{s}}{||\boldsymbol{s}||} \tag{20}$$

where $\dot{\gamma}$ is the plastic multiplier and $\boldsymbol{N} = \partial\Phi/\partial\boldsymbol{\sigma}$ governs the direction of plastic flow. Additionally, the Karush-Kuhn-Tucker (KKT) conditions ensure that the stress state is admissible and that there is plastic flow only when the stress is on the yield surface (Simo and Hughes 2006).

$$\Phi \leq 0,\ \dot{\gamma} \geq 0,\ \dot{\gamma}\Phi = 0. \tag{21}$$

Finally, the consistency condition ensures that the stress state persists on the yield surface in the event of plastic flow, precluding the possibility of a future inadmissible state, namely,

$$\dot{\gamma}\dot{\Phi} = 0. \tag{22}$$

Together, these equations form an initial value problem that describes the evolution of the material's internal state over the given load path. The continuous problem is then solved by discretizing the equations in pseudo-time using a backward Euler scheme (de Souza Neto et al. 2011). This leads to the fully implicit radial return mapping algorithm in Appendix A for isotropic von Mises plasticity which is simplified by coaxiality. The local residual equations in their incremental form ($i = 1, ..., N_{steps}$) are gathered into a vector, $\boldsymbol{H}_i^{e_q}$, containing the expressions for the update of the accumulated plastic strain, the plastic multiplier increment, and the plastic strain. For *elastic* loading the local residual equations correspond to,

$$\boldsymbol{H}_i^{e_q} = \begin{bmatrix} h_{i,1}^{e_q} \\ h_{i,2}^{e_q} \\ h_{i,3}^{e_q} \end{bmatrix} = \begin{bmatrix} \alpha_i - \alpha_{i-1} \\ \Delta\gamma_i \\ \boldsymbol{\varepsilon}_i^p - \boldsymbol{\varepsilon}_{i-1}^p \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \boldsymbol{0} \end{bmatrix} \tag{23}$$

and for *plastic* loading we obtain,

$$\boldsymbol{H}_i^{e_q} = \begin{bmatrix} h_{i,1}^{e_q} \\ h_{i,2}^{e_q} \\ h_{i,3}^{e_q} \end{bmatrix} = \begin{bmatrix} \alpha_i - \alpha_{i-1} - \Delta\gamma_i \\ \sqrt{\frac{3}{2}\boldsymbol{s}_i : \boldsymbol{s}_i} - \left(\sigma_y^0 + H\alpha_i\right) \\ \boldsymbol{\varepsilon}_i^p - \boldsymbol{\varepsilon}_{i-1}^p - \sqrt{\frac{3}{2}}\Delta\gamma_i \frac{\boldsymbol{s}_i}{||\boldsymbol{s}_i||} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \boldsymbol{0} \end{bmatrix}. \tag{24}$$

The index q represents the quadrature point index corresponding to element index e.

3 Topology optimization formulation

Here we present the basic topology optimization formulation of PolyPlas, including a discussion on the material interpolation, the density filter and projection scheme, and the actual topology optimization statement (objective and constraints). We remark that, in keeping with the PolyPlas philosophy, any of these expressions can be easily changed within the modular structure of the code.

3.1 Material interpolation

Density-based topology optimization appoints the element density in each finite element where $\rho_e = 0$ corresponds to void and $\rho_e = 1$ to solid material. As stated, this problem leads to an integer programming problem, often rendering the problem intractable. The continuous relaxation of the material interpolation, $\boldsymbol{\rho} \in [0, 1]$, enables the use of gradient-based optimization, largely mitigating the anticipated intractably large computational expense. Using such a continuous parameterization, the SIMP method, initially proposed by Bendsøe (1989), is used to interpolate the relevant material properties, including

$$E_e = \left(\epsilon_e + (1-\epsilon_e)\rho_e^p\right)E^{solid} \tag{25}$$

$$H_e = \left(\epsilon_e + (1-\epsilon_e)\rho_e^p\right)H^{solid} \tag{26}$$

$$\sigma_{y,p}^0 = \left(\epsilon_p + (1-\epsilon_p)\rho_e^q\right)\sigma_y^{0,solid} \tag{27}$$

where the exponents p and q represent the elastic and plastic penalization exponents, respectively, and the parameters E^{solid}, H^{solid}, and $\sigma_y^{0,solid}$ represent the elastic modulus, hardening modulus, and initial yield stress of the solid material. The

ersatz parameter is defined for the elastic material properties by, $\epsilon_e = 10^{-8}$ and for the initial yield stress by, $\epsilon_p = 10^{-4}$. Although the material interpolation is scaling down the elastic modulus in a physically consistent manner, the scaling of yield stress will result in high values of accumulated plastic strains in the intermediate density regions. In an attempt to avoid numerical instabilities associated with this phenomenon, a separation of material interpolation schemes is performed by setting the plastic material penalization exponent smaller than the elastic exponents (Maute et al. 1998).

3.2 Density filter and projection schemes

Although continuous parametrization is used for the density variables in the design space $\mathcal{A}$, this alone does not address the issue of well-posedness of the problem nor alleviate the potential checkerboard patterns (Talischi et al. 2012b). Hence, regularity restrictions are introduced in the design space by means of a linear polynomial filtering scheme. In this work, we adopt nodal design variables similar to Guest et al. (2004). A convex combination of the design variables, $\mathbf{z}$ form the filtered field, $\bar{\boldsymbol{\rho}}$ with a filter radius, R. This linear mapping from the design variables to their filtered counterparts may be expressed as

$$\bar{\rho}_i = \frac{\mathrm{z}_j\, \omega(\boldsymbol{x}_i, \boldsymbol{x}_j)}{\sum_j \omega(\boldsymbol{x}_i, \boldsymbol{x}_j)} = P_{ij}\mathrm{z}_j \tag{28}$$

where

$$\omega(\boldsymbol{x}_i, \boldsymbol{x}_j) = \max\left(1 - \frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|}{R},\ 0\right) \tag{29}$$

and the notation $\boldsymbol{x}_i$ represents the coordinates of node i. The filtered design variables are then mapped to each element centroid using the element shape functions, after which the filtered element densities are projected to produce the physical densities as discussed below.

Lastly, in order to partially mitigate the transition region of intermediate densities that remain from the continuous parametrization, the volume preserving Heaviside projection scheme is imposed on the filtered densities (Xu et al. 2010; Wang et al. 2011).

$$\rho_e(\bar{\rho}_e(\mathbf{z})) = \frac{\tanh(\beta\eta) + \tanh(\beta(\bar{\rho}_e - \eta))}{\tanh(\beta\eta) + \tanh(\beta(1-\eta))} \tag{30}$$

The β parameter governs the strength of the projection and the η parameter governs the threshold at which the projection occurs. For example, herein, we set $\eta = 0.5$ and the initial $\beta = \min(1.0,\ 2R/\tau)$ where τ is the largest edge length in the finite element mesh. After the first 100 optimization iterations, we apply a continuation scheme on the projection strength such that it increases in four equal increments for each subsequent 25 optimization iterations until it reaches the maximum projection strength $\beta_{max} = 2R/\tau$ at optimization iteration 200; this is an adaptation of the approach by da Silva et al. (2019).

3.3 Topology optimization statement

The design objective is to maximize the plastic work of a structure undergoing a specified applied displacement subject to a volume constraint. The mathematical optimization statement is:

$$\begin{aligned}
\max_{\mathbf{z}} \quad & f(\boldsymbol{\rho}(\mathbf{z}), \{\boldsymbol{u}_i\}, \{\boldsymbol{v}_i\}) = \int_t \int_\Omega \boldsymbol{\sigma} : \dot{\boldsymbol{\epsilon}}^p \, dV dt \\
\text{s.t.} \quad & g(\boldsymbol{\rho}(\mathbf{z})) = \frac{1}{V_{total}} \int_\Omega \boldsymbol{\rho}\, dV - V_{max} \leq 0 \\
& \boldsymbol{R}_i(\boldsymbol{\rho}(\mathbf{z}), \boldsymbol{u}_i, \boldsymbol{u}_{i-1}, \boldsymbol{v}_i, \boldsymbol{v}_{i-1}) = 0, \quad i = 1, ..., n \\
& \boldsymbol{H}_i(\boldsymbol{\rho}(\mathbf{z}), \boldsymbol{u}_i, \boldsymbol{u}_{i-1}, \boldsymbol{v}_i, \boldsymbol{v}_{i-1}) = 0, \quad i = 1, ..., n.
\end{aligned} \tag{31}$$

As seen from Eq. (31), the optimization problem is also subject to equilibrium constraints, ensuring that the structure satisfy both the global and local equilibrium at each time step (i.e., $\boldsymbol{R}_i = \mathbf{0}$ and $\boldsymbol{H}_i = \mathbf{0}$ for $i = 1, ..., n$). Here we introduce the vector of independent internal local variables, denoted as $\boldsymbol{v}$, where $\boldsymbol{v}_i = \{\alpha_i, \Delta\gamma_i, \boldsymbol{\epsilon}_i^p\}$. Both residuals depend on the set of internal variables, $\boldsymbol{v}_i$ and the nodal displacements, $\boldsymbol{u}_i$. Additionally, they also depend on the state from the previous time step, $\boldsymbol{v}_{i-1}$ and $\boldsymbol{u}_{i-1}$, thereby introducing the path dependency. The vector of design variables, $\mathbf{z}$, is controlled by the optimizer and is used to determine the filtered and projected element densities, $\boldsymbol{\rho}(\bar{\boldsymbol{\rho}}(\mathbf{z}))$. The scalar measure of plastic work is computed by integrating the contraction of the stress, $\boldsymbol{\sigma}$ and the rate of plastic strain, $\dot{\boldsymbol{\epsilon}}^p$ over time and volume using the trapezoidal rule. This is achieved numerically by,

$$f(\boldsymbol{\rho}(\mathbf{z}), \{\boldsymbol{u}_i\}, \{\boldsymbol{v}_i\}) \approx \frac{1}{2} \sum_{i=1}^{n} \sum_{e=1}^{N_{elem}} \sum_{q=1}^{N_{quad}} \left(\boldsymbol{\sigma}_i^{e_q} + \boldsymbol{\sigma}_{i-1}^{e_q}\right) : \left(\boldsymbol{\epsilon}_i^{p,e_q} - \boldsymbol{\epsilon}_{i-1}^{p,e_q}\right) w^{e_q} \tag{32}$$

where e_q notation refers to the quadrature point index q of element index e and w^{e_q} represents the Jacobian of the mapping multiplied by the corresponding quadrature weight. For more information on the e_q notation, see Sect. 4.1. Finally, the volume constraint is bounded by the maximum volume fraction, V_{max} where $\boldsymbol{\rho}$ is the vector of element densities and V_{total} is the total volume of the domain.

4 Path-dependent sensitivity analysis

One of the most significant challenges when considering elastoplasticity in topology optimization is the derivation of the sensitivities due to the path-dependent nature of the problem. Here, we adopt the adjoint method, which begins by constructing a Lagrangian function composed of the original function, f, and the inner products of the local and global residuals, $\boldsymbol{H}_i$ and $\boldsymbol{R}_i$, with their corresponding adjoint vectors, $\boldsymbol{\lambda}_i$ and $\boldsymbol{\mu}_i$,

$$\begin{aligned}\hat{f}(\boldsymbol{\rho},\{\boldsymbol{u}_n\},\{\boldsymbol{v}_n\}) = f(\boldsymbol{\rho},\{\boldsymbol{u}_n\},\{\boldsymbol{v}_n\}) + \sum_{i=1}^{n}\Big(\boldsymbol{\lambda}_i^T\boldsymbol{R}_i(\boldsymbol{\rho},\boldsymbol{u}_i,\boldsymbol{u}_{i-1},\boldsymbol{v}_i,\boldsymbol{v}_{i-1}) \\ + \boldsymbol{\mu}_i^T\boldsymbol{H}_i(\boldsymbol{\rho},\boldsymbol{u}_i,\boldsymbol{u}_{i-1},\boldsymbol{v}_i,\boldsymbol{v}_{i-1})\Big)\end{aligned} \tag{33}$$

We denote Eq. (33) as the augmented function, $\hat{f}$, for which the value and derivative with respect to $\boldsymbol{\rho}$ are equivalent to that of the original objective function,

$$\begin{aligned}&\frac{d\hat{f}}{d\boldsymbol{\rho}} = \frac{df}{d\boldsymbol{\rho}} = \frac{\partial f}{\partial\boldsymbol{\rho}} + \sum_{i=1}^{n}\frac{\partial f}{\partial\boldsymbol{u}_i}\frac{d\boldsymbol{u}_i}{d\boldsymbol{\rho}} + \frac{\partial f}{\partial\boldsymbol{v}_i}\frac{d\boldsymbol{v}_i}{d\boldsymbol{\rho}} + \\ &\sum_{i=1}^{n}\boldsymbol{\lambda}_i^T\left(\frac{\partial\boldsymbol{R}_i}{\partial\boldsymbol{\rho}} + \frac{\partial\boldsymbol{R}_i}{\partial\boldsymbol{u}_i}\frac{d\boldsymbol{u}_i}{d\boldsymbol{\rho}} + \frac{\partial\boldsymbol{R}_i}{\partial\boldsymbol{u}_{i-1}}\frac{d\boldsymbol{u}_{i-1}}{d\boldsymbol{\rho}} + \frac{\partial\boldsymbol{R}_i}{\partial\boldsymbol{v}_i}\frac{d\boldsymbol{v}_i}{d\boldsymbol{\rho}} + \frac{\partial\boldsymbol{R}_i}{\partial\boldsymbol{v}_{i-1}}\frac{d\boldsymbol{v}_{i-1}}{d\boldsymbol{\rho}}\right) \\ &+ \sum_{i=1}^{n}\boldsymbol{\mu}_i^T\left(\frac{\partial\boldsymbol{H}_i}{\partial\boldsymbol{\rho}} + \frac{\partial\boldsymbol{H}_i}{\partial\boldsymbol{u}_i}\frac{d\boldsymbol{u}_i}{d\boldsymbol{\rho}} + \frac{\partial\boldsymbol{H}_i}{\partial\boldsymbol{u}_{i-1}}\frac{d\boldsymbol{u}_{i-1}}{d\boldsymbol{\rho}} + \frac{\partial\boldsymbol{H}_i}{\partial\boldsymbol{v}_i}\frac{d\boldsymbol{v}_i}{d\boldsymbol{\rho}} + \frac{\partial\boldsymbol{H}_i}{\partial\boldsymbol{v}_{i-1}}\frac{d\boldsymbol{v}_{i-1}}{d\boldsymbol{\rho}}\right)\end{aligned} \tag{34}$$

The adjoint vectors are selected to avoid the expensive computation of the state derivatives, $d\boldsymbol{u}_i/\partial\boldsymbol{\rho}$ and $d\boldsymbol{v}_i/\partial\boldsymbol{\rho}$. This is done by rearranging terms,

$$\begin{aligned}\frac{d\hat{f}}{d\boldsymbol{\rho}} =& \frac{\partial f}{\partial\boldsymbol{\rho}} + \frac{d\boldsymbol{u}_n}{d\boldsymbol{\rho}}\left(\frac{\partial f}{\partial\boldsymbol{u}_n} + \boldsymbol{\lambda}_n^T\frac{\partial\boldsymbol{R}_n}{\partial\boldsymbol{u}_n} + \boldsymbol{\mu}_n^T\frac{\partial\boldsymbol{H}_n}{\partial\boldsymbol{u}_n}\right) \\ &+ \frac{d\boldsymbol{v}_n}{d\boldsymbol{\rho}}\left(\frac{\partial f}{\partial\boldsymbol{v}_n} + \boldsymbol{\lambda}_n^T\frac{\partial\boldsymbol{R}_n}{\partial\boldsymbol{v}_n} + \boldsymbol{\mu}_n^T\frac{\partial\boldsymbol{H}_n}{\partial\boldsymbol{v}_n}\right) \\ &+ \sum_{i=1}^{n}\left(\boldsymbol{\lambda}_i^T\frac{\partial\boldsymbol{R}_i}{\partial\boldsymbol{\rho}} + \boldsymbol{\mu}_i^T\frac{\partial\boldsymbol{H}_i}{\partial\boldsymbol{\rho}}\right) \\ &+ \sum_{i=1}^{n-1}\frac{d\boldsymbol{u}_i}{d\boldsymbol{\rho}}\left(\frac{\partial f}{\partial\boldsymbol{u}_i} + \boldsymbol{\lambda}_i^T\frac{\partial\boldsymbol{R}_i}{\partial\boldsymbol{u}_i} + \boldsymbol{\mu}_i^T\frac{\partial\boldsymbol{H}_i}{\partial\boldsymbol{u}_i}\right. \\ &\left. + \boldsymbol{\lambda}_{i+1}^T\frac{\partial\boldsymbol{R}_{i+1}}{\partial\boldsymbol{u}_i} + \boldsymbol{\mu}_{i+1}^T\frac{\partial\boldsymbol{H}_{i+1}}{\partial\boldsymbol{u}_i}\right) \\ &+ \sum_{i=1}^{n-1}\frac{d\boldsymbol{v}_i}{d\boldsymbol{\rho}}\left(\frac{\partial f}{\partial\boldsymbol{v}_i} + \boldsymbol{\lambda}_i^T\frac{\partial\boldsymbol{R}_i}{\partial\boldsymbol{v}_i} + \boldsymbol{\mu}_i^T\frac{\partial\boldsymbol{H}_i}{\partial\boldsymbol{v}_i}\right. \\ &\left. + \boldsymbol{\lambda}_{i+1}^T\frac{\partial\boldsymbol{R}_{i+1}}{\partial\boldsymbol{v}_i} + \boldsymbol{\mu}_{i+1}^T\frac{\partial\boldsymbol{H}_{i+1}}{\partial\boldsymbol{v}_i}\right)\end{aligned} \tag{35}$$

where the adjoint vectors are computed such that they satisfy the following systems of equations, starting with the final time step $i = n$:

$$\text{n}^{th}\text{ time step}: \begin{cases}\boldsymbol{\lambda}_n^T\dfrac{\partial\boldsymbol{R}_n}{\partial\boldsymbol{u}_n} + \boldsymbol{\mu}_n^T\dfrac{\partial\boldsymbol{H}_n}{\partial\boldsymbol{u}_n} = -\dfrac{\partial f}{\partial\boldsymbol{u}_n} \\ \boldsymbol{\lambda}_i^T\dfrac{\partial\boldsymbol{R}_i}{\partial\boldsymbol{v}_i} + \boldsymbol{\mu}_n^T\dfrac{\partial\boldsymbol{H}_n}{\partial\boldsymbol{v}_n} = -\dfrac{\partial f}{\partial\boldsymbol{v}_n}\end{cases} \tag{36}$$

and the remaining time steps, $i = 1, ..., n-1$:

$$\text{i}^{th}\text{ time step}: \begin{cases}\boldsymbol{\lambda}_i^T\dfrac{\partial\boldsymbol{R}_i}{\partial\boldsymbol{u}_i} + \boldsymbol{\mu}_i^T\dfrac{\partial\boldsymbol{H}_i}{\partial\boldsymbol{u}_i} = -\left(\dfrac{\partial f}{\partial\boldsymbol{u}_i} + \boldsymbol{\lambda}_{i+1}^T\dfrac{\partial\boldsymbol{R}_{i+1}}{\partial\boldsymbol{u}_i} + \boldsymbol{\mu}_{i+1}^T\dfrac{\partial\boldsymbol{H}_{i+1}}{\partial\boldsymbol{u}_i}\right) \\ \boldsymbol{\lambda}_i^T\dfrac{\partial\boldsymbol{R}_i}{\partial\boldsymbol{v}_i} + \boldsymbol{\mu}_i^T\dfrac{\partial\boldsymbol{H}_i}{\partial\boldsymbol{v}_i} = -\left(\dfrac{\partial f}{\partial\boldsymbol{v}_i} + \boldsymbol{\lambda}_{i+1}^T\dfrac{\partial\boldsymbol{R}_{i+1}}{\partial\boldsymbol{v}_i} + \boldsymbol{\mu}_{i+1}^T\dfrac{\partial\boldsymbol{H}_{i+1}}{\partial\boldsymbol{v}_i}\right).\end{cases} \tag{37}$$

The solution of these system of equations can be compactly described for $i = 1, .., n$ by denoting the right hand side of both Eqs. (36) and (37) as $\boldsymbol{F}_i^u$ and $\boldsymbol{F}_i^v$ and rearranging the terms,

$$\left(\frac{\partial\boldsymbol{R}_i}{\partial\boldsymbol{u}_i} - \frac{\partial\boldsymbol{R}_i}{\partial\boldsymbol{v}_i}\frac{\partial\boldsymbol{H}_i^{-1}}{\partial\boldsymbol{v}_i}\frac{\partial\boldsymbol{H}_i}{\partial\boldsymbol{u}_i}\right)^T\boldsymbol{\lambda}_i = \boldsymbol{F}_i^u - \frac{\partial\boldsymbol{H}_i^T}{\partial\boldsymbol{u}_i}\frac{\partial\boldsymbol{H}_i^{-T}}{\partial\boldsymbol{v}_i}\boldsymbol{F}_i^v \tag{38}$$

$$\boldsymbol{\mu}_i = \frac{\partial\boldsymbol{H}_i^{-T}}{\partial\boldsymbol{v}_i}\left(\boldsymbol{F}_i^v - \frac{\partial\boldsymbol{R}_i^T}{\partial\boldsymbol{v}}\boldsymbol{\lambda}_i\right) \tag{39}$$

Notice that the construction of the system of equations varies depending on whether the analysis is at the final time step or not. This implementation of this procedure for computing the adjoint vectors is further described in Sect. 5.4.1.

Finally, after determining the adjoint vectors, we arrive at the reduced form for the function sensitivity,

$$\frac{d\hat{f}}{d\boldsymbol{\rho}} = \frac{\partial f}{\partial\boldsymbol{\rho}} + \sum_{i=1}^{n}\left(\boldsymbol{\lambda}_i^T\frac{\partial\boldsymbol{R}_i}{\partial\boldsymbol{\rho}} + \boldsymbol{\mu}_i^T\frac{\partial\boldsymbol{H}_i}{\partial\boldsymbol{\rho}}\right) \tag{40}$$

This leads to the following explicit partial derivatives that must be obtained.

$$\text{For } f: \begin{cases}\dfrac{\partial f}{\partial\boldsymbol{\rho}} \\ \dfrac{\partial f}{\partial\boldsymbol{u}_i} \\ \dfrac{\partial f}{\partial\boldsymbol{v}_i}\end{cases}, \quad \text{for } \boldsymbol{R}_i: \begin{cases}\dfrac{\partial\boldsymbol{R}_i}{\partial\boldsymbol{\rho}} \\ \dfrac{\partial\boldsymbol{R}_i}{\partial\boldsymbol{u}_i} \\ \dfrac{\partial\boldsymbol{R}_i}{\partial\boldsymbol{u}_{i-1}} \\ \dfrac{\partial\boldsymbol{R}_i}{\partial\boldsymbol{v}_i} \\ \dfrac{\partial\boldsymbol{R}_i}{\partial\boldsymbol{v}_{i-1}}\end{cases}, \quad \text{for } \boldsymbol{H}_i: \begin{cases}\dfrac{\partial\boldsymbol{H}_i}{\partial\boldsymbol{\rho}} \\ \dfrac{\partial\boldsymbol{H}_i}{\partial\boldsymbol{u}_i} \\ \dfrac{\partial\boldsymbol{H}_i}{\partial\boldsymbol{u}_{i-1}} \\ \dfrac{\partial\boldsymbol{H}_i}{\partial\boldsymbol{v}_i} \\ \dfrac{\partial\boldsymbol{H}_i}{\partial\boldsymbol{v}_{i-1}}\end{cases} \tag{41}$$

Note that for the evaluation of the sensitivities with respect to the design variables $\mathbf{z}$, the chain rule must be applied to the sensitivities computed with respect to the filtered and projected design variables as shown below,

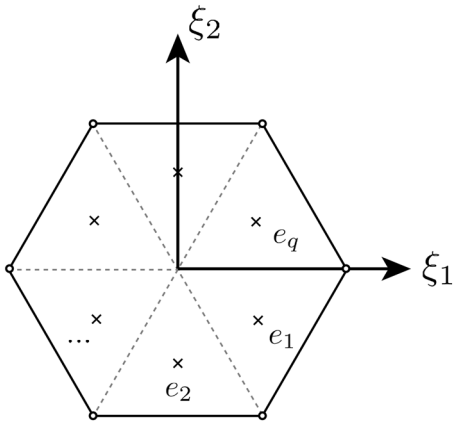


Fig. 2 Illustration of the e_q definition and the quadrature triangulation for the polygonal element

$$\frac{\partial f}{\partial \mathbf{z}} = \frac{\partial f}{\partial \boldsymbol{\rho}} \frac{\partial \boldsymbol{\rho}}{\partial \bar{\boldsymbol{\rho}}} \frac{\partial \bar{\boldsymbol{\rho}}}{\partial \mathbf{z}} = \frac{\partial f}{\partial \boldsymbol{\rho}} \frac{\partial \boldsymbol{\rho}}{\partial \bar{\boldsymbol{\rho}}} \mathbf{P} \tag{42}$$

4.1 Local residual derivatives

Here we start by introducing the implemented polygonal element integration scheme. General irregular polygons do not admit a standard quadrature rule. Therefore, here we adopt a triangulation scheme with a single integration point per triangle for all n-gons such that a quadrilateral element will have four quadrature points. The quadrature triangulation scheme is briefly illustrated in Fig. 2 along with the e_q notation used in the remainder of the article, i.e., e_q where the quadrature point index $q = 1, ..., N_{quad}$ for each element index $e = 1, ..., N_{elem}$ where N_{quad} varies depending on the polygonal element type (e.g., triangles (3 vertices), quadrilaterals (4 vertices), pentagons (5 vertices), etc.).

The partial derivatives depend on the selection of the set of independent internal variables, $\boldsymbol{v}_i$ as further discussed by Alberdi et al. (2018). In this work, the set of internal local variables are defined by $\boldsymbol{v}_i^{e_q} = \{\alpha_i^{e_q}, \Delta\gamma_i^{e_q}, \boldsymbol{\varepsilon}_i^{p,e_q}\}$, therefore requiring the derivatives,

$$\frac{\partial f^{e_q}}{\partial \boldsymbol{v}_i^{e_q}} = \left[\frac{\partial f^{e_q}}{\partial \alpha_i^{e_q}}, \frac{\partial f^{e_q}}{\partial \Delta\gamma_i^{e_q}}, \frac{\partial f^{e_q}}{\partial \boldsymbol{\varepsilon}_i^{p,e_q}} \right]^T \tag{43}$$

at each quadrature point. We require the partial derivatives with respect to the internal variables at both the current time step,

$$\frac{\partial \boldsymbol{H}_i^{e_q}}{\partial \boldsymbol{v}_i^{e_q}} = \left[\frac{\partial \boldsymbol{H}_i^{e_q}}{\partial \alpha_i^{e_q}}, \frac{\partial \boldsymbol{H}_i^{e_q}}{\partial \Delta\gamma_i^{e_q}}, \frac{\partial \boldsymbol{H}_i^{e_q}}{\partial \boldsymbol{\varepsilon}_i^{p,e_q}} \right]^T \tag{44}$$

and at the previous time step,

$$\frac{\partial \boldsymbol{H}_i^{e_q}}{\partial \boldsymbol{v}_{i-1}^{e_q}} = \left[\frac{\partial \boldsymbol{H}_i^{e_q}}{\partial \alpha_{i-1}^{e_q}}, \frac{\partial \boldsymbol{H}_i^{e_q}}{\partial \Delta\gamma_{i-1}^{e_q}}, \frac{\partial \boldsymbol{H}_i^{e_q}}{\partial \boldsymbol{\varepsilon}_{i-1}^{p,e_q}} \right]^T. \tag{45}$$

For an *elastic* step, the partials of the local residual with respect to the internal variables are shown below:

$$\begin{aligned} &\frac{\partial \boldsymbol{H}_i^{e_q}}{\partial \alpha_i^{e_q}} = \begin{bmatrix} 1 \\ 0 \\ \mathbf{0} \end{bmatrix}, \frac{\partial \boldsymbol{H}_i^{e_q}}{\partial \alpha_{i-1}^{e_q}} = \begin{bmatrix} -1 \\ 0 \\ \mathbf{0} \end{bmatrix}, \frac{\partial \boldsymbol{H}_i^{e_q}}{\partial \Delta\gamma_i^{e_q}} = \begin{bmatrix} 0 \\ 1 \\ \mathbf{0} \end{bmatrix}, \\ &\frac{\partial \boldsymbol{H}_i^{e_q}}{\partial \Delta\gamma_{i-1}^{e_q}} = \begin{bmatrix} 0 \\ 0 \\ \mathbf{0} \end{bmatrix}, \frac{\partial \boldsymbol{H}_i^{e_q}}{\partial \boldsymbol{\varepsilon}_i^{p,e_q}} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbb{I}^s \end{bmatrix}, \frac{\partial \boldsymbol{H}_i^{e_q}}{\partial \boldsymbol{\varepsilon}_{i-1}^{p,e_q}} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ -\mathbb{I}^s \end{bmatrix} \end{aligned} \tag{46}$$

The partial derivatives of the local residual with respect to the density and nodal displacement variables in this case are zero:

$$\frac{\partial \boldsymbol{H}_i^{e_q}}{\partial \rho_e} = \begin{bmatrix} 0 \\ 0 \\ \mathbf{0} \end{bmatrix}, \frac{\partial \boldsymbol{H}_i^{e_q}}{\partial \boldsymbol{u}_i^{e_q}} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \frac{\partial \boldsymbol{H}_i^{e_q}}{\partial \boldsymbol{u}_{i-1}^{e_q}} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix} \tag{47}$$

For a step resulting in *plastic* flow, we arrive at the following derivatives,

$$\begin{aligned} &\frac{\partial \boldsymbol{H}_i^{e_q}}{\partial \alpha_i^{e_q}} = \begin{bmatrix} 1 \\ -H \\ \mathbf{0} \end{bmatrix}, \frac{\partial \boldsymbol{H}_i^{e_q}}{\partial \alpha_{i-1}^{e_q}} = \begin{bmatrix} -1 \\ 0 \\ \mathbf{0} \end{bmatrix}, \frac{\partial \boldsymbol{H}_i^{e_q}}{\partial \Delta\gamma_i^{e_q}} = \begin{bmatrix} -1 \\ 0 \\ -\sqrt{\frac{3}{2}} \frac{\boldsymbol{s}_i^{e_q}}{||\boldsymbol{s}_i^{e_q}||} \end{bmatrix}, \\ &\frac{\partial \boldsymbol{H}_i^{e_q}}{\partial \Delta\gamma_{i-1}^{e_q}} = \begin{bmatrix} 0 \\ 0 \\ \mathbf{0} \end{bmatrix}, \frac{\partial \boldsymbol{H}_i^{e_q}}{\partial \boldsymbol{\varepsilon}_i^{p,e_q}} = \begin{bmatrix} \mathbf{0} \\ -\sqrt{6}G \frac{\boldsymbol{s}_i^{e_q}}{||\boldsymbol{s}_i^{e_q}||} \\ \mathbb{I}^s + \boldsymbol{A}_i^{e_q} \end{bmatrix}, \frac{\partial \boldsymbol{H}_i^{e_q}}{\partial \boldsymbol{\varepsilon}_{i-1}^{p,e_q}} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ -\mathbb{I}^s \end{bmatrix}, \end{aligned} \tag{48}$$

where we define the term $\boldsymbol{A}_i^{e_q}$ by

$$\boldsymbol{A}_i^{e_q} = \frac{\sqrt{6}G\Delta\gamma}{||\boldsymbol{s}_i^{e_q}||} \left(\mathbb{P}^{dev} - \frac{\boldsymbol{s}_i^{e_q}}{||\boldsymbol{s}_i^{e_q}||} \otimes \frac{\boldsymbol{s}_i^{e_q}}{||\boldsymbol{s}_i^{e_q}||} \right). \tag{49}$$

The corresponding local residual derivatives with respect to the density and nodal displacement variables are determined as,

$$\frac{\partial \boldsymbol{H}_i^{e_q}}{\partial \rho_e} = \begin{bmatrix} 0 \\ C_i \\ \mathbf{0} \end{bmatrix}, \frac{\partial \boldsymbol{H}_i^{e_q}}{\partial \boldsymbol{u}_i^{e_q}} = \begin{bmatrix} \mathbf{0} \\ \sqrt{6}G \frac{\boldsymbol{s}_i^{e_q}}{||\boldsymbol{s}_i^{e_q}||} : \boldsymbol{B}^u \\ -\boldsymbol{A}_i^{e_q} : \boldsymbol{B}^u \end{bmatrix}, \frac{\partial \boldsymbol{H}_i^{e_q}}{\partial \boldsymbol{u}_{i-1}^{e_q}} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix} \tag{50}$$

with the value C_i defined as

$$C_i = \frac{\partial G}{\partial \rho_e}\frac{\sqrt{6}}{||\boldsymbol{s}_i^{e_q}||}\boldsymbol{s}_i^{e_q} : (\boldsymbol{\varepsilon}_i^{e_q} - \boldsymbol{\varepsilon}_i^{p,e_q}) - \left(\frac{\partial \sigma_y}{\partial \rho_e} + \frac{\partial H}{\partial \rho_e}\alpha_{e_q}^i\right). \quad (51)$$

4.2 Global residual derivatives

The global residual equations are constructed from the element level by,

$$\boldsymbol{R}_i^e = \int_{\Omega_e} \boldsymbol{B}^{u^T} : \boldsymbol{s}_i \, dV + p_i \int_{\Omega_e} \boldsymbol{B}_{div}^{u^T} \, dV \quad (52)$$

where external traction loads and body forces have been neglected. The expressions for the partial derivatives of the global residual are shown below. Note that the global residual equation is only dependent on the state variables at the current time step and thus the partial derivatives with respect to the state variables at the previous time step, $\partial \boldsymbol{R}_i/\partial \boldsymbol{u}_{i-1} = \partial \boldsymbol{R}_i/\partial \boldsymbol{v}_{i-1} = \boldsymbol{0}$. In addition, the global residual is only dependent on the plastic strain component of the local state variables leading to $\partial \boldsymbol{R}_i^e/\partial \alpha_i = \partial \boldsymbol{R}_i^e/\partial \Delta\gamma_i = \boldsymbol{0}$. The nonzero partial derivatives are shown in the following expressions,

$$\frac{\partial \boldsymbol{R}_i^e}{\partial \rho_e} = \int_{\Omega_e} 2\frac{\partial G}{\partial \rho_e}\boldsymbol{B}^{u^T} : \mathbb{P}^{dev} : (\boldsymbol{\varepsilon}_i - \boldsymbol{\varepsilon}_i^p) dV + \frac{\partial \kappa}{\partial \rho_e}\Theta_{i+1}\int_{\Omega_e} \boldsymbol{B}_{div}^{u^T} dV \quad (53)$$

$$\frac{\partial \boldsymbol{R}_i^e}{\partial \boldsymbol{u}_i} = \int_{\Omega_e} 2G\boldsymbol{B}^{u^T} : \mathbb{P}^{dev} : \boldsymbol{B}^u \, dV + \frac{\kappa}{V_e}\left(\int_{\Omega_e} \boldsymbol{B}_{div}^{u^T} dV\right)\left(\int_{\Omega_e} \boldsymbol{B}_{div}^{u^T} dV\right) \quad (54)$$

$$\frac{\partial \boldsymbol{R}_i^e}{\partial \boldsymbol{\varepsilon}_i^p} = -\int_{\Omega_e} 2G\boldsymbol{B}^{u^T} : \mathbb{P}^{dev} \, dV \quad (55)$$

4.3 Plastic work derivatives

For the partial derivatives of the plastic work objective function we start by separating the terms affiliated with the final time step, n and all other time steps, $i = 1, ..., n-1$:

$$f(\boldsymbol{\rho}, \{\boldsymbol{u}_n\}, \{\boldsymbol{v}_n\}) = \frac{1}{2}\int_\Omega \boldsymbol{\sigma}_n : (\boldsymbol{\varepsilon}_n^p - \boldsymbol{\varepsilon}_{n-1}^p) dV + \frac{1}{2}\sum_{i=1}^{n-1}\int_\Omega \boldsymbol{\sigma}_i : (\boldsymbol{\varepsilon}_{i+1}^p - \boldsymbol{\varepsilon}_{i-1}^p) dV \quad (56)$$

The partial derivative of the plastic work with respect to the element densities is determined by,

$$\frac{\partial f}{\partial \rho_e} = \frac{1}{2}\int_\Omega \frac{\partial \boldsymbol{s}_n}{\partial \rho_e} : (\boldsymbol{\varepsilon}_n^p - \boldsymbol{\varepsilon}_{n-1}^p) dV + \frac{1}{2}\sum_{i=1}^{n-1}\int_\Omega \frac{\partial \boldsymbol{s}_i}{\partial \rho_e} : (\boldsymbol{\varepsilon}_{i+1}^p - \boldsymbol{\varepsilon}_{i-1}^p) dV \quad (57)$$

where

$$\frac{\partial \boldsymbol{s}_i}{\partial \rho_e} = 2\frac{\partial G}{\partial \rho_e}\mathbb{P}^{dev} : (\boldsymbol{\varepsilon}_i - \boldsymbol{\varepsilon}_i^p). \quad (58)$$

Note that the pressure component of the stress is not included in the partial derivatives due to the assumption that the material is plastically incompressible where $p\boldsymbol{I} : \boldsymbol{\varepsilon}^p = 0$. The remaining partials of the plastic work, $\partial f/\partial \boldsymbol{v}_i$ and $\partial f/\partial \boldsymbol{u}_i$ required in Eqs. (36) and (37) are separated into expressions for the final time step, n, and all previous time steps, $i = 1, ..., n-1$. For all time steps the plastic work does not explicitly depend on the equivalent plastic strain, α_i, or the plastic multiplier, $\Delta\gamma_i$, leading to,

$$\frac{\partial f}{\partial \alpha_i} = \frac{\partial f}{\partial \Delta\gamma_i} = 0. \quad (59)$$

The remaining partial derivatives of the plastic work are defined for the final time step, $i = n$, by,

$$\frac{\partial f}{\partial \boldsymbol{\varepsilon}_n^p} = \frac{1}{2}\int_\Omega -2G\mathbb{P}^{dev} : (\boldsymbol{\varepsilon}_n^p - \boldsymbol{\varepsilon}_{n-1}^p) dV + \frac{1}{2}\int_\Omega (\boldsymbol{s}_n + \boldsymbol{s}_{n-1}) dV \quad (60)$$

$$\frac{\partial f}{\partial \boldsymbol{u}_n} = \frac{1}{2}\int_{\Omega_e} 2G\mathbb{P}^{dev} : (\boldsymbol{\varepsilon}_n^p - \boldsymbol{\varepsilon}_{n-1}^p) : \boldsymbol{B}^u dV \quad (61)$$

and for all previous time steps, $i = 1, .., n-1$, by

$$\frac{\partial f}{\partial \boldsymbol{\varepsilon}_i^p} = \frac{1}{2}\int_\Omega -2G\mathbb{P}^{dev} : (\boldsymbol{\varepsilon}_{i+1}^p - \boldsymbol{\varepsilon}_{i-1}^p) dV - \frac{1}{2}\int_\Omega (\boldsymbol{s}_{i+1} - \boldsymbol{s}_{i-1}) dV \quad (62)$$

$$\frac{\partial f}{\partial \boldsymbol{u}_i} = \frac{1}{2}\int_{\Omega_e} 2G\mathbb{P}^{dev} : (\boldsymbol{\varepsilon}_{i+1}^p - \boldsymbol{\varepsilon}_{i-1}^p) : \boldsymbol{B}^u dV. \quad (63)$$

5 PolyPlas organization

Here we explain the PolyPlas organization with special attention to its main modules. We discuss the PolyPlasScript.py, the polygonal element implementation details, the elastoplastic forward analysis, and most importantly the details involved for the implementation of the path-dependent sensitivity analysis. Whenever appropriate, we draw an analogy with the PolyTech family of educational software, including PolyTop (Talischi et al. (2012b)).

5.1 The PolyPlasScript.py

As in PolyTop, we use a script file PolyPlasScript.py to setup all the information necessary to create an instance of the PolyPlas class and subsequently run the

Table 1 Main Python files used in `PolyPlas`

File name	Description
`PolyPlasScript`	Script containing all the input information required to run a `PolyPlas` example
`PolyPlas`	Main program that conducts the nonlinear forward analysis, sensitivity analysis and runs the topology optimization problem
`PolyPlasUtilities`	Produces `PolyPlasPrecomputedData`, `currentPlasticState`, and `MMA_State` objects
`PolyMesher`	Generates polygonal finite element mesh for selected `BoundaryValueProblem`
`PolyGeometryAndBCs`	Contains `BoundaryValueProblem` subclasses to define various domains and their associated boundary conditions
`PolyFilter`	Implementation of the polynomial filter and the identity filter (no filter)
`PolyInterpolation`	Material interpolation class construction for RAMP and SIMP interpolation
`PolyProjection`	Implementation of tanh Heaviside projection and identity projection

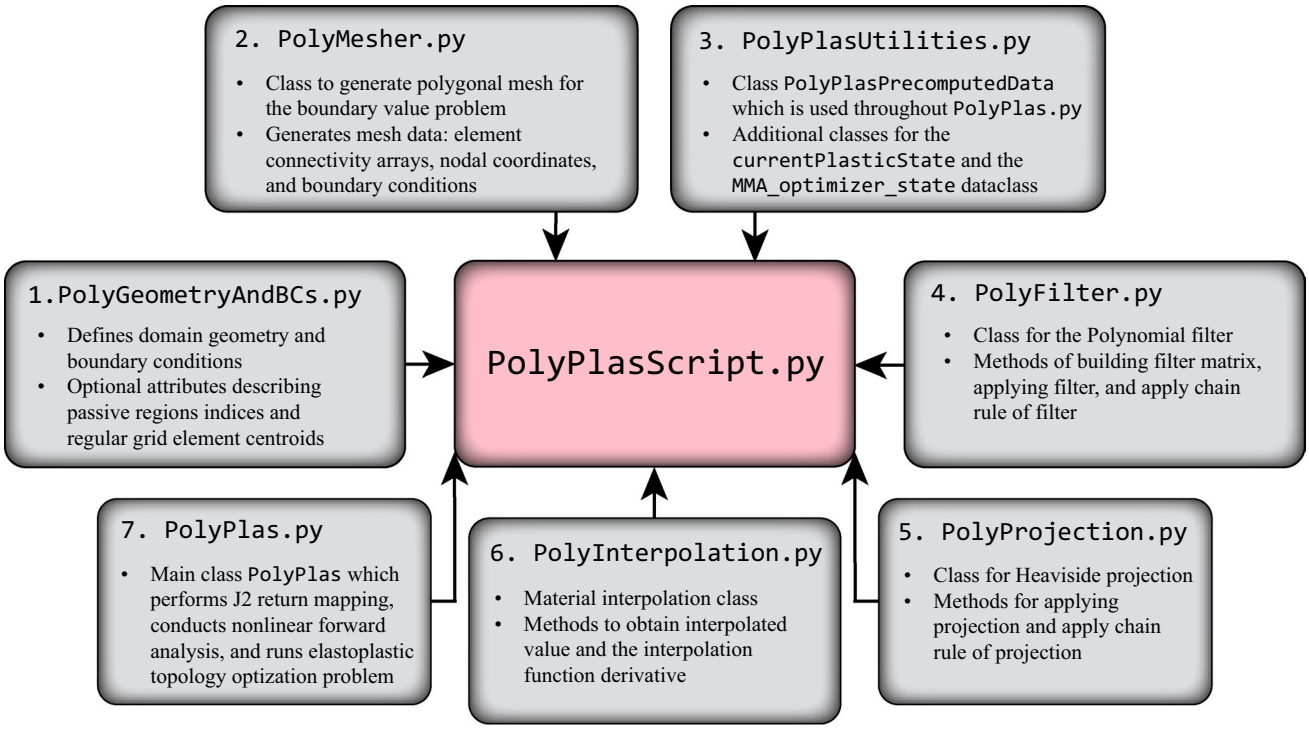


Fig. 3 Illustration of how files are integrated within `PolyPlasScript.py` in order to run the elastoplastic topology optimization problem

topology optimization problem. All of the files needed to run the topology optimization problem are gathered with their file names and brief descriptions shown in Table 1. In addition, Fig. 3 describes each component in the sequence they are utilized within `PolyPlasScript.py`. Note that for the sake of clarity and good programming practice, all file and variable names are intended to be self-explanatory.

The `PolyPlasScript.py` begins with a list of all the available boundary value problems in `PolyGeometryandBCs.py`. The boundary value problems are described by creating subclasses of the base class `BoundaryValueProblem` which contains the abstract methods required to implement a new boundary value problem. For each `BoundaryValueProblem`, the domain geometry, boundary conditions, and applied displacement are specified. Note that the program is restricted to a single applied displacement region but the displacement direction may occur in the x, y, or x and y direction. The attribute `applied_displacement_magnitude` of the boundary value problem is intended to allow for more user flexibility within the `PolyPlas.py` script. A `PolyMesher` object is then created in the `PolyPlasScript.py` using the selected boundary value problem, number of elements, maximum polymesher iterations and a `boolean` variable for the use

of a regular grid; for more details on the polygonal mesh generation refer to Talischi et al. (2012a).

A material property Dict is defined within the script taking entries of the elastoplastic material properties by the elastic modulus, Poisson's ratio, hardening modulus, and the initial yield stress, respectively. The PolyMesher object along with the material parameters Dict is used to obtain the precomputed_data object of the class PolyPlasPrecomputedData from PolyPlasUtilities.py which contains variables that are used repeatedly and stay constant throughout PolyPlas , e.g., the hardening modulus, the number of elements, the deviatoric projection tensor, and so on. To view the complete list of variables contained in the PolyPlasPrecomputedData, see lines 15 – 65 in PolyPlasUtilities.py.

Next, a design variable filter object is created, from the PolynomialFilter class in PolyFilter.py, which allows for a relative filter radius to be used via a boolean input. If the variable is True, the filter radius is a multiple of the largest element edge length, $R_{used} = R * \tau$ and if False, $R_{used} = R$ per the standard procedure. Here one can also specify the filter type, polynomial or identity (no filter), and may specify symmetry restrictions. Following this, the projection function object is defined by the polyprojection.get_projection_function() method which takes the projection type (tanh or identity), initial projection strength β, and projection threshold η as input. Afterward, the material interpolation schemes for the elastic and plastic material parameters are defined, where each is an instance of an interpolation class from the PolyInterpolation.py file. The material interpolation is created by specifying the type of material interpolation (SIMP or RAMP), ersatz parameter, and penalization parameter. Note that the penalization parameter is updated during optimization via a continuation strategy present in the get_function_values_and_gradients() method of PolyPlas.py. Such a continuation strategy greatly improves numerical stability in the elastoplastic optimization problem. Finally, the PolyPlas object is created in the script file using the aforementioned variables as shown on lines 90 – 100 of PolyPlasScript.py. To see a complete list of all input variables and their corresponding types, see the summary in Table 2.

Before running the topology optimization problem, the MMA_state object is created as an instance of the MMA_Optimizer_state class in lines 122 – 185 in the PolyPlasUtilities.py file. This object stores all relevant information needed in updating the design variables using the well-known Method of Moving Asymptotes (MMA) optimizer (Svanberg 1987). Finally, the run_top_opt_problem() method from PolyPlas.py is called in the script followed by methods plot_density_field() and plot_optimization_history(). The implementation details involved in solving the topology optimization problem in PolyPlas are described in detail in the following sections.

5.2 Polygonal element implementation details

The generation of the polygonal finite element information is conducted in PolyPlasUtilities.py which contains functions pertaining to the triangulation of the polygons in get_polygonal_triangulation(), gathering of the quadrature point information in get_polygonal_element_quadrature(), and functions on obtaining the shape function value and gradient information, as shown in lines 239 – 360. The procedure for obtaining the polygonal finite element information follows the standard approach as outlined in Talischi et al. (2012a).

An important feature to notice when considering polygonal finite elements in the PolyPlas program is the looping over the polygonal element types and the creation of dictionaries with keys pertaining to the number of vertices in each polygonal element type. This is necessary for the

Table 2 List of inputs in PolyPlas class

Variable name	
design_variable_filter	Instance of FilterBase class
elastic_material_interpolation	Instance of MaterialInterpolationFuction class
plastic_material_interpolation	Instance of MaterialInterpolationFuction class
projection	Instance of ProjectionBase class
maximum_projection_strength	Maximum projection strength
precomputed_data	Instance of PolyPlasPrecomputedData class
volume_fraction_upper_bound	V_{max} from Eq. (31), type: float
number_of_time_steps	Number of pseudo-time steps for forward analysis, type: int
directory_name	Directory name for storing output data, type: string

computations due to the dimensions affiliated with the element degrees of freedom varying per polygonal element type. An example of this is demonstrated in the code snippet below in which the plastic work objective function is updated within the method `run_forward_analysis()` in `PolyPlas.py` lines 442 – 458 at the current time step.

```
# Computing the update of the plastic work objective function value
for number_of_vertices in unique_number_of_vertices:
    deviatoric_stress_tensor_sum = \
        self.state.current_deviatoric_stress_tensors[number_of_vertices] + \
        self.state.previous_deviatoric_stress_tensors[number_of_vertices]

    plastic_strain_difference = \
        self.state.current_plastic_strain_tensors[number_of_vertices] - \
        self.state.previous_plastic_strain_tensors[number_of_vertices]

    JxW = self.precomputed_data.jacobian_determinant_x_quadrature_weights \
        [number_of_vertices]

    objective_function_value += oe.contract("...,eqij,eqij,eq->",
                                            -0.5,
                                            deviatoric_stress_tensor_sum,
                                            plastic_strain_difference,
                                            JxW)
```

Here each entry in the array `unique_number_of_vertices` corresponds to the number of vertices in each of the unique element types. The objective function is updated numerically following Eq. (32). The relevant information from the `state` variable pertaining to the current time step is used in computing the `deviatoric_stress_tensor_value` and the `plastic_strain_difference`. These variables are then used in the function `oe.contract` which performs the tensor contraction operation using conventional Einstein notation. Further information on how to use the `oe.contract` function can be found in the tutorial outlined in Appendix C.

5.2.1 Standard bilinear quadrilateral element implementation

If this user instead desires to use standard bilinear quadrilateral (Q4) elements in their analysis, they can do so by utilizing the functions we have implemented in the `PolyPlasUtilities.py` file. These functions include the standard approach in obtaining the quadrature point information and the respective shape function values and gradients as can be seen in `PolyPlasUtilities.py` lines 364 – 436. To use these functions the user simply has to call the function `get_shape_function_table_and_quadrature_weights_quad_element_type()` as shown in line 673. Additionally, the user has to ensure that the mesh is composed of a structured grid by setting the variable, `use_regular_grid_if_implemented=True`, when initializing the `PolyMesher` object in the `PolyPlasScript.py`. Note that the regular grid can only be imposed on rectangular-type domains; this highlights the benefits of using polygonal finite elements which can mesh arbitrary, curved domains.

5.3 Elastoplastic forward analysis

The elastoplastic forward analysis needs to be performed at every optimization iteration which is why topology optimization with elastoplasticity can be a very computationally expensive task. As mentioned previously, the equations are solved incrementally using a fully implicit pseudo-time integration scheme and, within each increment, the nodal displacements are updated using a global Newton Raphson iteration. Both the time integration and Newton iteration loops are located in the `run_forward_analysis()` method within `PolyPlas.py` on lines 372 – 414. The update of the global and local state variables is performed using the method, `update_state()`. To do this the method `update_state()` calls other `PolyPlas` methods `get_element_quantities()` and

get_J2_update(). The method get_element_quantities() is used to compute the element level contributions to the global residual vector and the tangent stiffness matrix. The return mapping procedure outlined in Appendix A takes place in get_J2_update() and the supplied total_strain_tensors are computed under the assumption of plane strain. Note that all stress and strain tensors in PolyPlas are stored as 3x3 matrices, enabling easier extension to three-dimensional problems. The element residual and element Jacobian matrices are then computed in a loop over each element type, beginning with the Jacobian matrices as shown in Eq. (14),

The element residual and Jacobian information is used to assemble the global system of equations and to determine the update of the nodal displacement vector. The global Newton Raphson iteration continues until the relative residual norm is less than the specified tolerance, $\|\boldsymbol{R}^m_{rel}\| \equiv \|\boldsymbol{R}^m\|/\|\boldsymbol{R}^0\| < \text{tol}_{\text{NR}}$, as shown in the method run_forward_analysis(). Within the Newton Raphson scheme, a backtracking line search procedure is conducted for improved convergence stability in the nonlinear forward analysis. Once the global equilibrium equations have converged for the given time step, the state

```
element_jacobian_matrices = oe.contract("eqmij,eqijkl,eqnkl->emn",
                                        virtual_displacement_symmetric_gradients,
                                        constitutive_tensors[number_of_vertices],
                                        virtual_displacement_symmetric_gradients,
                                        jacobian_determinant_x_quadrature_weights)
```

```
element_jacobian_matrices += oe.contract("...,e,em,en->emn",
                                         bulk_modulus,
                                         elastic_interpolation_per_element /
                                           element_volumes_per_elem_type,
                                         virtual_displacement_volume_weighted_divergence,
                                         virtual_displacement_volume_weighted_divergence)
```

Similarly the element residual vectors are computed following Eq. (11) by,

```
element_residual_vectors = oe.contract("eqmij,eqij,eq->em",
                                       virtual_displacement_symmetric_gradients,
                                       current_deviatoric_stress_tensors,
                                       jacobian_determinant_x_quadrature_weights)
```

```
element_residual_vectors += oe.contract("e,em->em",
                                        pressure,
                                        virtual_displacement_volume_weighted_divergence)
```

object is saved, storing all relevant physical data for that time index. The `state` object is an instance of the class `currentPlasticState`, to view all the variables stored within `state` see lines in the `PolyPlasUtilities.py` file. The `state` was initialized and stored within the `PolyPlas` object by the `get_initial_plastic_state()` function from `PolyPlasUtilities.py` lines 898 – 932. Storing the `state` data at each time step is crucial in computing the path-dependent sensitivity analysis which is further elaborated upon in Sect. 5.4.1. Subsequently after saving the `state`, the method `transfer_state()` of the class `currentPlasticState` is called to store the current state variable values $(\cdot)_i$ in the entries named with "previous" $(\cdot)_{i-1}$, in preparation for the next time increment. For more details on the nonlinear finite element routine, including the global Newton iteration with line search, see Algorithm 1.

Algorithm 1 Global equilibrium iteration with inexact line search

Initialize the objective function value, $\hat{f} = 0$
for $i = 1$ to n **do**
  Set current applied displacement for time step index i
  Initialize newton iteration index $m = 1$
  Initialize the nodal displacement vector $\mathbf{u}_0 = \mathbf{0}$
  while $\|\boldsymbol{R}^m_{rel}\| > \text{tol}_{\text{NR}}$ **do**
    Call `update_state` to obtain nodal displacement update $\Delta\mathbf{u}^{(m)}$ and current residual norm, $\|\boldsymbol{R}^m\|$
    if $m = 1$ **then**
      Set $\|\boldsymbol{R}^0\| = \max(1.0, \|\boldsymbol{R}^m\|)$
    end if
    if $m > 1$ **then**
      Set line search step size $\alpha^1 = 0.8$
      for $k = 1$ to K **do**
        Update $\mathbf{u}_i^{(m)} = \mathbf{u}_i^{(m-1)} + \alpha^k \Delta\mathbf{u}^{(m)}$ and compute residual norm $\|\boldsymbol{R}^m\|$
        if $\|\boldsymbol{R}^m\| < \|\boldsymbol{R}^{(m-1)}\|$ **then**
          end line search
        end if
        Set $\alpha^{k+1} = \alpha^k \cdot \alpha^k$
      end for
    end if
    Compute relative residual norm $\|\boldsymbol{R}^m_{rel}\| = \|\boldsymbol{R}^m\| / \|\boldsymbol{R}^0\|$
    Increase newton iteration index, $m = m + 1$
  end while
  Update the objective function via Equation (32)
  Save the converged `state` for current time step index, i
  Transfer `state` data from current to previous for next time step, $i + 1$
end for

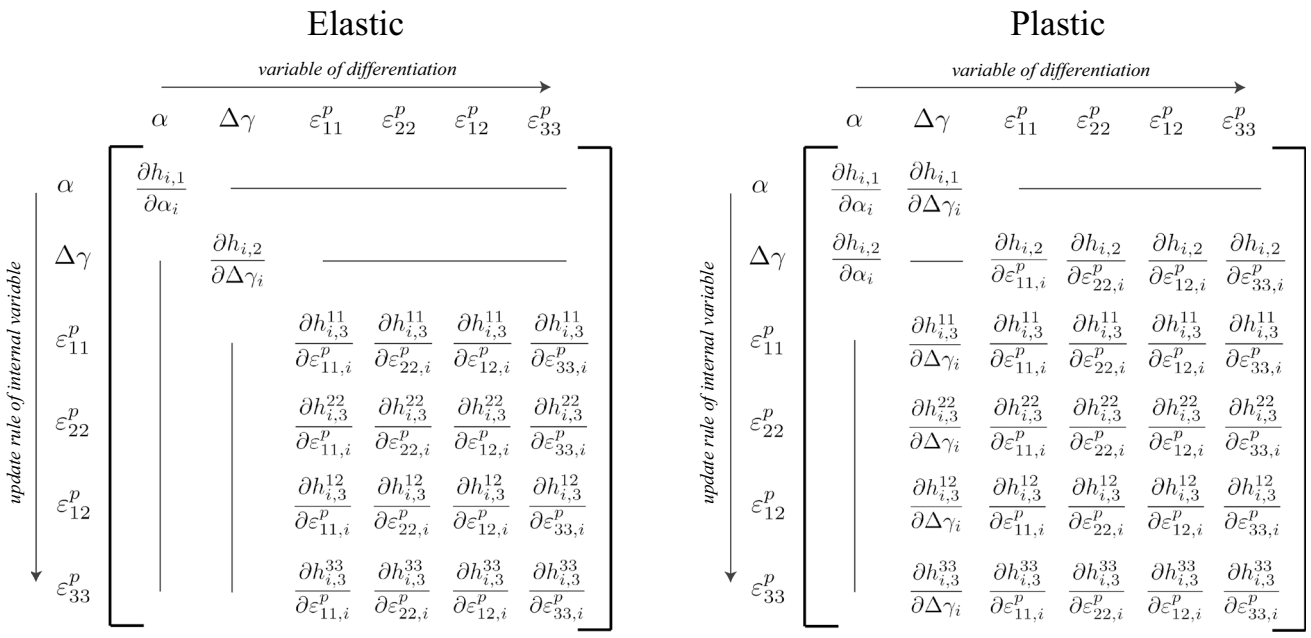


Fig. 4 The local residual derivative with respect to the local internal variables for a single quadrature point in the *elastic* region (left) and in the *plastic* region (right)

5.4 Sensitivity analysis implementation

This section places strong emphasis on the algorithm used in conducting the path-dependent sensitivity analysis (beginning with the final time step and working backward) along with a thorough explanation on the procedure involved in carrying out the partial derivatives of the local residual vector. These details are crucial for the implementation of topology optimization considering path-dependent physics.

5.4.1 Path-dependent procedure

The path-dependent sensitivity analysis is computed by calling the `compute_objective_function_sensitivity()` method in `PolyPlas`. Here the procedure begins by loading the `state` at the final time step n by:

```
self.state = self.states[total_number_of_time_steps - 1].
```

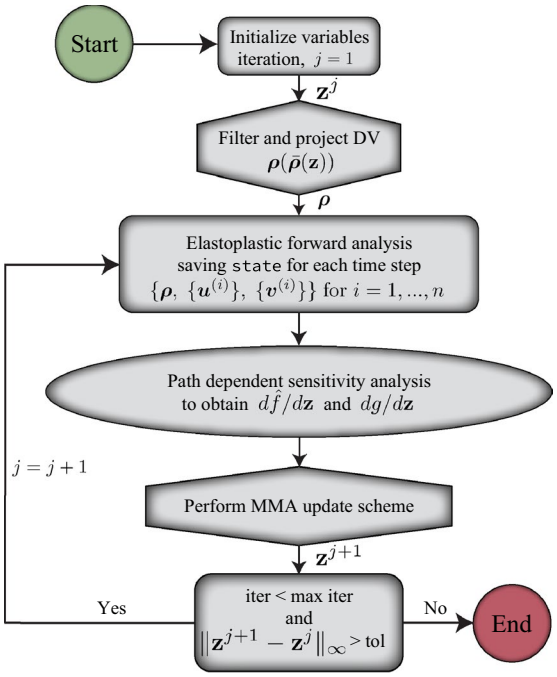


Fig. 5 Flowchart of topology optimization procedure

Table 3 Material parameters of Aluminum 2024-T351

E [MPa]	ν	σ_{y0} [MPa]	H [MPa]
74633.0	0.3	344.0	2000.0

This returns the `state` from the final time step $i = n$, which is used to assemble and solve the system of equations in (38) and (39). The procedure begins at the final time step since the adjoint vectors at the current time step, $\boldsymbol{\lambda}_i$ and $\boldsymbol{\mu}_i$, are dependent on the adjoint vectors at the next time step, $\boldsymbol{\lambda}_{i+1}$ and $\boldsymbol{\mu}_{i+1}$, as observed in Eq. (37). The system

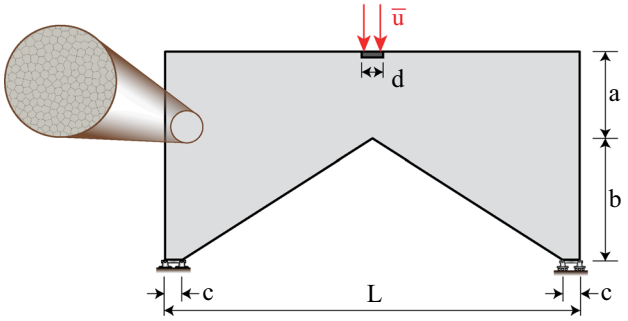


Fig. 6 Portal frame domain and boundary conditions

of adjoint equations is formed and solved in the method `compute_sensitivity_contribution()`. Within this function the required partial derivatives of the local residual, global residual, and plastic work objective function are computed. The local residual derivatives in the method `compute_local_residual_sensitivity_per_elem_type()` are discussed in more detail in the following section. The `compute_sensitivity_contribution()` function updates the objective function sensitivity after determining the $\boldsymbol{\lambda}_i^T$ and $\boldsymbol{\mu}_i^T$ adjoint vectors and returns the components of the updated $\boldsymbol{F}_c^{i-1}$ and $\boldsymbol{F}_u^{i-1}$ terms associated with the $\partial \boldsymbol{H}_i / \partial \boldsymbol{v}_{i-1}$ and $\partial \boldsymbol{H}_i / \partial \boldsymbol{u}_{i-1}$ derivatives. This method is then called again in a `for` loop for the remaining time steps, $i = n - 1, ..., 1$, until the final update of the analytical expression for the augmented objective function sensitivity from Eq. (40) is complete. For further information on the procedure, see Algorithm 2. The final step for obtaining the complete sensitivity with respect to the design variables, $\partial \hat{f} / \partial \mathbf{z}$ is to apply the chain rule as shown in Eq. (42). To observe the verification of the implemented sensitivity analysis in `PolyPlas`, see Appendix B.

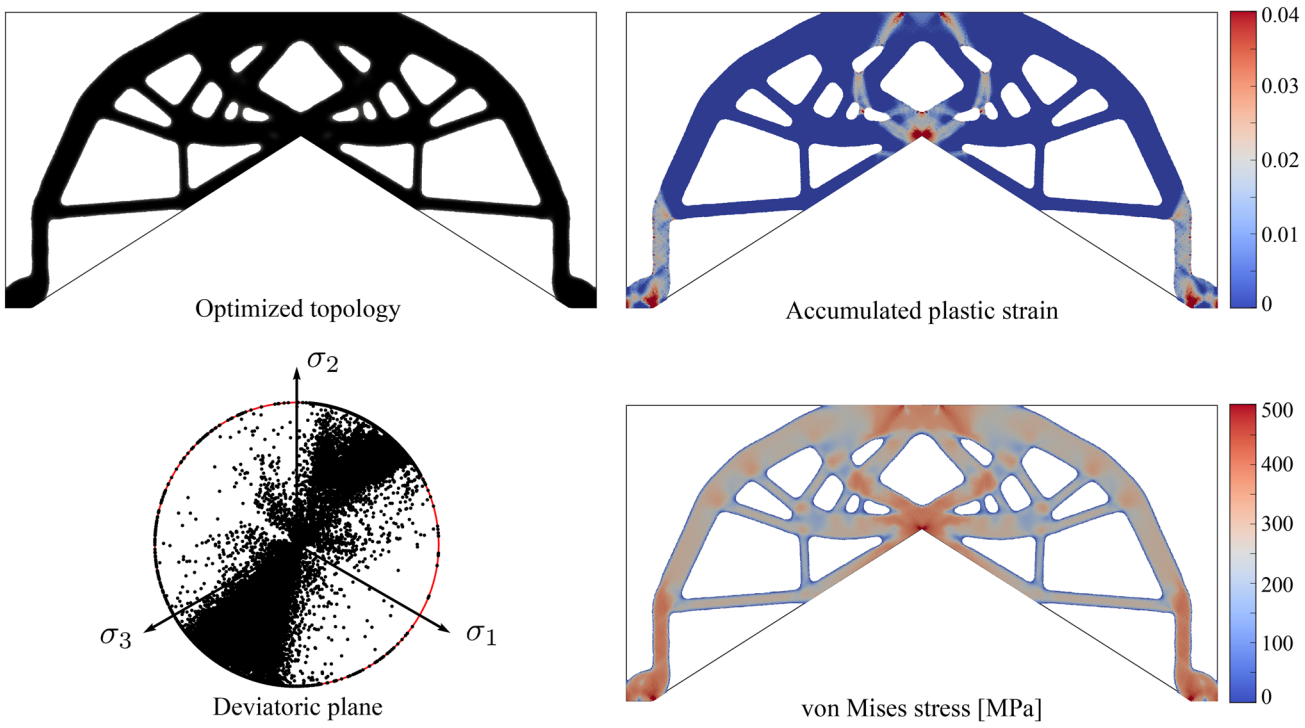


Fig. 7 Portal frame results featuring: the optimized topology (top-left), the accumulated plastic strain distribution in the optimized structure (top-right), the deviatoric plane with 12,464 quadrature points on the yield surface (bottom-left), and the von Mises stress distribution of the optimized structure (bottom-right)

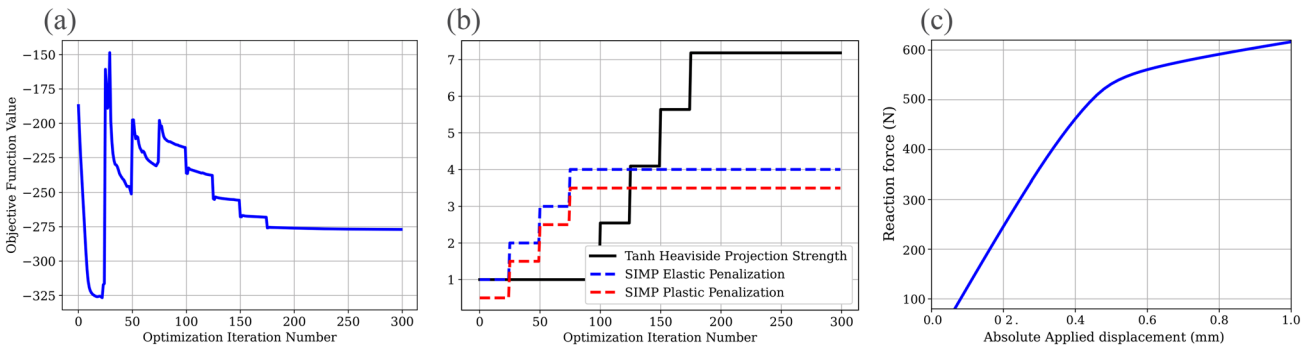


Fig. 8 Portal frame **a** objective function convergence history; **b** penalization and projection schemes information; **c** forward analysis reaction force versus absolute applied displacement of the final optimized structure

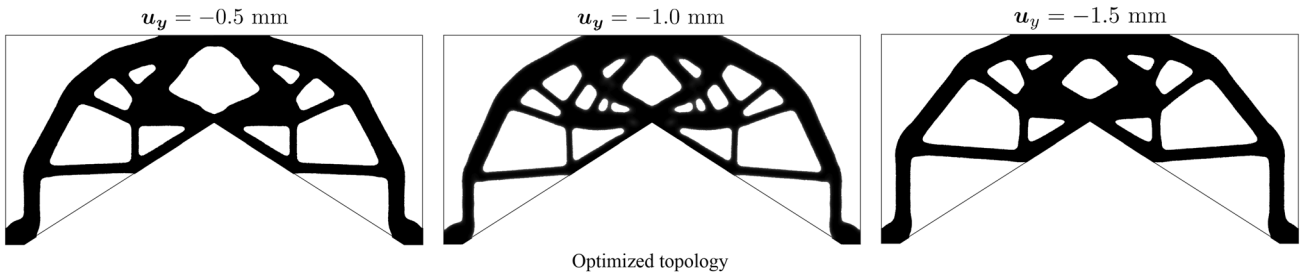


Fig. 9 The optimized topologies of the Portal frame for three cases of applied displacement: $u_y = -0.5$ mm (left), $u_y = -1.0$ mm (middle), and $u_y = -1.5$ mm (right)

Algorithm 2 Procedure for path-dependent sensitivity analysis

Start by initializing the objective function derivative $d\hat{f}/d\boldsymbol{\rho} = \mathbf{0}$
Load **state** at final time step, n
Compute partial derivatives needed to assemble the system in Equation (36)
Construct $\boldsymbol{F}_c^n$ and $\boldsymbol{F}_u^n$ per Equation (36)
Solve Equations (38) and (39) for $\boldsymbol{\lambda}_n$ and $\boldsymbol{\mu}_n$
Update the $d\hat{f}/d\boldsymbol{\rho}$ derivative via Equation (40)
Compute and store $\boldsymbol{F}_c^{n-1}$ and $\boldsymbol{F}_u^{n-1}$ containing $\partial \boldsymbol{H}_n/\partial \boldsymbol{v}_{n-1}$ and $\partial \boldsymbol{H}_n/\partial \boldsymbol{u}_{n-1}$
for $i = n-1$ to 1 **do**
    Load `state` at current time step, i
    Compute partial derivatives needed to assemble the system in Equation (37)
    Construct $\boldsymbol{F}_c^i$ and $\boldsymbol{F}_u^i$ per Equation (37) using the previously stored terms
    Solve Equations (38) and (39) for $\boldsymbol{\lambda}_i$ and $\boldsymbol{\mu}_i$
    Update the $d\hat{f}/d\boldsymbol{\rho}$ derivative via Equation (40)
    Compute and store $\boldsymbol{F}_c^{i-1}$ and $\boldsymbol{F}_u^{i-1}$ containing $\partial \boldsymbol{H}_i/\partial \boldsymbol{v}_{i-1}$ and $\partial \boldsymbol{H}_i/\partial \boldsymbol{u}_{i-1}$
end for

5.4.2 Local residual derivatives

Here we present a description of the local residual partial derivatives in the `PolyPlas` program, primarily focusing on the implementation of the local residual partial derivative with respect to the current internal local variables, $\partial \boldsymbol{H}_i^{e_q}/\partial \boldsymbol{v}_i^{e_q}$. While the explicit expressions for the local residual derivatives were shown in Sect. 4.1, here we demonstrate how the partial derivatives were organized and implemented. The local residual derivative is stored as a higher dimensional array with dimensions of $(q, n_{dof}^{local}, n_{dof}^{local})$ where q represents the total number of quadrature points per element type and n_{dof}^{local} represents the total number of local degrees of freedom (i.e., the internal local variables: $\boldsymbol{\alpha}^{e_q}, \Delta\boldsymbol{\gamma}^{e_q}, \boldsymbol{\varepsilon}_{xx}^{p,e_q}, \boldsymbol{\varepsilon}_{yy}^{p,e_q}, \boldsymbol{\varepsilon}_{xy}^{p,e_q}, \boldsymbol{\varepsilon}_{zz}^{p,e_q}$). The layout of the local residual derivative with respect to the local variables for a single quadrature point is shown for both the elastic and plastic step in Fig. 4. Note that the e_q terms in the superscripts of the partials are omitted for conciseness.

One can interpret this implementation as the rows corresponding to the update rules of the internal local variables and the columns corresponding to the internal local variable of differentiation. The definition of the update laws is shown in Eqs. (23) and (24) where the update of the accumulated plastic strain α_i corresponds to $h_{i,1}$, the update of the plastic multiplier $\Delta\gamma_i$ partially governed by the yield function in $h_{i,2}$, and the update of the plastic strain $\boldsymbol{\varepsilon}_i^p$ corresponding to the flow rule by $h_{i,3}$. Here the superscripts of the flow rule $h_{i,3}^{jk}$ correspond to the jk^{th} component of the tensor. We also denote $\varepsilon_{11}^p = \varepsilon_{xx}^p$, $\varepsilon_{22}^p = \varepsilon_{yy}^p$, $\varepsilon_{12}^p = \varepsilon_{xy}^p$, $\varepsilon_{33}^p = \varepsilon_{zz}^p$. This derivative is computed in the method `compute_local_residual_sensitivity_per_elem_type()`, on lines 1135 – 1384. The procedure begins by creating an array containing all quadrature point indices undergoing elastic loading denoted by `elastic_indices` which are then used to compute the `local_residual_derivative_wrt_current_local_variables_per_elem_type` values using the partial derivative expressions formerly derived in Eqs. (46). For the fourth order tensor of the partial, $\partial h_{i,3}/\partial \boldsymbol{\varepsilon}_i^p = \mathbb{I}^{sym}$, the first two indices correspond to the components of the local residual equation and the last two indices represent the plastic strain tensor component. The implementation of this is shown

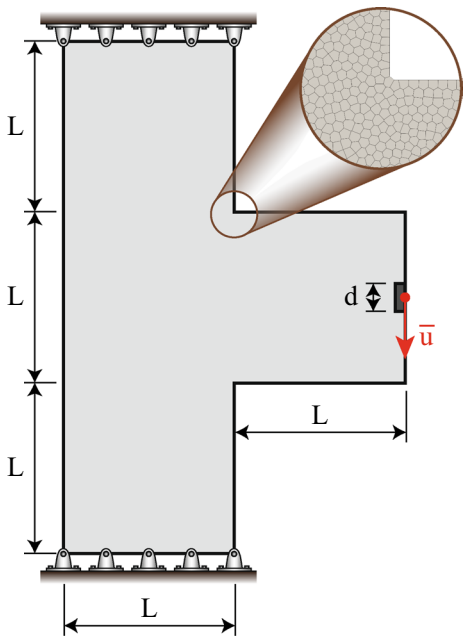


Fig. 10 Corbel domain and boundary conditions

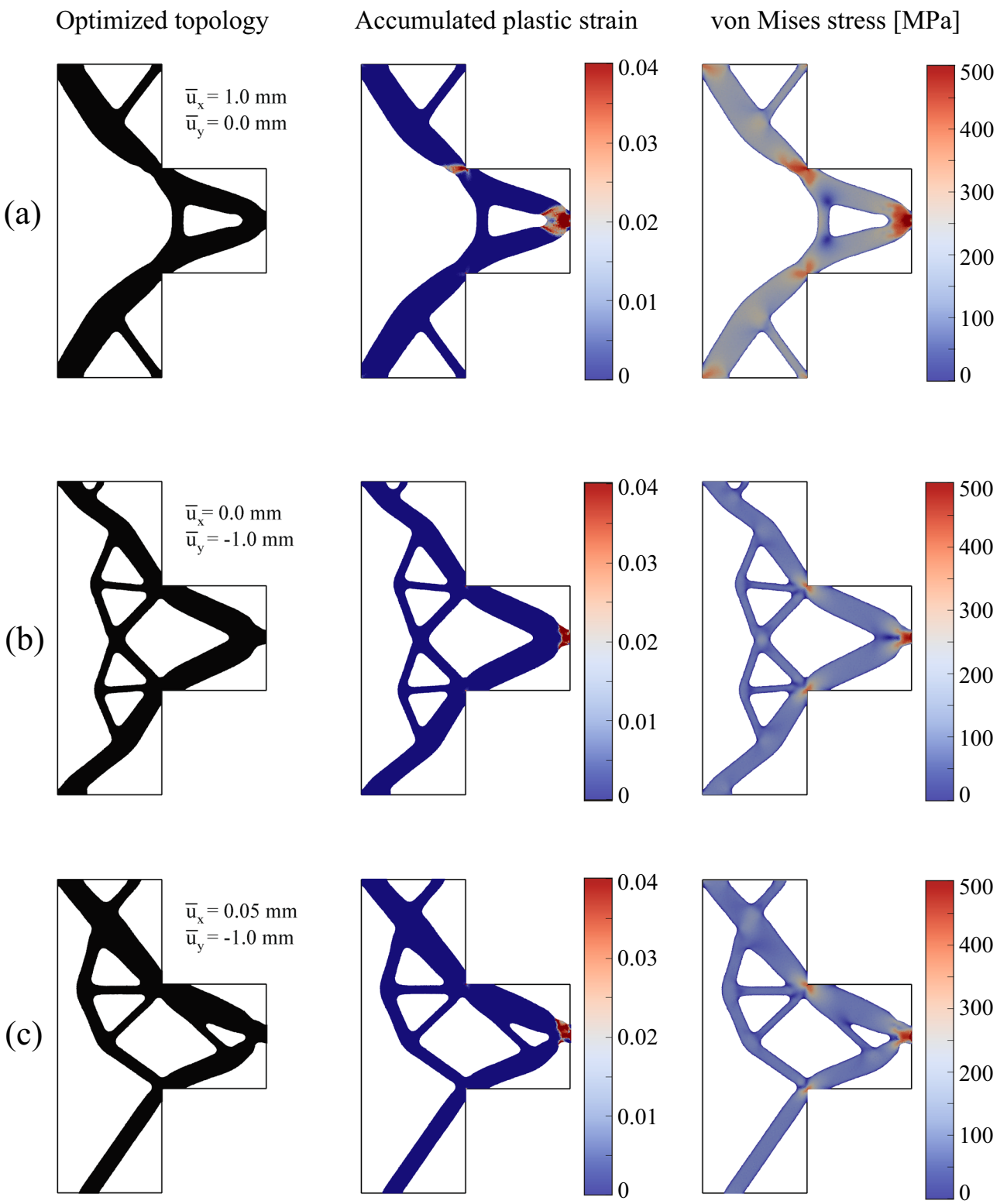


Fig. 11 Results of the Corbel structure showcasing the optimized topology and the distribution of the accumulated plastic strain and the von Mises stress for three different combinations of applied displacements: **a** an applied displacement of $\bar{\boldsymbol{u}}_x = 1.0$ mm with a final plastic work of $W^p = 1587.46$ N-mm, **b** an applied displacement of $\bar{\boldsymbol{u}}_y = -1.0$ mm with a final plastic work of $W^p = 431.37$ N-mm, and **c** an applied displacement of $\bar{\boldsymbol{u}}_x = 0.05$ mm and $\bar{\boldsymbol{u}}_y = -1.0$ mm with a final plastic work of $W^p = 463.86$ N-mm

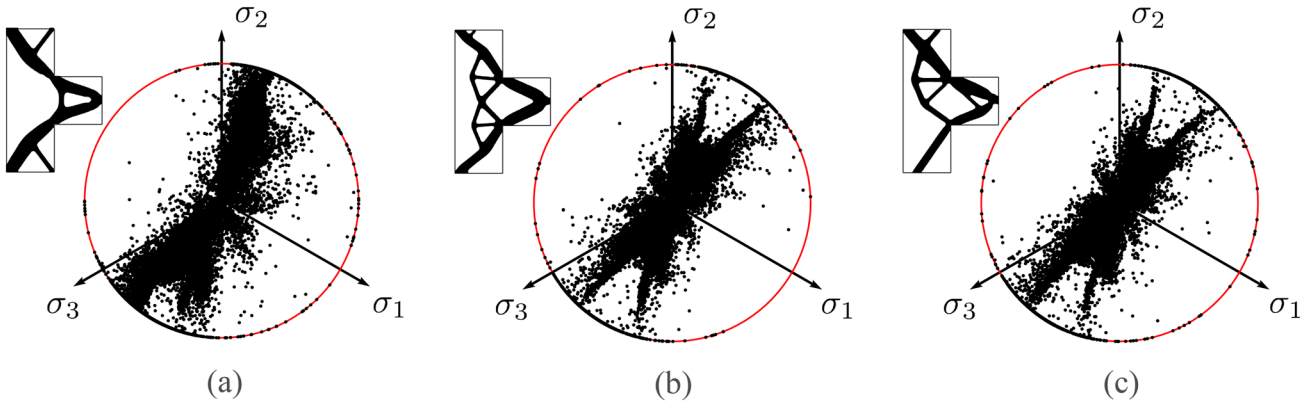


Fig. 12 Deviatoric plane next to the corresponding Corbel structure containing **a** 4,361 quadrature points on the yield surface, **b** 6,893 quadrature points on the yield surface, and **c** 6,796 quadrature points on the yield surface, out of a total of 148,083 quadrature points

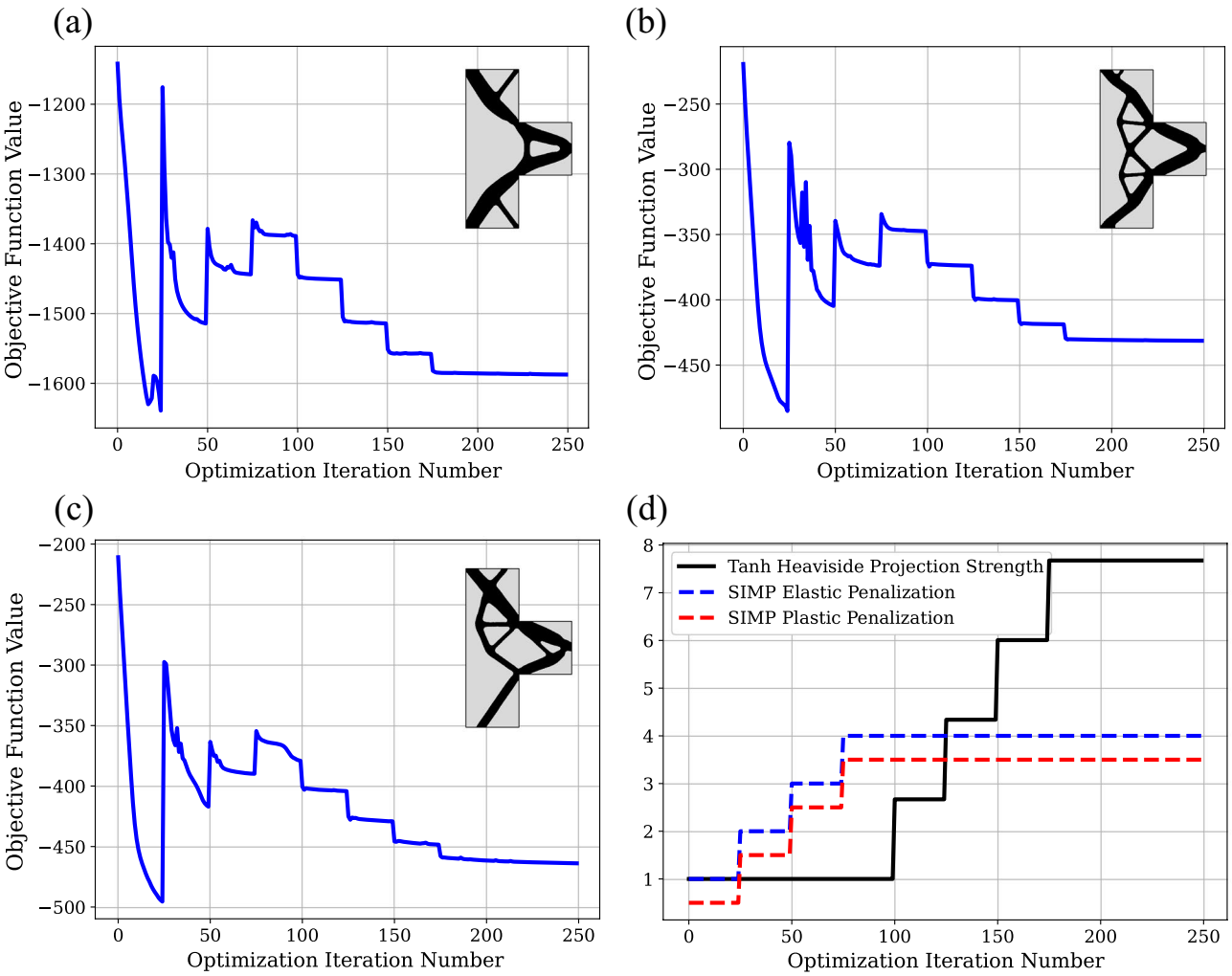


Fig. 13 Corbel optimization details including the convergence plots with their corresponding structures in parts **a**–**c** and **d** the material penalization parameters and projection strength histories

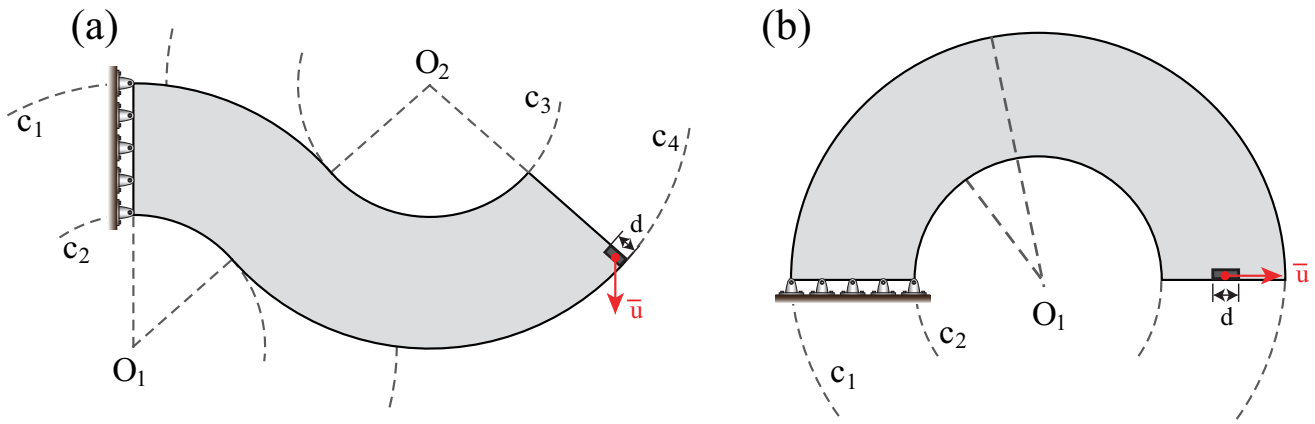


Fig. 14 Geometries and boundary conditions of the curved domains: **a** the Serpentine domain and **b** the Curved beam

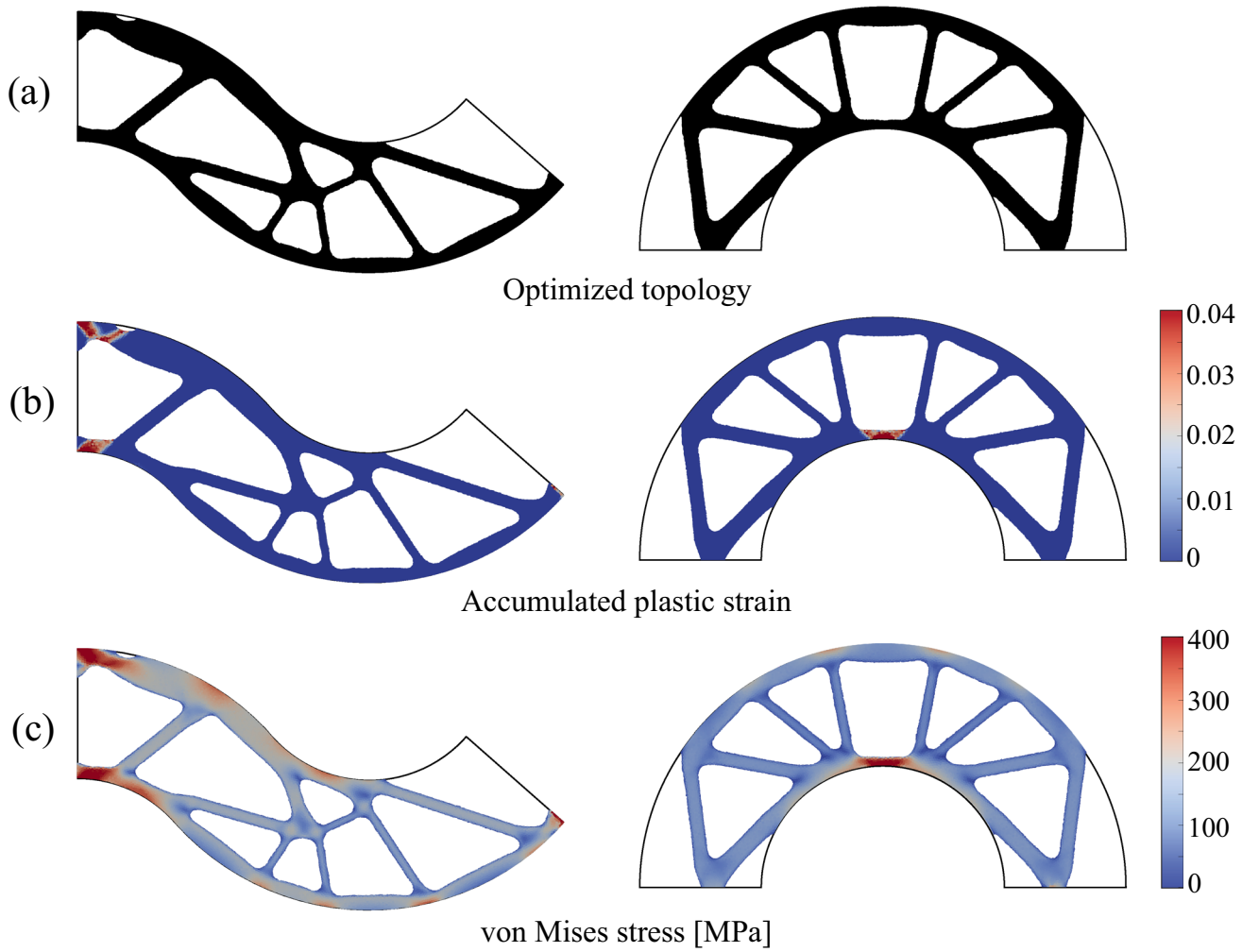


Fig. 15 Results of the Serpentine domain (left) and Curved beam (right) with **a** illustrating the optimized topology, **b** the distribution of the accumulated plastic strain and (c) the distribution of the von Mises stress

on lines 1208 – 1230. The remaining partial derivatives, $\partial h_{i,1}/\partial \alpha_i = \partial h_{i,2}/\partial \Delta\gamma_i = 1.0$ are inserted into the `local_residual_derivative_wrt_current_local_variables_per_elem_type` tensor to resemble the left matrix in Fig. 4.

For the quadrature points undergoing plastic flow, which we index by `plastic_indices`, the computation requires additional implementation steps using the `oe.contract` method. An example of this is shown in the computation of the flow rule derivative with respect to the plastic multiplier, $\partial h_{i,3}/\partial \Delta\gamma_i$.

```
flow_rule_derivative_wrt_plastic_multiplier = oe.contract("...,q,qij->qij",
                                -(3.0/2.0)**0.5,
                                1.0/uninterpolated_deviatoric_stress_norms
                                [plastic_indices],
                                uninterpolated_deviatoric_stress_tensor
                                [plastic_indices])
```

Once this is computed, the components of this tensor are inserted into their corresponding locations in the `local_residual_derivative_wrt_current_local_variables_per_elem_type` array, see lines 1332 – 1335 in `PolyPlas.py` for reference. The remaining partial derivatives are computed and stored similarly, along with the derivative of the local residual with respect to the previous local variables $\partial \boldsymbol{H}_i/\partial \boldsymbol{v}_{i-1}$. The remaining derivatives $\partial \boldsymbol{H}_i/\partial \boldsymbol{u}_i$, $\partial \boldsymbol{H}_i/\partial \boldsymbol{u}_{i-1}$, and $\partial \boldsymbol{H}_i/\partial \rho_e$ are computed and stored in a similar manner as well, corresponding to Eqs. (47) and (50).

The correct implementation of the path-dependent sensitivity analysis is a nontrivial task. In addition to the finite difference checker of the objective function and volume function derivatives with respect to the design variables in the `PolyPlas` method `finite_difference_sensitivity_checker()`, there exist multiple unit tests in `test_polyplas.py` which perform individual finite difference checks on each explicit partial derivative to help ensure accuracy. These unit tests played a significant role in the development of the complete sensitivity analysis and a similar protocol is advised for one who aims to extend this framework.

5.5 Topology optimization procedure overview

Here, flowchart 5 is presented to guide the reader through all the pieces involved in the topology optimization procedure considering elastoplasticity. The main method, `run_top_opt_problem()`, drives the solutions of the topology optimization problem. This method takes the `MMA_state`, the maximum number of optimization iterations, the convergence tolerance, and the `check_grad` boolean variable as input. Within this method the design variables $\mathbf{z}$ are filtered and projected to obtain $\boldsymbol{\rho}$ which are the physical density variables used in the elastoplastic forward analysis. The method `get_function_values_and_gradients()` calls methods to run the elastoplastic forward analysis, obtain the plastic work objective function, and obtain the sensitivity information for both the objective function, $\partial \hat{f}/\partial \mathbf{z}$ and the volume constraint, $\partial g/\partial \mathbf{z}$. Although in this framework the plastic work objective function is chosen, the design of `PolyPlas` is highly modular, allowing for easy extension to other objective functions. Within this method, the appropriate material interpolation and projection scheme parameters are updated in a continuation scheme. Back in the `run_top_opt_problem()` method, the

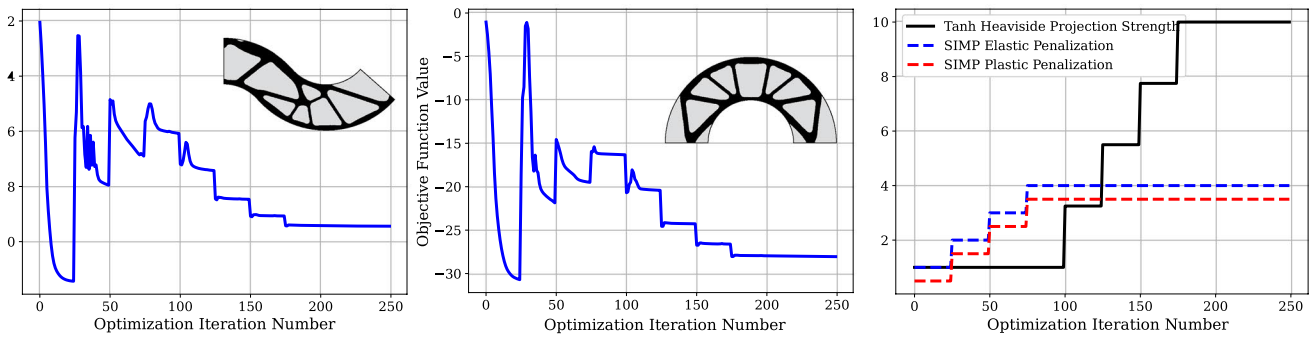


Fig. 16 Optimization details of the curved domains with the objective function convergence for the Serpentine domain (left) and for the Curved beam (middle), with their corresponding projection strengths and material penalization parameters (right)

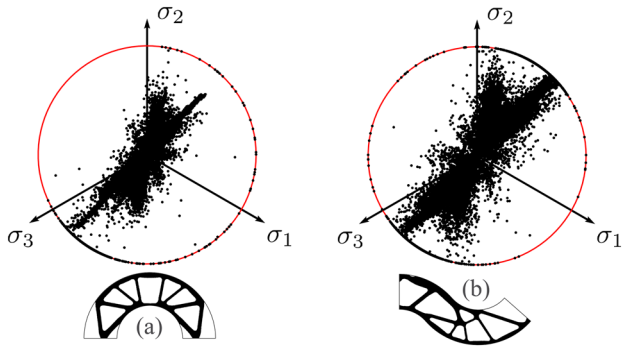


Fig. 17 Deviatoric planes for the curved domains; **a** the Curved beam with 767 quadrature points on the yield surface from a total of 94,650 and **b** the Serpentine domain with 2, 131 quadrature points on the yield surface from a total of 94,628

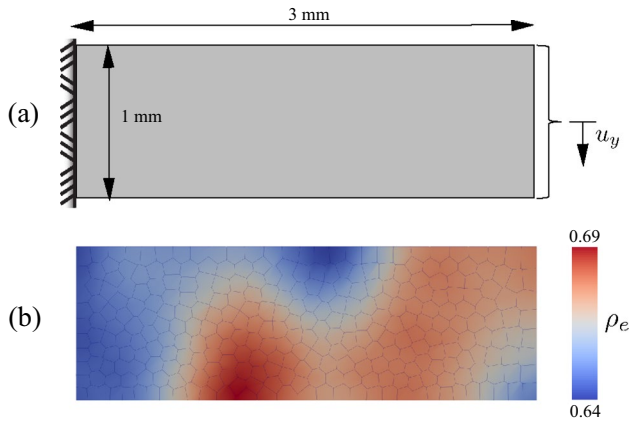


Fig. 18 Illustration of the cantilever beam used in the sensitivity analysis verification with **a** dimensions and boundary conditions and **b** the random density distribution

objective function is scaled using the absolute value of the objective function at the first optimization iteration, while the volume constraint is scaled based the magnitude of the bound, both of which help to ensure optimal performance of the MMA algorithm. Next the `perform_design_update()` method updates the design variables using an adaptation of the MMA implementation by Deetman (2024) (a Python implementation of the original Matlab MMA code by Svanberg (1987)). The iteration continues until either the maximum number of optimization iterations has been reached or the L_∞ norm of the design variable update is less than the specified tolerance (see Fig. 5).

6 Numerical examples

The numerical results presented in this section are intended to showcase the capabilities of the `PolyPlas` program and highlight a few key features. A summary of the example domains and problem setup information is provided in Appendix D. Note that for all examples we employ a continuation strategy on both the elastic and plastic penalization parameters to help alleviate numerical issues associated with large plastic strains in low-density regions. We begin with exponents $p = 1,\ q = 0.5$ and increase their values by 0.5 after every 25 optimization iterations until they reach $p = 4,\ q = 3.5$ at $j = 75$. After the continuation on the material interpolation, a continuation scheme is imposed on the projection strength as well, see Sect. 3.2 for details. The material parameters are the same across all examples and are set to closely align with the material properties calibrated for Aluminum 2024-T351 (Bao and Wierzbicki 2004), see Table 3. The optimization problem is updated using the Method of Moving Asymptotes with a move limit of 0.5. To help prevent divergence, the applied displacement is slowly increased such that the first few increments correspond to primarily elastic loading before transitioning into plastic deformation. Such an approach is recommended for all examples to help ensure the convergence of the elastoplastic forward analysis during the optimization process. Further information regarding the problem setup can be found in each problem's description below.

6.1 Portal frame

This first example analyzes the portal frame which is a typical benchmark problem found in the elastoplastic topology optimization literature. The portal frame domain and boundary conditions are illustrated in Fig. 6 where the dimensions are set by $L = 60$ mm, $a = 12.5$ mm, $b = 17.5$ mm, $c = 2.75$ mm, and $d = 4$ mm. Half of the domain is analyzed for computational efficiency using the `HalfPortalFrame` boundary value problem with a finite element mesh composed of 20,500 polygonal elements. The polynomial filter is used with a filter radius of $R = 1.0$ mm and the volume fraction upper bound is set to $V_{max} = 0.4$. The forward problem is run with 14 time steps for a final total applied displacement of $\bar{\boldsymbol{u}}_y = -1.0$ mm. The optimization parameters include a maximum number of iterations of 300, and a convergence tolerance of tol $= 10^{-8}$.

Maximizing plastic work leads to energy absorbing elastoplastic structures as illustrated in Fig. 7. It is observed that the structure undergoes a significant amount of plastic strain where a high concentration of the accumulated plastic strain forms between the applied displacement region and the re-entrant corner of the frame, in addition to the region around the supports. Regions of high von Mises stress correspond to the regions of high accumulated plastic strain, as consistent with the von Mises yield criteria. The normalized deviatoric plane is shown illustrating the 12,464 quadrature points that are on the yield surface where $\sim 10.26\%$ of all quadrature points are undergoing plastic flow. The final plastic work of the structure is $W^p = 276.96$ N-mm. Additionally, the unscaled objective function convergence history is

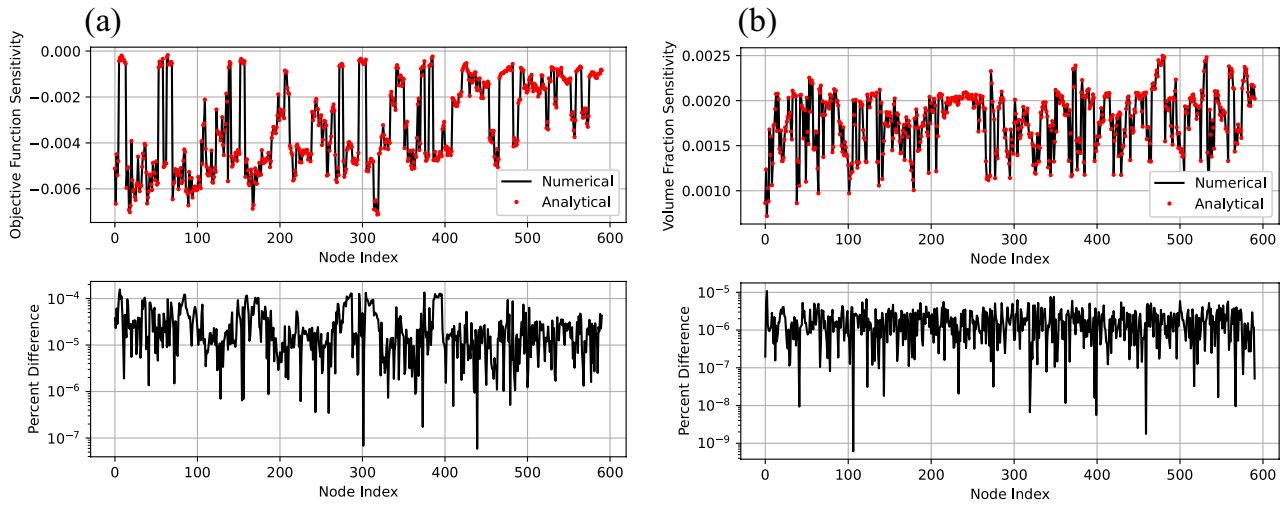


Fig. 19 Comparison of the numerical and analytical sensitivities for **a** the objective function (i.e., plastic work) and **b** the volume fraction along with the corresponding percent difference

shown in Fig. 8a where the spikes in the convergence correspond to the parameter updates in the continuation scheme, shown in Fig. 8b. Lastly, we show the forward analysis of the final optimized topology plotting the reaction force versus the applied displacement in Fig. 8c, where we observe the transition from a predominantly elastic response to a plastic response.

6.1.1 Applied displacement magnitude review

To examine the affect of the applied displacement magnitude on the elastoplastic topology optimization problem, we examine two additional applied displacement cases for the Portal frame: $\boldsymbol{u}_y = -0.5$ mm over 7 pseudo-time steps and $\boldsymbol{u}_y = -1.5$ mm over 21 pseudo-time steps. The time steps are defined such that all cases have the same applied displacement increment size. The examples have the same problem setup information as indicated in the previous Portal frame result. The final topologies, including the original example from Sect. 6.1, are shown in Fig. 9. From these results, we observe how even a slight change in the applied displacement leads to drastically different final designs. The plastic work objectives are reported as $W^p = 65.74$ N-mm for the case of $\boldsymbol{u}_y = -0.5$ mm, $W^p = 276.96$ N-mm for $\boldsymbol{u}_y = -1.0$ mm (from the previous example), and $W^p = 613.71$ N-mm for the final case of $\boldsymbol{u}_y = -1.5$ mm.

6.2 Corbel structure

Here, the Corbel structure is optimized for three different applied displacement combinations. The Corbel has a side length, $L = 40$ mm and an applied displacement region width of $d = 3.2$ mm with boundary conditions shown in Fig. 10. The finite element mesh is composed of 25,000 unstructured polygonal elements and contains a passive region near the applied displacement boundary with a size of $d \times w$ where $w = 8.0\tau$. This was placed in an effort to avoid numerical instabilities related to high plastic strains near the applied displacement boundary. The polynomial filter is used with a filter radius of $R = 3.0$ mm. We examine three load cases with displacement applied over 14 time steps; (a) $\bar{\boldsymbol{u}}_x = 1.0$ mm $\bar{\boldsymbol{u}}_y = 0.0$ mm, (b) $\bar{\boldsymbol{u}}_x = 0.0$ mm $\bar{\boldsymbol{u}}_y = -1.0$ mm, and (c) $\bar{\boldsymbol{u}}_x = 0.05$ mm $\bar{\boldsymbol{u}}_y = -1.0$ mm. The continuation on the projection strength and material interpolation exponents is shown in Fig. 13. Finally, the domain is optimized considering a volume fraction upper bound of $V_{max} = 0.35$, a maximum number of optimization iterations of 250, and a convergence tolerance of tol $= 5.0 \cdot 10^{-4}$.

As observed in Fig. 11, the direction of the applied displacements leads to vastly different topologies. For the first example in Fig. 11 part (a), the topology contains two main thick members to maximize plastic work for displacement in the x direction. We also notice high regions of accumulated plastic strains and von Mises stress around the region of applied displacement and in the re-entrant corners. For the Corbel structure subject to displacement in the y direction in Fig. 11 part (b) we observe additional members in the topology which act as bracing to maximize energy absorption for the downward displacement. Although both of the previous applied displacements form a symmetric boundary value problem, symmetry is not imposed and thus we do not obtain perfect symmetry due to the unstructured mesh. For

the final applied loading case in Fig. 11 part (c), we added a small contribution of applied displacement in the x direction in addition to the same previous applied displacement in case (b). With this, we obtain an asymmetric design with a significant portion of the structure's volume in the top half of the domain. While the final topologies of the final two applied displacement cases don't resemble each other, their deviatoric plane do with similar principal stress distribution as observed in Fig. 12. The deviatoric plane for case (a) shows a larger distribution of quadrature points near the yield surface; however, there are fewer quadrature points on the yield surface at 4,361 versus the 6,893 and 6,796 quadrature points for cases (b) and (c), respectively. Finally, the convergence plots are shown in Fig. 13 where all examples reach the maximum number of optimization iterations.

6.3 Curved domains

The final numerical examples presented here showcase an important feature of unstructured polygonal elements: the meshing of complex, curved domains. Here, we optimize two curved cantilever beams, the first of which we denote as the Serpentine domain and the second which we call the Curved beam, as shown in Fig. 14. The Serpentine domain has dimensions defined by origins, $O_1 = (0, -2.6458\text{ mm})$ and $O_2 = (9\text{ mm}, 5.2915\text{ mm})$ and curves defined by $c_1 = (O_1, 8.0\text{ mm})$, $c_2 = (O_1, 4.0\text{ mm})$, $c_3 = (O_2, 4.0\text{ mm})$, and $c_4 = (O_2, 8.0\text{ mm})$ following the notation $c = (\text{origin}, \text{radius})$; it also has an applied displacement width of $d = 0.53$ mm. The Curved beam obtains dimensions of $O_1 = (0.0, 0.0)$ with curves $c_1 = (O_1, 10.0\text{ mm})$ and $c_2 = (O_1, 20.0\text{ mm})$ with an applied displacement width of $d = 2.5$ mm. The Serpentine domain has an applied displacement of $\bar{\boldsymbol{u}}_y = -0.5$ mm and the Curved beam has an applied displacement of $\bar{\boldsymbol{u}}_x = 0.5$ mm, both imposed over 14 time steps. Both structures are meshed with 16,000 polygonal elements, are subject to a volume fraction upper bound of $V_{max} = 0.4$, and use a polynomial filter with a relative radius of $R = 5.0\tau$. For the Curved beam we impose symmetry about the y-axis (see Fig. 15).

The results of the curved domains are shown in Fig. 15 where the Serpentine domain has large amounts of accumulated plastic strains near the supports and the applied displacement region. The Curved domain instead experiences high accumulated plastic strain in the inner, center location of the beam. The final plastic work of the Curved beam and the Serpentine domain are $W^p = 28.05$ N-mm and $W^p = 9.44$ N-mm as seen in the unscaled objective function convergence plots in Fig. 16. In addition, because both of the domains use the relative filter radius, they also have the same maximum projection strength. The stress states are illustrated in Fig. 17.

7 Conclusion

This paper places significant emphasis on the topology optimization methodology considering elastoplasticity for pedagogical purposes, from the nonlinear forward problem to the complex path-dependent sensitivity analysis. The authors motivate the use of the `oe.contract` method from the `opt-einsum` package for an educational and intuitive transition from equations on paper to implementation in code. The `PolyPlas` program is demonstrated to produce optimized results with high energy absorption through the framework of maximizing the plastic work.

A detailed discussion on the theoretical framework of elastoplastic topology optimization is presented in conjunction with providing a modular, open-source program, `PolyPlas`. This code is inspired by the preexisting `PolyTop` programs; however, it is developed in an entirely new framework in Python utilizing object oriented programming for improved modularity and organization. We wish to add to the library of educational topology optimization codes considering unstructured polygonal elements. Similar to the former developments, `PolyPlas` contains a general framework allowing the consideration of complex domain geometries, utilizing a Python implementation of `PolyMesher`. By introducing an educational code for topology optimization considering elastoplasticity, we hope that `PolyPlas` stimulates further advancement in the field of topology optimization considering energy-dissipative phenomena by means of an open-source program for academic and industrial users alike.

Appendix A Return mapping algorithm

Here, the standard return mapping algorithm is described as presented in de Souza Neto et al. (2011). The procedure of updating the state variables at the next pseudo-time step begins with the elastic trial state,

$$\boldsymbol{\varepsilon}_{i+1}^{p,trial} = \boldsymbol{\varepsilon}_i^p \tag{64}$$

$$\boldsymbol{s}_{i+1}^{trial} = 2G\mathbb{P}^{dev}(\boldsymbol{\varepsilon}_{i+1} - \boldsymbol{\varepsilon}_{i+1}^{p,trial}) \tag{65}$$

$$\alpha_{i+1}^{trial} = \alpha_i. \tag{66}$$

If the trial state is admissible, (i.e., $\Phi(\boldsymbol{s}_{i+1}^{trial}, \alpha_{i+1}^{trial}) \leq 0$), then it is accepted,

$$\boldsymbol{\varepsilon}_{i+1}^{p} = \boldsymbol{\varepsilon}_{i+1}^{p,trial} \tag{67}$$

$$\boldsymbol{s}_{i+1} = \boldsymbol{s}_{i+1}^{trial} \tag{68}$$

$$\alpha_{i+1} = \alpha_{i+1}^{trial} \tag{69}$$

$$\mathbb{C}_{i+1}^{dev} = 2G\mathbb{P}_{dev} \tag{70}$$

where we also provide the deviatoric part of the consistent tangent, $\mathbb{C}_{i+1}^{dev}$. If the trial state is not admissible (i.e., $\Phi(\boldsymbol{s}_{i+1}^{trial}, \alpha_{i+1}^{trial}) > 0$), then the radial return mapping is performed. The corresponding update of the plastic multiplier increment may be expressed in closed form for the linear hardening rule assumed in this work:

$$\Delta\gamma_{i+1} = \frac{\Phi(\boldsymbol{s}_{i+1}^{trial}, \alpha_{n+1}^{trial})}{3G + H} \tag{71}$$

Alternatively, one could extend this framework to nonlinear hardening functions in which the plastic multiplier increment may be determined via a local Newton iteration. The corresponding update of the state variables for the plastic step is provided below, along with the deviatoric part of the consistent tangent.

$$\boldsymbol{\varepsilon}_{i+1}^{p} = \boldsymbol{\varepsilon}_{i+1}^{p,trial} + \Delta\gamma\boldsymbol{N} \tag{72}$$

$$\boldsymbol{s}_{i+1} = \left(1 - \frac{\Delta\gamma 3G}{q_{i+1}^{trial}}\right)\boldsymbol{s}_{i+1}^{trial} \tag{73}$$

$$\alpha_{i+1} = \alpha_{i+1}^{trial} + \Delta\gamma \tag{74}$$

$$\boxed{\mathbb{C}_{i+1}^{dev} = 2G\left(1 - \frac{\Delta\gamma 3G}{q_{i+1}^{trial}}\right)\mathbb{P}^{dev} + 6G^2\left(\frac{\Delta\gamma}{q_{i+1}^{trial}} - \frac{1}{3G + H}\right)\boldsymbol{N}\otimes\boldsymbol{N}} \tag{75}$$

Appendix B Sensitivity analysis verification

Here, the details regarding the sensitivity analysis verification are presented. A cantilever beam is analyzed with a downward prescribed displacement of −0.1 mm, applied over 10 time steps with material properties as shown in Table 3 and a finite element mesh composed of 300 polygonal elements. The dimensions and boundary conditions are depicted in Fig. 18.

The numerical sensitivities are obtained by the central finite difference method,

$$\frac{df(\boldsymbol{x})}{dx_i} \approx \frac{f(\boldsymbol{x} + \Delta\boldsymbol{x}_i) - f(\boldsymbol{x} - \Delta\boldsymbol{x}_i)}{2\Delta\boldsymbol{x}} \tag{76}$$

where the $\Delta\boldsymbol{x}_i$ denotes a vector with the only nonzero perturbation of 10^{-6} corresponding to design variable i. The numerical sensitivity is computed for each design variable and is compared against the analytical sensitivity as shown in Fig. 19. This procedure is performed in the `PolyPlas` function **finite_difference_sensitivity_checker()**.

Appendix C The `oe.contract` tutorial

The `oe.contract` method from the `opt-einsum` package (Daniel and Gray, 2018) is a powerful tool used in the `PolyPlas` program to evaluate expressions in Einstein summation convention. For mechanicians this enables a smoother transition from the derivations on paper to the implementation in code. This package was built off of Numpy's einsum package (Harris et al. 2020) to perform in the same way as the `np.einsum` method; however, it reduces the execution time of large `einsum` operations by using algorithms to optimize the execution of complex contractions. This tutorial aims to demonstrate some examples relating the mechanics equations to their corresponding `oe.contract` application. For more information on the `opt_einsum` package, see the official documentation.

Starting off with the standard linear elastic relationship for the Cauchy stress tensor by the contraction of the fourth order isotropic material tensor and the elastic strain tensor we have the following in summation notation,

$$\boldsymbol{\sigma}_{ij} = \mathbb{C}_{ijkl}\boldsymbol{\varepsilon}_{kl} \tag{77}$$

which may be written equivalently in the program by,

```
stress = oe.contract("ijkl,kl->ij", material_tensor, elastic_strain)
```
(78)

where the first argument in `oe.contract` contains the indices of the input variables and the indices of the resulting tensor are placed to the right of the arrow. For scalar output, there are no indices places to the right of the arrow. The remaining entries correspond to the tensorial inputs for the described operation (e.g., the material constitutive tensor and the elastic strain tensor).

For further flexibility, `oe.contract` also allows alternative array operations that may not follow the standard rules of classical Einstein notation. An example of this that is commonly used in PolyPlas includes computing a contraction in a vectorized fashion for all elements and/or quadrature points simultaneously. Using the same equation

as before, one could instead compute the stress tensor for each quadrature point in every element by,

stress = oe.contract("eqijkl,eqkl->eqij", material_tensor, elastic_strain). (79)

Notice that this operation doesn't follow the standard Einstein summation rules since the repeated *eq* indices don't behave as dummy indices and no summation is performed over them. Instead, the output results in a contraction operation performed over k and l only.

Below, we present a few more examples of indicial notation expressions and their corresponding implementation in code. This next example demonstrates how the '...' entry broadcasts a constant scalar in the contraction operation when computing the deviatoric stress,

$$\boldsymbol{s}_{ij} = 2G\mathbb{P}^{dev}_{ijkl}\boldsymbol{\epsilon}_{kl} \tag{80}$$

with its corresponding `oe.contract` implementation performed via

deviatoric_stress =oe.contract("...,ijkl,eqkl->eqij", (81)

2.0 * shear_modulus, (82)

deviatoric_projection_tensor, (83)

elastic_strain). (84)

Next a portion of the flow rule derivative with respect to the plastic strain is shown below to showcase how to implement outer products in the `oe.contract` operation. A portion of this partial derivative expression is denoted by $\boldsymbol{c}_{ijkl}$ and `temp_1` in the code.

$$\boldsymbol{c}_{ijkl} = \frac{\boldsymbol{s} \otimes \boldsymbol{s}}{||\boldsymbol{s}||} = \frac{\boldsymbol{s}_{ij}\boldsymbol{s}_{kl}}{(\boldsymbol{s}_{ab}\boldsymbol{s}_{ab})^{1/2}} \tag{85}$$

temp_1 =oe.contract("qij,qkl->qijkl", (86)

deviatoric_stress_over_deviatoric_stress_norm, (87)

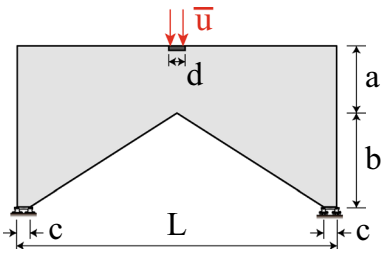
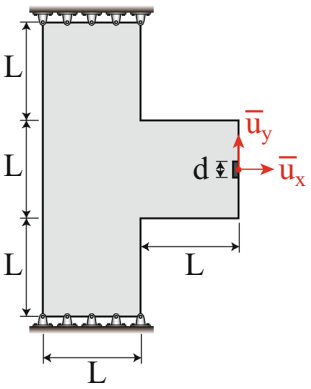
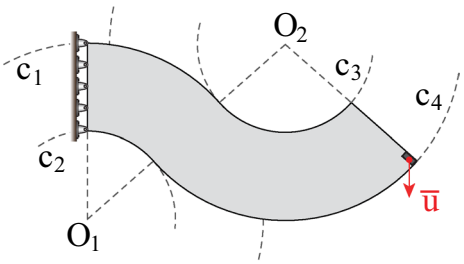
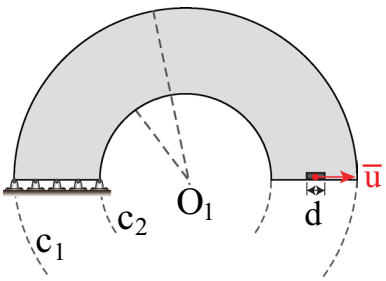
deviatoric_stress_over_deviatoric_stress_norm) (88)

Here the q index refers to the *total* number of quadrature points per element type. The indices can represent any dimension so long as they are consistent within the operation. To view the full implementation of the partial derivative of the flow rule with respect to the plastic strain as seen in Eqs. (48) and 49, see lines $1299 - 1306$ and $1341 - 1359$ of `PolyPlas.py`.

Appendix D Library of benchmark examples

Table 4.

Table 4 Examples provided with `PolyPlas`

Domain	Description
	• Domain: `HalfPortalFrame` • Dimensions: $L = 60$ mm, $a = 12.5$ mm, $b = 17.5$ mm, $c = 2.75$ mm • Applied displacement: $\bar{\boldsymbol{u}}_y = 1.0$ over 14 time steps
	• Domain: `Corbel` • Dimensions: $L = 60$ mm, $d = 3.2$ mm • Applied displacement: (a) $\bar{\boldsymbol{u}}_x = 1.0$ mm $\bar{\boldsymbol{u}}_y = 0.0$ mm, (b) $\bar{\boldsymbol{u}}_x = 0.0$ mm $\bar{\boldsymbol{u}}_y = -1.0$ mm, and (c) $\bar{\boldsymbol{u}}_x = 0.05$ mm $\bar{\boldsymbol{u}}_y = -1.0$ mm; over 14 time steps
	• Domain: `SerpentineDomain` • Dimensions: $O_1 = (0, -2.6458$ mm), $O_2 = (9$ mm, 5.2915 mm), $c_1 = (O_1, 8.0$ mm), $c_2 = (O_1, 4.0$ mm), $c_3 = (O_2, 4.0$ mm), $c_4 = (O_2, 8.0$ mm), $d = 0.53$ mm • Applied displacement: $\bar{\boldsymbol{u}}_y = 0.5$ over 14 time steps
	• Domain: `CurvedBeam` • Dimensions: $O_1 = (0.0, 0.0)$, $c_1 = (O_1, 10.0$ mm), $c_2 = (O_1, 20.0$ mm), $d = 2.5$ mm • Applied displacement: $\bar{\boldsymbol{u}}_x = 0.5$ over 14 time steps

Acknowledgements The authors acknowledge the support provided by the National Science Foundation (NSF) Graduate Research Fellowship Program. In addition, we acknowledge support granted by the NSF under grants 2105811 and 2323415. We wish to extend our gratitude to Professor Emily D. Sanders, Professor Oliver Giraldo Londoño, and Mr. Lucien Tsai, for providing useful comments and suggestions. We also would like to acknowledge the support granted by the Margareta Engman Augustine endowment at Princeton University.

Declarations

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

Replication of results To reproduce the numerical results, the authors refer the reader to the Appendix D which contains a library of benchmark examples and their implementation details. All results presented in this manuscript can be replicated using the `PolyPlas` program provided in the supplementary information. For access to the open-source program PolyPlas, see the git repository, https://github.com/emilyalcazar1/PolyPlas.git.

Ethical approval The opinions and conclusions presented in this article are those of the authors and do not necessarily reflect the views of the sponsoring organizations.

References

Aage N, Andreassen E, Lazarov BS (2015) Topology optimization using petsc: an easy-to-use, fully parallel, open source topology optimization framework. Struct Multidisc Optim 51:565–572

Abueidda DW, Kang Z, Koric S, James KA, Jasiuk IM (2021) Topology optimization for three-dimensional elastoplastic architected materials using a path-dependent adjoint method. Int J Numer Meth Eng 122(8):1889–1910

Alberdi R, Khandelwal K (2017) Topology optimization of pressure dependent elastoplastic energy absorbing structures with material damage constraints. Finite Elem Anal Des 133:42–61

Alberdi R, Khandelwal K (2019) Design of periodic elastoplastic energy dissipating microstructures. Struct Multidisc Optim 59:461–483

Alberdi R, Zhang G, Li L, Khandelwal K (2018) A unified framework for nonlinear path-dependent sensitivity analysis in topology optimization. Int J Numer Meth Eng 115(1):1–56

Amir O (2017) Stress-constrained continuum topology optimization: a new approach based on elasto-plasticity. Struct Multidisc Optim 55(5):1797–1818

Andreassen E, Clausen A, Schevenels M, Lazarov BS, Sigmund O (2011) Efficient topology optimization in MATLAB using 88 lines of code. Struct Multidisc Optim 43:1–16

Bao Y, Wierzbicki T (2004) A comparative study on various ductile crack formation criteria. J Eng Mater Technol 126(3):314–324

Bendsøe MP (1989) Optimal shape design as a material distribution problem. Struct Optim 1:193–202

Bendsøe MP, Kikuchi N (1988) Generating optimal topologies in structural design using a homogenization method. Comput Methods Appl Mech Eng 71(2):197–224

Bogomolny M, Amir O (2012) Conceptual design of reinforced concrete structures using topology optimization with elastoplastic material modeling. Int J Numer Meth Eng 90(13):1578–1597

Chi H, Pereira A, Menezes IF, Paulino GH (2020) Virtual element method (VEM)-based topology optimization: an integrated framework. Struct Multidisc Optim 62(3):1089–1114

Daniel G, Gray J et al (2018) Opt_einsum-a python package for optimizing contraction order for einsum-like expressions. J Open Source Softw 3(26):753

da Silva GA, Beck AT, Sigmund O (2019) Stress-constrained topology optimization considering uniform manufacturing uncertainties. Comput Methods Appl Mech Eng 344:512–537

Deetman A (2024) Gcmma-mma-python: Python implementation of the method of moving asymptotes (0.3.0). Zenodo. https://doi.org/10.5281/zenodo.13337495

de Souza Neto EA, Peric D, Owen DR (2011) Computational methods for plasticity: theory and applications. Wiley, Hoboken

Ferrari F, Sigmund O (2020) A new generation 99 line MATLAB code for compliance topology optimization and its extension to 3D. Struct Multidisc Optim 62:2211–2228

Ferrari F, Sigmund O, Guest JK (2021) Topology optimization with linearized buckling criteria in 250 lines of MATLAB. Struct Multidisc Optim 63(6):3045–3066

Giraldo-Londoño O, Paulino GH (2021) Polydyna: a MATLAB implementation for topology optimization of structures subjected to dynamic loads. Struct Multidisc Optim 64(2):957–990

Giraldo-Londoño O, Paulino GH (2021) Polystress: a MATLAB implementation for local stress-constrained topology optimization using the augmented lagrangian method. Struct Multidisc Optim 63(4):2065–2097

Guest JK, Prévost JH, Belytschko T (2004) Achieving minimum length scale in topology optimization using nodal design variables and projection functions. Int J Numer Meth Eng 61(2):238–254

Han J, Furuta K, Kondoh T, Nishiwaki S, Terada K (2024) Topology optimization of finite strain elastoplastic materials using continuous adjoint method: formulation, implementation, and applications. Comput Methods Appl Mech Eng 429:117181

Harris CR, Millman KJ, Van Der Walt SJ, Gommers R, Virtanen P, Cournapeau D (2020) Array programming with NumPy. Nature 585(7825):357–362

Ivarsson N, Wallin M, Amir O, Tortorelli DA (2021) Plastic work constrained elastoplastic topology optimization. Int J Numer Meth Eng 122(16):4354–4377

Kato J, Hoshiba H, Takase S, Terada K, Kyoya T (2015) Analytical sensitivity in topology optimization for elastoplastic composites. Struct Multidisc Optim 52:507–526

Kuci E, Jansen M (2022) Level set topology optimization of elastoplastic materials with local stress constraints. Struct Multidisc Optim 65(6):170

Li L, Zhang G, Khandelwal K (2017) Design of energy dissipating elastoplastic structures under cyclic loads using topology optimization. Struct Multidisc Optim 56:391–412

Li L, Zhang G, Khandelwal K (2017) Topology optimization of energy absorbing structures with maximum damage constraint. Int J Numer Meth Eng 112(7):737–775

Li K, Wallin M, Ristinmaa M, Cheng G (2024) A "poor-man's'' deformation plasticity based approach to topology optimization of elastoplastic structures. Int J Solids Struct 305:113056

Liu K, Tovar A (2014) An efficient 3D topology optimization code written in MATLAB. Struct Multidisc Optim 50:1175–1196

Maute K, Schwarz S, Ramm E (1998) Adaptive topology optimization of elastoplastic structures. Struct Optim 15:81–91

Michaleris P, Tortorelli DA, Vidal CA (1994) Tangent operators and design sensitivity formulations for transient non-linear coupled problems with applications to elastoplasticity. Int J Numer Meth Eng 37(14):2471–2499

Nagtegaal JC, Parks DM, Rice J (1974) On numerically accurate finite element solutions in the fully plastic range. Comput Methods Appl Mech Eng 4(2):153–177

Pereira A, Talischi C, Paulino GH, Menezes MIF, Carvalho MS (2016) Fluid flow topology optimization in polytop: stability and computational implementation. Struct Multidisc Optim 54:1345–1364

Russ JB, Waisman H (2021) A novel elastoplastic topology optimization formulation for enhanced failure resistance via local ductile failure constraints and linear buckling analysis. Comput Methods Appl Mech Eng 373:113478

Sanders ED, Pereira A, Aguiló MA, Paulino GH (2018) Polymat: an efficient matlab code for multi-material topology optimization. Struct Multidisc Optim 58:2727–2759

Senhora FV, Giraldo-Londono O, Menezes IF, Paulino GH (2020) Topology optimization with local stress constraints: a stress aggregation-free approach. Struct Multidisc Optim 62:1639–1668

Sigmund O (2001) A 99 line topology optimization code written in matlab. Struct Multidisc Optim 21:120–127

Sigmund O, Maute K (2013) Topology optimization approaches: a comparative review. Struct Multidisc Optim 48(6):1031–1055

Simo JC, Hughes TJ (2006) Computational inelasticity, vol 7. Springer, New York

Simo J, Taylor RL, Pister K (1985) Variational and projection methods for the volume constraint in finite deformation elasto-plasticity. Comput Methods Appl Mech Eng 51(1–3):177–208

Svanberg K (1987) The method of moving asymptotes-a new method for structural optimization. Int J Numer Meth Eng 24(2):359–373

Talischi C, Paulino GH, Pereira A, Menezes IF (2012) Polymesher: a general-purpose mesh generator for polygonal elements written in MATLAB. Struct Multidisc Optim 45:309–328

Talischi C, Paulino GH, Pereira A, Menezes IF (2012) Polytop: a matlab implementation of a general topology optimization framework

using unstructured polygonal finite element meshes. Struct Multidisc Optim 45:329–357
Vidal CA, Haber RB (1993) Design sensitivity analysis for rate-independent elastoplasticity. Comput Methods Appl Mech Eng 107(3):393–431
Wallin M, Jönsson V, Wingren E (2016) Topology optimization based on finite strain plasticity. Struct Multidisc Optim 54:783–793
Wang F, Lazarov BS, Sigmund O (2011) On projection methods, convergence and robust formulations in topology optimization. Struct Multidisc Optim 43:767–784
Wang C, Zhao Z, Zhou M, Sigmund O, Zhang XS (2021) A comprehensive review of educational articles on structural and multidisciplinary optimization. Struct Multidisc Optim. https://doi.org/10.1007/s00158-021-03050-7
Xia L, Breitkopf P (2015) Design of materials using topology optimization and energy-based homogenization approach in MATLAB. Struct Multidisc Optim 52(6):1229–1241
Xu S, Cai Y, Cheng G (2010) Volume preserving nonlinear density filter based on heaviside functions. Struct Multidisc Optim 41:495–505
Yuge K, Kikuchi N (1995) Optimization of a frame structure subjected to a plastic deformation. Struct Optim 10:197–208
Zhang G, Li L, Khandelwal K (2017) Topology optimization of structures with anisotropic plastic materials using enhanced assumed strain elements. Struct Multidisc Optim 55:1965–1988
Zhao T, Ramos AS Jr, Paulino GH (2019) Material nonlinear topology optimization considering the Von Mises criterion through an asymptotic approach: max strain energy and max load factor formulations. Int J Numer Meth Eng 118(13):804–828
Zhao T, Lages EN, Ramos AS Jr, Paulino GH (2020) Topology optimization considering the Drucker–Prager criterion with a surrogate nonlinear elastic constitutive model. Struct Multidisc Optim 62(6):3205–3227