Honda Research Institute EU
Honda Research Institute Europe GmbH
https://www.honda-ri.de/

Level Set Topology Optimization with Sparse Automatic Differentiation

Andreas Neofytou, Thiago de Jesus de Araujo Rios, Mariusz Bujny, Stefan Menzel, Hyunsun Alicia Kim

2024

Preprint:

This is a post-peer-review, pre-copyedit version of an article published in Structural and Multidisciplinary Optimization. The final authenticated version is available online at: https://doi.org/10.1007/s00158-024-03894-9

Level Set Topology Optimization with Sparse Automatic Differentiation

Andreas Neofytou[a,*], Thiago Rios[b], Mariusz Bujny[c], Stefan Menzel[b], H. Alicia Kim[a]

[a]*Structural Engineering Department, University of California San Diego, CA 92093, USA*
[b]*Honda Research Institute Europe, 63073 Offenbach/Main, Germany*
[c]*NUMETO, 43-190 Mikolow, Poland*

The article has been accepted for publication in Structural and Multidisciplinary Optimization 67, 178 (2024). https://doi.org/10.1007/s00158-024-03894-9.

The final publication is available at Springer via https://doi.org/10.1007/s00158-024-03894-9

Level Set Topology Optimization with Sparse Automatic Differentiation

Andreas Neofytou[a,*], Thiago Rios[b], Mariusz Bujny[c], Stefan Menzel[b], H. Alicia Kim[a]

[a] *Structural Engineering Department, University of California San Diego, CA 92093, USA*
[b] *Honda Research Institute Europe, 63073 Offenbach/Main, Germany*
[c] *NUMETO, 43-190 Mikolow, Poland*

Abstract

Analytical differentiation for a smooth and accurate sensitivity field is typically used for efficient structural and multidisciplinary optimization. However, it can be challenging for multiphysics and non-linear problems. An alternative approach is automatic differentiation. For large problems with many design variables AD can be computationally expensive and memory demanding and thus its use is still limited. To address some of these challenges, we propose to exploit the sparsity of the level set topology optimization in combination with a hybrid mode when using automatic differentiation. The modularized level set topology optimization used here enables the use of automatic differentiation in combination with the classical level set method. In our method, we utilize the operator overloading approach, and we start our development by comparing different operator overloading libraries. Next, a sparse automatic differentiation implementation is proposed to take advantage of the sparsity of the level set method, in which sensitivities are only required within a narrow band close to the level set boundary. The obtained results indicate that this sparsity can improve the efficiency of the implementation. However, the reduction in memory requirements is not as significant. The obtained results indicate that this sparsity can improve the efficiency of the implementation. To improve the memory consumption as well, we introduce a hybrid mode, where instead of computing the total sensitivity directly with operator overloading, the expression is first derived analytically and then operator overloading is used to obtain the partial derivatives. Our studies show that combining the hybrid mode with the sparsity of the level set topology optimization results in improvements in efficiency, with almost an order of magnitude less computational time for the biggest mesh size studied. Finally, the combination of the forward and reverse modes depending on the partial derivatives at hand is exploited to further improve memory requirements and computational cost.

Keywords: Level set topology optimization (LSTO), automatic differentiation (AD), operator overloading (OO), sparsity, hybrid implementation

1. Introduction

The computation of sensitivities in topology optimization can be automated through the use of automatic differentiation (AD), a well-known collection of methods for computing derivatives by applying the chain rule of differentiation to computer programs (Griewank, 2000). The main idea is that differentiable functions can be broken down into a series of simple operations, such as addition, multiplication, and trigonometric functions. Even if the differentiation of the original

*Corresponding author: Andreas Neofytou, anneofytou@ucsd.edu

function may be very complicated, differentiation of the individual operations is straightforward and the chain rule can provide the derivative of the original expression with high accuracy. AD can be advantageous from a practical point of view since it automates the computation of design sensitivities. This enables addressing complex multidisciplinary problems with efficient gradient-based optimizers. The generic character of AD also allows for building modular optimization frameworks and combining them with different methods and tools.

There are two modes of AD: the forward and reverse mode. The forward mode can compute the derivatives as the code runs, while in the reverse mode, the code runs, first, storing intermediate variables and operations in a data structure called the *tape*, and then derivatives are produced by working backwards using this information. The forward mode is more efficient in cases where there are less design variables than objectives whereas the reverse mode is a better choice in problems in which there are more design variables than objectives (Martins and Hwang, 2013). One of the difficulties with the reverse mode is that the data of all the operations that is to be differentiated need to be stored in memory, which increases memory requirements.

AD can be implemented in various ways. These generally fall into two categories, namely, source code transformation (Hascoët and Pascual, 2013) and operator overloading (OO) (Pryce and Reid, 1998). The former method, as indicated by its name, requires the modification of the original solver. The source code transformation tools add new lines to the original solver code to compute the derivatives. In cases where an extensive solver code already exists, this approach is cumbersome and inconvenient. Operator overloading tools, on the other hand, exploit the functionality of modern languages such as C++ to define operators on user-defined data types and can be applied to any code written in a programming language that supports overloading (Peltzer et al., 2020). OO requires little code modification which makes this approach substantially more convenient. The main body of the original solver code does not have to change but some of the variable types need to be redefined. These types include both the original variables and their derivatives. As a result, every operation in the code is overloaded to provide the derivatives of all the operations. Thus, we utilize the OO approach in our AD methodology.

In recent years, interests have emerged in the use of AD in topology optimization (TO). The computation of sensitivities is an essential step in TO. Among the various methods for computing derivatives, analytical methods such as the adjoint approach have been extensively used to compute accurate sensitivities efficiently. However, as the complexity of the problem at hand increases, analytical derivation can be more and more challenging and susceptible to errors. This includes, for example, the case of coupled multiphysics problems or complex multidisciplinary problems where alternatives such as heuristic approaches, computationally costly non-gradient algorithms or finite difference schemes are used to replace the complex analytical derivarions (Aulig et al., 2018; Bujny et al., 2021, 2023). The use of AD can provide a convenient and robust alternative in computing sensitivities.

Laniewski-Wollk and Rokicki (2016) developed a topology optimization framework for a variety of multiphysics fluid flow problems. A discrete adjoint formulation, effective for a wide range of Lattice Boltzmann Methods (LBMs) was proposed. In their work, a source code transformation AD tool, Tapanade (Hascoët and Pascual, 2013), was used to apply reverse mode AD only partially, to differentiate the model's dynamics while enhancing the parallelization of the adjoint code. Norgaard et al. (2017) has illustrated the applicability of reverse mode AD in density-based topology optimization through a series of multiphysics examples including transient wave propagation and the LBM for computing fluid flows. The authors used and compared different AD implementations, including the OO method and source code transformation using the AD libraries CoDiPack and Tapenade. Chandrasekhar et al. (2021) revisited AD in TO using the Just After Execution (JAX) library (Bradbury et al., 2018). The paper provides an open-source implementation for TO with reverse mode AD within JAX, called AuTO, and illustrates its use through 2D examples for compliance minimization, compliant mechanism design, and design of

microstructures.

Dilgen et al. (2018) employed OO AD to perform TO of turbulent flows. Rather than differentiating the entire code using AD, the authors applied AD selectively to compute the partial derivatives used in an analytically derived adjoint formulation instead. This hybrid approach between the analytical formulation and AD has also been proposed by Mader et al. (2008) and Marta et al. (2007) to reduce the memory requirements of the reverse mode AD approach. Similar hybrid approaches have been used by shape optimization works on different physics including aerodynamic design (Gao et al., 2017; Mader et al., 2008), fluid-structure interactions (Sanchez et al., 2017), fluid flow (Ham et al., 2019), and structural problems (Lopez et al., 2021). He and Abueidda (2022) used AD to compute the derivatives in multiphysics TO problems involving thermoelasticity and piezoelectricity. The paper mainly illustrated the applicability of AD in the context of TO through the python-based library JAX (Bradbury et al., 2018).

AD in the form of semi-symbolic differentiation is also used by several TO works together with the FEniCS software (Alnaes et al., 2015). Some examples include the works by Laurain (2018) on level set structural optimization for compliance minimization, Alonso et al. (2021) on steady-state laminar and turbulent flows, and Gupta et al. (2020) on 2D and 3D parallel TO examples of compliance minimization. FEniCS provides the partial derivatives using symbolic differentiation whereas the Dolfin Adjoint package (Mitusch et al., 2019) can be used to assemble the total derivative. In order to do so, the symbolic manipulations are recorded in a structure analogous to an OO tape but at a higher level of abstraction since it stores the entire equation solutions. Alternatively, Yan et al. (2022) assembled the partials obtained through symbolic differentiation in FEniCS into the total derivative using OpenMDAO (Gray et al., 2019). The approach was used to solve compliance minimization considering linear and nonlinear elasticity models and the thermoelastic equation on structured and unstructured meshes. A similar methodology can be used for TO through commercial software such as COMSOL multiphysics (Souza et al., 2021; Picelli et al., 2020). Gangl et al. (2021) presented a framework for automated shape differentiation, combining the mathematical Lagragian approach with AD. Specifically, symbolic differentiation was employed using the finite element software package NGSolve (Schöberl, 2014). Shape derivatives were obtained for different classes of PDE-constrained optimization problems considering examples in nonlinear elasticity and the Maxwell's equations.

The symbolic differentiation approach relies on a high-level symbolic representation of the forward problem in near-mathematical notation. Such implementations provide derivatives very efficiently. However, they are only applicable to finite element implementations and within the specific software and thus they cannot be generalized to arbitrary codes.

For topology optimization problems with many design variables, AD with OO libraries can be computationally expensive and memory demanding and thus its use is still limited. In the present paper we demonstrate how OO can be used more efficiently and with less memory requirements by exploiting the features of level set topology optimization (LSTO). In particular, we propose a framework utilizing a novel hybrid AD implementation that exploits the sparsity of our LSTO, and we illustrate the resulting improvements in efficiency and memory requirements. The modularized LSTO architecture used here maps the element-based sensitivities to compute consistent shape sensitivities. This enables the use of AD to provide the sensitivities of the elements with respect to the design variables. A hybrid implementation between the analytical derivation and OO, combining the forward and reverse modes is also investigated.

The remainder of the paper is organized as follows: Section 2 outlines the modularized LSTO used in this work. In Sec. 3 the incorporation of AD with OO in LSTO is discussed. Section 3.1 provides the implementation details and numerical investigations on different AD tools through the example of stress minimization. Section 3.2 explains how the sparsity of LSTO can be utilized in OO AD and in Sec. 3.3 the sparse implementation is introduced into the hybrid mode that combines analytical derivation and OO. In Sec. 3.4 everything comes together to construct the

unique approach we propose in this work, a hybrid OO AD mode that utilizes the sparsity of LSTO and combines the forward and reverse modes.

2. LSTO Architecture

In this work, we are using the LSTO method studied in the previous literature such as Allaire et al. (2004), Wang et al. (2003) and Dunning and Kim (2015). The LSTO method used in this work has several unique features that enable the use of AD in the context of the classical level set method. In this section, the basic concepts of the method are outlined. For a more in-depth discussion the reader is referred to Jauregui et al. (2023) and Kambampati et al. (2020).

In LSTO the material distribution of the mechanical structure is defined implicitly based on the level set field as

$$\begin{cases} \phi(\mathbf{x}) \geq 0, & \mathbf{x} \in \Omega, \\ \phi(\mathbf{x}) = 0, & \mathbf{x} \in \Gamma, \\ \phi(\mathbf{x}) < 0, & \mathbf{x} \notin \Omega, \end{cases} \tag{1}$$

where $\phi(\mathbf{x})$ is the level set function evaluated at point $\mathbf{x}$, Ω is the domain of the structure, and Γ is the structural boundary. The level set function is updated at each optimization iteration by solving the following level set advection equation (Osher et al., 2004)

$$\frac{d\phi}{dt} + |\nabla\phi(\mathbf{x})| V_n(\mathbf{x}) = 0, \tag{2}$$

which is commonly discretized as

$$\phi_r^{k+1} = \phi_r^k - \Delta t |\nabla\phi_r^k| V_{n,i}, \tag{3}$$

where r is a nodal point on the level set grid, k is the iteration number, and $\nabla\phi$ is the level set function gradient, computed using the Hamilton-Jacobi weighted essentially non-oscillatory scheme (HJ-WENO) (Jiang and Peng, 2000; Sethian, 1999).

The velocity field, V_n, needed to solve (2) is obtained by solving the following linearized optimization problem,

$$\begin{aligned} \underset{V_{n,i}}{min} \quad & \Delta F = \sum_i^{np} \left(\left(\frac{\partial F}{\partial \Omega} \right)_i A_i z_i \right), \\ s.t. \quad & G_j = \sum_i^{np} \left(\left(\frac{\partial G_j}{\partial \Omega} \right)_i A_i z_i \right) \leq g_j^t \quad j = 1, 2, ..., N_g, \\ & -l_i \leq z_i \leq l_i, \end{aligned} \tag{4}$$

where n_p is the number of boundary points, $z_i = V_{n,i}\Delta t$ is the movement of the i^{th} boundary point, the pseudo time step $\Delta t = 1$, A_i is the area of the discretized boundary associated with the i^{th} boundary point, g_j^t is the target constraint value for the j^{th} constraint, N_g is the total number of constraints and l_i is the boundary movement limit. The terms $\frac{\partial F}{\partial \Omega}$ and $\frac{\partial G_j}{\partial \Omega}$ represent the boundary shape sensitivities of the objective function and the j^{th} constraint, respectively. Conventionally, these shape sensitivities are computed based on the continuous adjoint method (Allaire et al., 2004; Wang et al., 2003; Dunning and Kim, 2015; Picelli et al., 2018).

A unique feature of our LSTO method is a procedure that allows for the use of the discrete adjoint method in the context of the traditional LSTO (Kambampati et al., 2020) by mapping

element-based sensitivities onto the boundary as shape sensitivities. This also enables the use of AD considered in this paper. The overall concept is illustrated in Fig. 1.

A TO method can be broken down into three separate modules (Fig. 1): the design update module which includes the optimization algorithm for design update, the state equation solver module in which the governing partial differential equation (PDE) is solved to obtain the sensitivities of the objective F and constraints G_j with respect to the design variables, and a mapping function that transfers the design parameterization onto the analysis mesh for computing the state variables and then returns sensitivities with respect to the design variables to the optimizer.

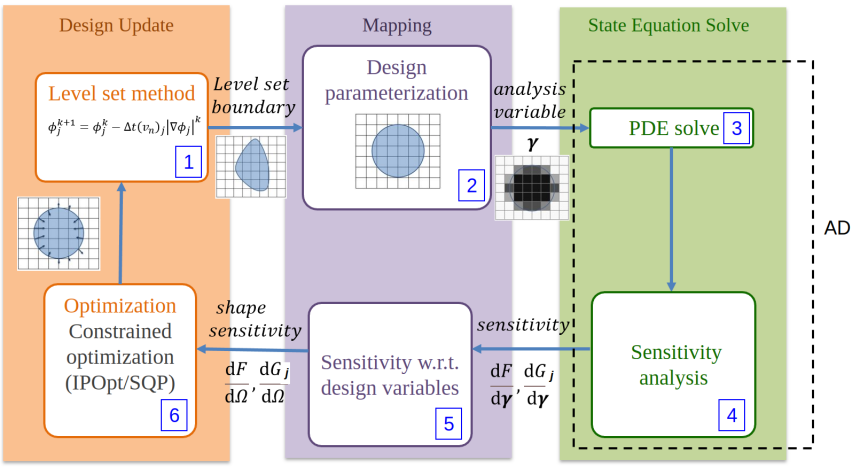


Figure 1: Modularized LSTO architecture.

In the case of density-based TO the mapping is straightforward since the same mesh is used both for the design optimization and the solution of the PDE. This is not the case for LSTO, since the level set field domain does not need to be the same as the analysis mesh. This mapping of the level set field onto the analysis mesh can be done in a number of ways, for example by employing conforming mesh (Allaire et al., 2014) or by using immersed methods such as the extended finite element method (XFEM) (Jenkins and Maute, 2016) and the finite cell method (FCM) (Cai et al., 2014). In the LSTO used here, the level set field (box 1 in Fig. 1) is mapped onto the analysis mesh using the Ersatz material approximation as shown in Figs. 2 (a) and (b). A density field is created on a regular fixed analysis mesh based on the area fractions of the elements, as these are obtained from the location of the boundary and the level set field. These area fractions are computed for each element as the area of the element over which $\phi \geq 0$ divided by the total area of the element. Elements that lie entirely inside the structure ($\phi \geq 0$) are structural elements with a density property $\boldsymbol{\gamma} = 1$, elements that are completely outside ($\phi < 0$) are void and have density values $\boldsymbol{\gamma} = \boldsymbol{\gamma}_{min} << 1$, and elements that are cut by a boundary have intermediate density values $0 \leq \boldsymbol{\gamma} \leq 1$. The density field constructed on the fixed mesh acts as the design variables that map the design onto the analysis mesh, which is one functionality of the mapping (box 2). Now, the material properties are interpolated using the density field and the PDE can be solved (box 3). The element-based sensitivities can be obtained using either the discrete adjoint method, or AD (box 4). Here, any combination of PDE solver and AD tool can be used in a plug-and-play fashion. The mapping function is used again to map these element-based sensitivities onto the boundary as shape sensitivities, as shown in Fig. 2 (c)-(d). To compute the shape sensitivity for a given boundary point i, a small perturbation is assigned

to it as shown in Fig. 2 (c) and (d). This perturbation changes the densities of the elements near this boundary point and the density of each element, m, can be associated with the boundary movement as

$$\frac{\partial \gamma_m}{\partial z_i} \approx \frac{\gamma_m - \gamma_m^\delta}{\delta}, \tag{5}$$

where γ_m is the density of the m^{th} element before the perturbation, γ_m^δ is the density of the m^{th} element after the perturbation, and δ is the magnitude of the perturbation. Here, the step size is computed based on the grid size as $10^{-4}\Delta x$, where Δx is the width of one level set grid element, which has been demonstrated to provide accurate results in Kambampati et al. (2020). The element-based sensitivities can be converted into the boundary shape sensitivities, $(\frac{\partial F}{\partial \Omega})_i$ and $(\frac{\partial G_j}{\partial \Omega})_i$ in (4) through the following relations

$$\left(\frac{\partial F}{\partial \Omega}\right)_i = \frac{\partial F}{\partial z_i} = \sum_m^{N_e} \frac{\partial F}{\partial \gamma_m}\frac{\partial \gamma_m}{\partial z_i}, \tag{6}$$

$$\left(\frac{\partial G_j}{\partial \Omega}\right)_i = \frac{\partial G_j}{\partial z_i} = \sum_m^{N_e} \frac{\partial G_j}{\partial \gamma_m}\frac{\partial \gamma_m}{\partial z_i}, \tag{7}$$

where N_e is the number of elements whose densities are affected by the perturbation. Further details can be found in Kambampati et al. (2020).

The shape sensitivities are then fed into the optimization algorithm of choice (box 6) to solve the optimization problem in (4), obtain the velocity field V_n, and update the level set function using (3) (box 1).

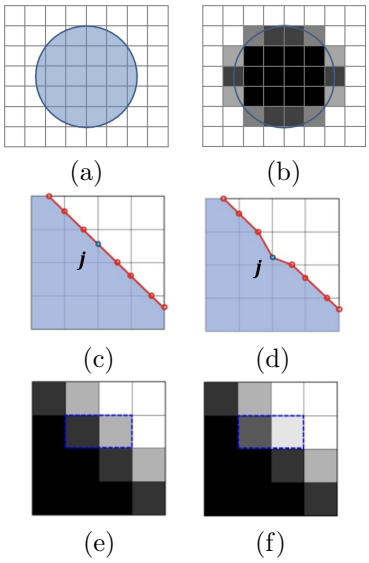


Figure 2: Perturbation scheme to compute shape sensitivities at the boundary points based on neighbouring element sensitivities: The level set boundary in (a) is mapped on the analysis domain through the density field in (b). To compute shape sensitivities, each boundary point j is perturbed as shown in (c) and (d). These boundary perturbations are associated with changes in the density field as shown in (e) and (f).

In the case of a general TO problem, the implementation is exactly the same. The OO tool needs to track and record the part of the code in which the dependency on the design variables

appears, i.e., the part where densities are multiplied by the stiffness matrix and the solution of the resulting governing equations.

3. Level Set Topology Optimization With Operator Overloading

In this section, OO is discussed in the context of the modularized LSTO explained in Sec. 3. Different implementations are investigated in terms of computational time and number of operations that need to be recorded by OO.

3.1. Implementation Details

As shown in Fig. 1, the AD enters the overall LSTO implementation at the point where the density field is passed to the PDE solver. The overall implementation is illustrated by the pseudo code in Algorithm 1.

Algorithm 1 LSTO with OO

$i = 0$ ▷ Iteration index
$\phi = \phi_0$ ▷ Initialize level set function
while $\Delta F \geq tolerance \quad and \quad i < MaxIter$ **do**
  area fractions ← LSTO boundary ▷ Discretize level set boundary
  Begin AD tape
  $\boldsymbol{\gamma}_{OO}[N_e] \leftarrow Initialize$ ▷ $\boldsymbol{\gamma_{OO}}$ = OO active variable vector, N_e = Number of elements
  $\boldsymbol{\gamma}$ = area fraction ▷ Density field based on the element area fractions
  $i \leftarrow i + 1$
  for $m \leftarrow 0$ to N_e **do** ▷ Loop through the number of elements, N_e
    $\gamma_{OO,m} \leftarrow \gamma_m$
    $\mathbf{K}_{e,m} = (\gamma_{min} + \gamma_{OO,m}(1 - \gamma_{min}))\mathbf{K}_{e,0}$ ▷ Interpolation based on (14)
    $\mathbf{K} \leftarrow assemble, \mathbf{K}_{e,0}$ ▷ Assemble the total stiffness matrix, (15)
  end for
  $\mathbf{Ku} = \mathbf{F}$ ▷ Solve governing equation, (13)
  objective ← $\sigma_{pn}(\boldsymbol{\gamma}_{OO}, \mathbf{u})$ ▷ Compute Objective
  End AD tape
  $\frac{d\sigma_{pn}}{d\boldsymbol{\gamma}} \leftarrow$ Reverse mode OO ▷ Total derivative directly obtained by reverse mode OO
  $\frac{\partial\sigma_{pn}}{\partial\Omega} \leftarrow \frac{d\sigma_{pn}}{d\boldsymbol{\gamma}}\frac{\partial\boldsymbol{\gamma}}{\partial\mathbf{z}}$ ▷ Compute shape sensitivities using perturbation mapping (6)
  $V_{n,i} \leftarrow \frac{\partial\sigma_{pn}}{\partial\Omega}$ ▷ Solve the linearized optimization in (4) problem for the velocities, $V_{n,i}$
  $\phi = \phi_{new}$ ▷ Update the level set function using (3)
end while

For our numerical investigation, we choose to minimize p-norm stress in the L-shaped beam problem depicted in Fig. 3 (Le et al., 2010; Picelli et al., 2018), as stress sensitivities are often considered more challenging than standard benchmark problems targeting compliance minimization under a volume constraint. The findings in this manuscript are applicable to other physics or governing equations. Our LSTO applied to other physics have been demonstrated for example by Kambampati et al. (2020) in fluid-thermo-mechnical optimization and Hyun et al. (2022) in transient acoustic problems. Here, we incorporate the use of AD in the approach. The objective of interest is p-norm stress,

$$\sigma_{pn} = \left(\int_{\Omega} \sigma_{vm}^{p} d\Omega \right)^{(1/p)}, \tag{8}$$

which can be discretized as

$$\sigma_{pn,d} = \left(\sum_{k=1}^{N_e} \sum_{l=1}^{N_g} (\sigma_{vm,kl})^p \right)^{(1/p)}, \tag{9}$$

where $\sigma_{pn,d}$ is the discretized $p-$norm stress, N_e is the number of elements, N_g is the number of Gauss integration points, and $\sigma_{vm,kl}$ is the von Mises stress at the l-th Gauss integration point of the k-th element given by (Holmberg et al., 2013)

$$\sigma_{vm,kl} = \sqrt{\gamma_k \left((\mathbf{C}\mathbf{B}_l\mathbf{u}_k)^T \mathbf{V} (\mathbf{C}\mathbf{B}_l\mathbf{u}_k) \right)}, \tag{10}$$

where B_l is the strain-displacement matrix at the l-th Gauss integration point, $\mathbf{u}_i$ are the nodal displacement values of the k-th element, $\mathbf{C}$ is the elasticity tensor and $\mathbf{V}$ is the Voight matrix expressed as

$$\begin{bmatrix} 1 & -0.5 & 0 \\ -0.5 & 1 & 0 \\ 0 & 0 & 1.5 \end{bmatrix}. \tag{11}$$

The optimization problem is formulated as

$$\begin{aligned} &minimize && \sigma_{pn,d} \\ &s.t. && Volume \leq \bar{V}, \end{aligned} \tag{12}$$

where $\bar{V}$ is the volume constraint value. The discretized linear elasticity equation can then be expressed as

$$\mathbf{K}\mathbf{u} = \mathbf{f}, \tag{13}$$

where $\mathbf{u}$ is the displacement vector and $\mathbf{f}$ is the load vector. The elemental stiffness matrix is obtained through the following material interpolation formulation,

$$\mathbf{K}_{e,m} = (\gamma_{min} + \gamma_m(1 - \gamma_{min}))\mathbf{K}_{e,0}, \tag{14}$$

where $K_{e,0}$ is the stiffness matrix of an element with density $\gamma = 1$.
The total system matrix is determined as

$$\mathbf{K} = \sum_{m}^{N_e} \mathbf{K}_{e,m}, \tag{15}$$

Figure 4 shows our comparison of AD tools in terms of the time spent for the gradient computation versus the time spent for the solution of the forward problem for different mesh sizes. The computational time is defined as the average time per iteration. The forward problem was solved using the iterative conjugate gradient solver in the C++ linear algebra library Eigen Guennebaud et al. (2010), with the convergence criterion set to 10^{-6}. Here, two popular C++ OO libraries, namely, ADOLC (Walther and Griewank, 2012) and CoDiPack (Sagebaum et al., 2019) are tested. In addition, the Python library Dolfin Adjoint (Mitusch et al., 2019) which uses semi-symbolic AD is used in the comparison for two reasons: The first is to illustrate how the modularized LSTO approach allows for the incorporation of different AD tools, and the second is to investigate the efficiencies of different AD tools. The computational time for the analytical derivation using the discrete adjoint method is also provided as the baseline for the comparison,

since this approach can offer the best efficiency that we would ideally aim to reach with these AD tools.

As can be seen in Fig. 4, the symbolic differentiation of Dolfin Adjoint provides the best efficiency, close to the analytical approach, whereas the CoDiPack and ADOLC are much less efficient. Furthermore, in terms of scalability the OO computational time increase rises quickly as the mesh size increases.

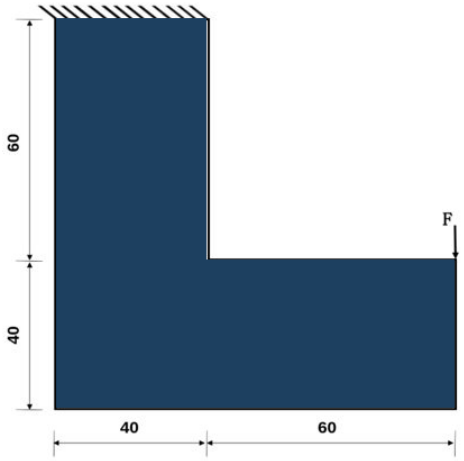


Figure 3: L-Beam test case.

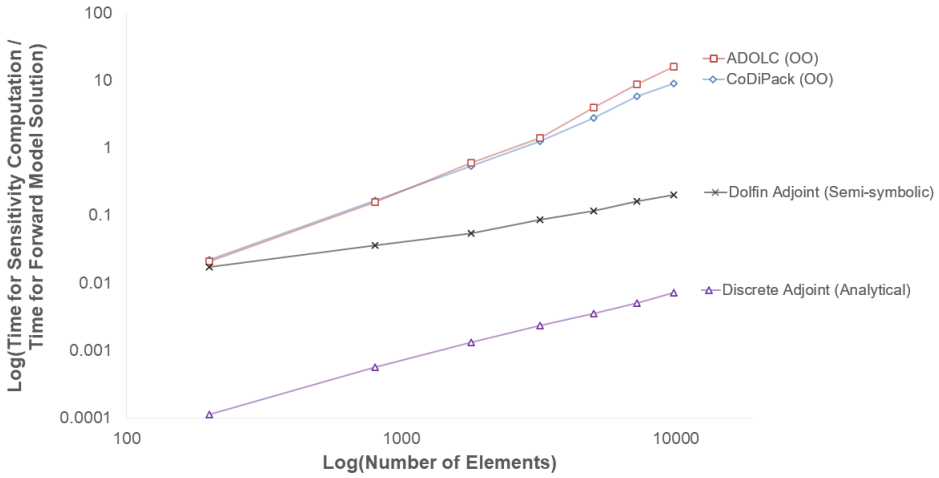
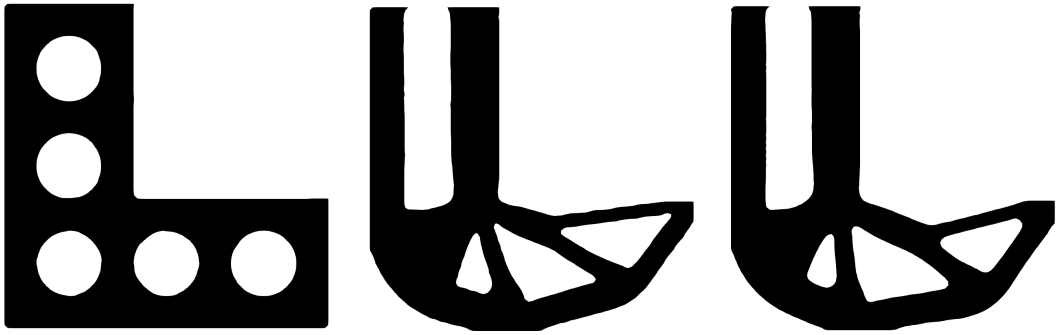


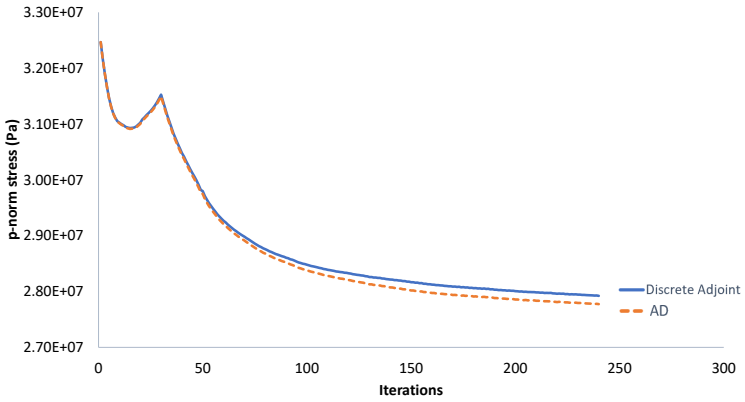
Figure 4: Comparison of the average computational time for one iteration between different AD libraries.

The topologies obtained with the analytical discrete adjoint approach and with AD using CoDiPack are provided in Fig. 5. The volume constraint is set to $\bar{V} = 50\%$ of the L-shaped domain. The objective histories of the two approaches are as shown in Fig. 5 (d). The histogram

of the L_2 norm of the difference in the element sensitivities between the two methods using the finest mesh of 1×10^4 elements is shown in Fig. 6, where the convergence criterion of the linear solver was set to 1×10^{-6}. The difference has the maximum L_2 norm of 3.0×10^{-5}. While we have not been able to determine the convergence criterion of the CoDiPack's package, we believe the difference can be explained by the different convergence criteria used by the two methods. The objective values indicate a 0.5% difference in their structural performance.

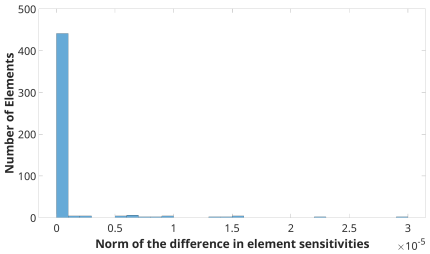


(a) (b) (c)



(d)

Figure 5: Optimized designs obtained with the initial hole configuration in (a) via: (b) The analytical discrete adjoint method and (c) AD with CoDiPack. The objective histories are provided in (d).



(a) (b)

Figure 6: Norm of the difference between the element sensitivities obtained with the discrete adjoint and AD using CoDiPack for the L-shaped beam.

3.2. Sparse OO

As explained in Sec. 2, only the sensitivities of the elements whose densities are affected by the boundary point perturbation take part in the computation of the final shape sensitivities. This typically includes elements that are already cut by the boundary and elements that are close to the boundary such that the perturbation cuts the previously uncut elements, for example if the boundary happens to lie at one of the nodes of the element. We exploit this sparsity to improve the computational time and memory consumption of an AD implementation for TO. This idea is shown in Fig. 7. Since only the sensitivities of elements near the boundary are needed, in the OO implementation we assign as active variables only the densities for these elements. From here on, these elements are referred to as boundary sensitivity elements (BSE). This reduces the number of variables for which the OO AD library needs to track operations and differentiate. Here we study the AD application in the context of TO. In a more general sense, the scaling of the various AD libraries has been studied for a wide range of dimensions (which include mesh density). For example Carpenter et al. (2015) presented a comparative study of different C++ libraries for up to 2^{12} variables. As the number of variables increases, the ratio of the boundary elements to the volume elements tends to decrease (Museth, 2013). Thus, an individual library will benefit from the complexity reduction of the sparse AD method. As an analogy, Liu et al. (2018) illustrated the significant computational efficiency and memory improvements when employing sparse schemes for the solution of the linear elasticity equations in a density-based, compliance minimization TO with the analytical sensitivity formulation.

The sparse implementation is illustrated in Algorithm 2. The main difference with the implementation in Fig. 1 is in lines 10-12, where only the design variables, i.e., densities $\boldsymbol{\gamma}$, of the BSE are assigned as active variables for the OO library, denoted by $\boldsymbol{\gamma_{OO}}$, to track and record. Thus, the size of the active variable vector is equal to the number of only the BSE, not all the elements in the mesh. The tape records all the information until the solution of the linear system and the computation of the objective in lines 17-18, after which the total sensitivity can be computed.

Figure 8 illustrates a comparison of the average time per iteration between the full-mesh implementation in Sec. 3.1 and the current, sparse implementation using the CoDiPack library. As can be seen, efficiency is gained by employing the sparse implementation and this is more evident as the mesh size increases (up to 1.8 times faster).

However, when we look at the number of operations stored in the tape in Fig. 9, we can observe that the difference between the two implementations is not substantial. Both implementations still have to record the part of the code in which the linear system is solved, i.e., (13), which as shown later in Sec. 3.3 is the most significant contribution to the number of operations.

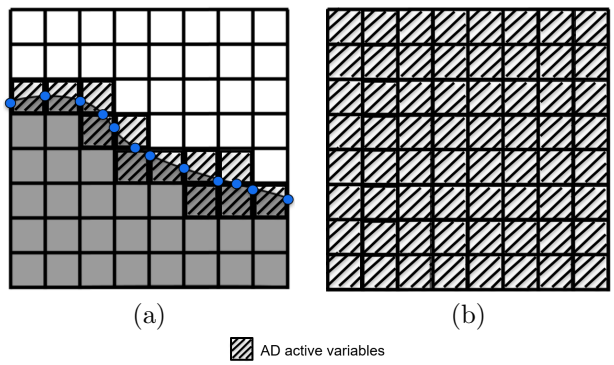


Figure 7: (a) Sparse LSTO OO implementation and (b) full OO implementation.

Algorithm 2 LSTO with Sparse OO

$i = 0$ ▷ Iteration index
$\phi = \phi_0$ ▷ Initialize level set function
while $\Delta F \geq tolerance \quad and \quad i < MaxIter$ **do**
  area fractions ← LSTO boundary ▷ Discretize level set boundary
  Begin AD tape
  $\boldsymbol{\gamma}_{OO}[N_{BSE}] \leftarrow Initialize$ ▷ $\boldsymbol{\gamma_{OO}}$ = OO active variable vector, N_{BSE} = Number of BSE
  $\boldsymbol{\gamma}$ = area fraction ▷ Density field based on the element area fractions
  $i \leftarrow i + 1$
  for $m \leftarrow 0$ to N_e **do** ▷ Loop through the number of elements, N_e
    if m ∈ BSE **then** ▷ if element is a boundary sensitivity element
      $\boldsymbol{\gamma}_{OO,m} \leftarrow \gamma_m$
    end if
    $\mathbf{K}_{e,m} = (\gamma_{min} + \gamma_{OO,m}(1 - \gamma_{min}))\mathbf{K}_{e,0}$ ▷ Interpolation based on (14)
    $\mathbf{K} \leftarrow assemble, \mathbf{K}_{e,0}$ ▷ Assmeble the total stiffness matrix, (15)
  end for
  $\mathbf{Ku} = \mathbf{F}$ ▷ Solve governing equation, (13)
  objective ← $\sigma_{pn}(\boldsymbol{\gamma}_{OO}, \mathbf{u}_{OO})$ ▷ Compute Objective
  End AD tape
  $\frac{d\sigma_{pn}}{d\boldsymbol{\gamma}} \leftarrow$ Reverse mode OO ▷ Total derivative directly obtained by reverse mode OO
  $\frac{\partial\sigma_{pn}}{\partial\Omega} \leftarrow \frac{d\sigma_{pn}}{d\boldsymbol{\gamma}}\frac{\partial\boldsymbol{\gamma}}{\partial\mathbf{z}}$ ▷ Compute shape sensitivities using perturbation mapping (6)
  $V_{n,i} \leftarrow \frac{\partial\sigma_{pn}}{\partial\Omega}$ ▷ Solve the linearized optimization in (4) problem for the velocities, $V_{n,i}$
  $\phi = \phi_{new}$ ▷ Update the level set function using (3)
end while

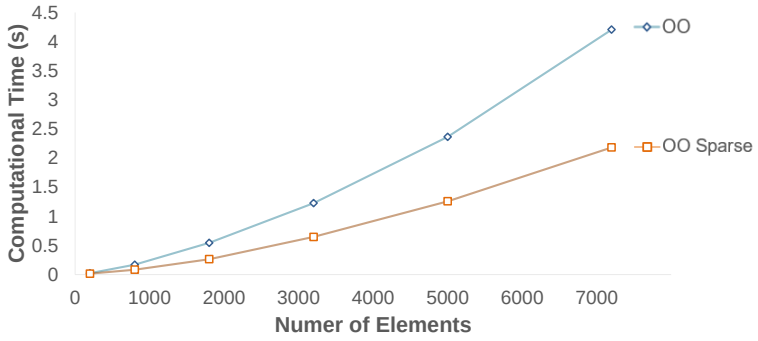


Figure 8: Comparison of the average computational time for one iteration between the OO sparse implementation and original OO using Codipack.

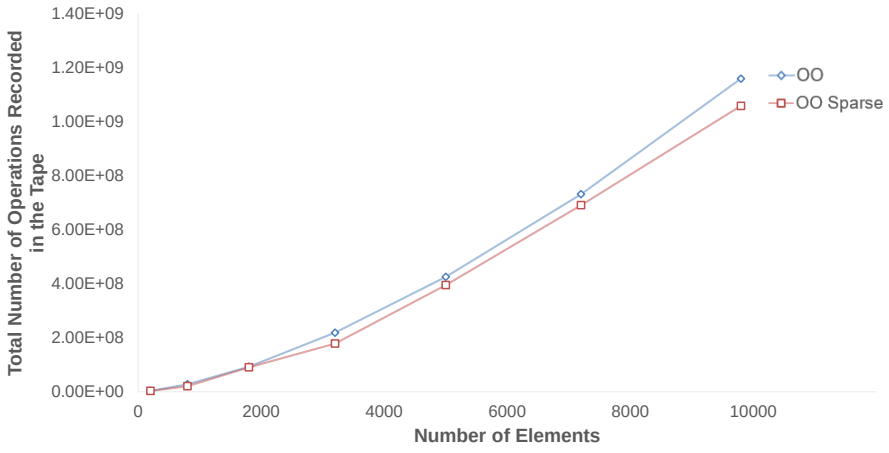


Figure 9: Number of operations stored in the tape: comparison between sparse OO and OO using ADOLC.

3.3. Hybrid OO

The concept of the hybrid mode has been explored for shape optimization, for example by Mader et al. (2008), where the authors illustrated the significant gains in memory consumption. Here, we exploit the sparsity of the level set method to make this approach even more efficient and memory-effective.

Instead of using AD to obtain the total derivative of the objective function with respect to the design variables, the total expression is analytically derived and the AD is used only selectively within the code to obtain the partials. In the case of the p-norm objective for example, the total derivative for the p-norm stress objective can be expressed as

$$\frac{d\sigma_{pn}}{d\boldsymbol{\gamma}} = \frac{\partial\sigma_{pn}}{\partial\boldsymbol{\gamma}} + \frac{\partial\sigma_{pn}}{\partial\mathbf{u}}\frac{\partial\mathbf{u}}{\partial\boldsymbol{\gamma}}. \tag{16}$$

The unknown implicit term, $\frac{\partial\mathbf{u}}{\partial\boldsymbol{\gamma}}$ can be obtained by the total derivative of the residual vector, $\mathbf{R}(\mathbf{u}(\boldsymbol{\gamma}),\boldsymbol{\gamma}) = \mathbf{K}\mathbf{u} - \mathbf{F} = 0$, with respect to the design variable vector, $\boldsymbol{\gamma}$

$$\frac{d\mathbf{R}}{d\boldsymbol{\gamma}} = \frac{\partial\mathbf{R}}{\partial\boldsymbol{\gamma}} + \frac{\partial\mathbf{R}}{\partial\mathbf{u}}\frac{\partial\mathbf{u}}{\partial\boldsymbol{\gamma}} = \mathbf{0} \implies \frac{\partial\mathbf{u}}{\partial\boldsymbol{\gamma}} = -\frac{\partial\mathbf{R}}{\partial\boldsymbol{\gamma}}\left[\frac{\partial\mathbf{R}}{\partial\mathbf{u}}\right]^{-1}. \tag{17}$$

Inserting (17) into (16) the total objective sensitivity can be obtained as

$$\frac{d\sigma_{pn}}{d\boldsymbol{\gamma}} = \frac{\partial\sigma_{pn}}{\partial\boldsymbol{\gamma}} - \boldsymbol{\lambda}^T\frac{\partial\mathbf{R}}{\partial\boldsymbol{\gamma}}, \tag{18}$$

in which $\boldsymbol{\lambda}$ is the adjoint variable vector computed by solving the the adjoint linear system

$$\frac{\partial\mathbf{R}}{\partial\mathbf{u}}\boldsymbol{\lambda} = \frac{\partial\sigma_{pn}}{\partial\mathbf{u}}, \tag{19}$$

AD is then used to compute the Jacobian matrices given by the partials $\frac{\partial\mathbf{R}}{\partial\boldsymbol{\gamma}}$ and $\frac{\partial\mathbf{R}}{\partial\mathbf{u}}$ and the vectors $\frac{\partial\sigma_{pn}}{\partial\boldsymbol{\gamma}}$ and $\frac{\partial\sigma_{pn}}{d\mathbf{u}}$. Once the partials are obtained, (18) and (19) can be solved to obtain the total sensitivity vector $\frac{d\sigma_{pn}}{d\boldsymbol{\gamma}}$. The solution of the adjoint system in (19) requires the full length of the partials $\frac{\partial\mathbf{R}}{\partial\mathbf{u}}$ and $\frac{\partial\sigma_{pn}}{d\mathbf{u}}$, and so the sparse scheme is not applied to these terms. However, for the partials $\frac{\partial\mathbf{R}}{\partial\boldsymbol{\gamma}}$ and $\frac{\partial\sigma_{pn}}{\partial\boldsymbol{\gamma}}$ the sparsity scheme can be employed to improve efficiency.

In Fig. 10, the number of operations stored in the ADOLC tape (an option provided by the ADOLC library) for the original OO implementation, the sparse OO, and the hybrid OO are given. As can be seen, there is a significant reduction in the storage of operations when the hybrid mode is used.

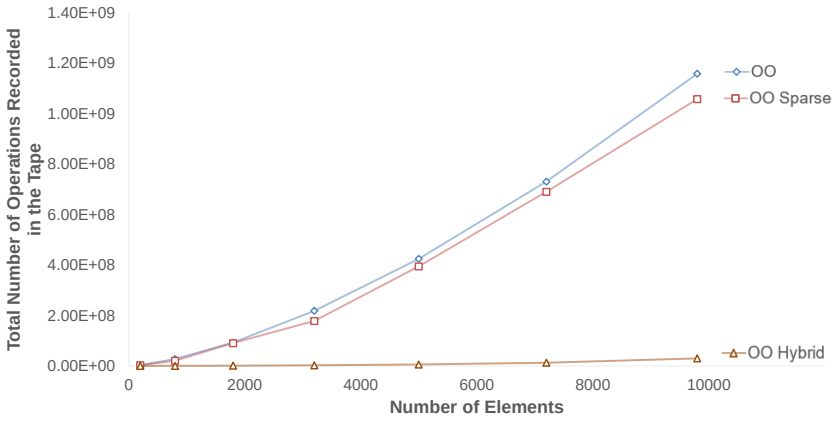


Figure 10: Total number of operations recorded in ADOLC tape for OO, Sparse OO and Hybrid OO.

To explain this difference in the number of operations stored, the overall scheme is shown in Algorithm 3. The OO process needs to track different parts in the code in order to provide the partials. To do this, two separate tapes are used as shown in lines 5 and 18 in Algorithm 3. The first tape records the dependency of the residual on the design variables, i.e., the densities $\boldsymbol{\gamma}$. Since in the residual expression, $\mathbf{R}(\mathbf{u}(\boldsymbol{\gamma}), \boldsymbol{\gamma}) = \mathbf{K}\mathbf{u} - \mathbf{F} = 0$, $\mathbf{F}$ does not depend on $\boldsymbol{\gamma}$, the partial reduces to $\frac{\partial \mathbf{K}}{\partial \boldsymbol{\gamma}}\mathbf{u}$. It is important to note that this formulation would be different in the case for example of a problem with design-dependent loads. As with the implementation in Sec. 3.2, only the densities of the BSE are assigned as the active variables $\boldsymbol{\gamma}_{OO}$ (lines 10-12). Thus again, the size of the active variable vector reduces from all the finite elements down to the number of the BSE (line 6). The dependency of the total stiffness matrix, $\mathbf{K}$, on $\boldsymbol{\gamma}_{OO}$ is created in lines 13-14 and thus the first tape closes after the assembly process.

The second tape needs to record the dependency of the residual $\mathbf{R}$ and the objective function on the state variables, i.e., the displacements $\mathbf{u}$. Thus, tape 2 begins right after the solution of the linear system and the computation of $\mathbf{u}$ in line 17. The active variables in this instance are the displacements of all the BSE, and so now the active variable vector has a length size equal to the total number of degrees of freedom (DOFs) in the BSE in line 19. Lines 21-23 assign these active variables, $\mathbf{u}_{OO}$. The dependency of the residual and the objective on the state variables is built in lines 25, 26 and tape 2 closes right after.

Eventually, using the information from tapes 1 and 2 all the partials can be computed in lines 28-29. Substituting the partials into (19) and (16) provides the total sensitivity of the p-norm stress with respect to the design variables in line 31.

Putting everything together, Fig. 11 shows the computational time of all the OO implementations using Codipack along with the computational time for the analytical derivation using the discrete adjoint method. As expected, the analytical derivation is the most efficient. The least efficient is actually the hybrid mode implementation. Although the hybrid mode results in big memory savings as indicated by Fig. 10, the additional manual implementation to obtain the partials cancels the efficiency gains. However, when the hybrid mode is combined with the sparsity of LSTO, the resulting implementation is the most efficient of all OO AD implementations.

It is important to clarify, that the comparisons are made based on our specific implementation. For example, there may be more efficient ways to program the hybrid mode in general. However, exploiting the sparsity of the level set method within a specific implementation should improve the efficiency for that implementation.

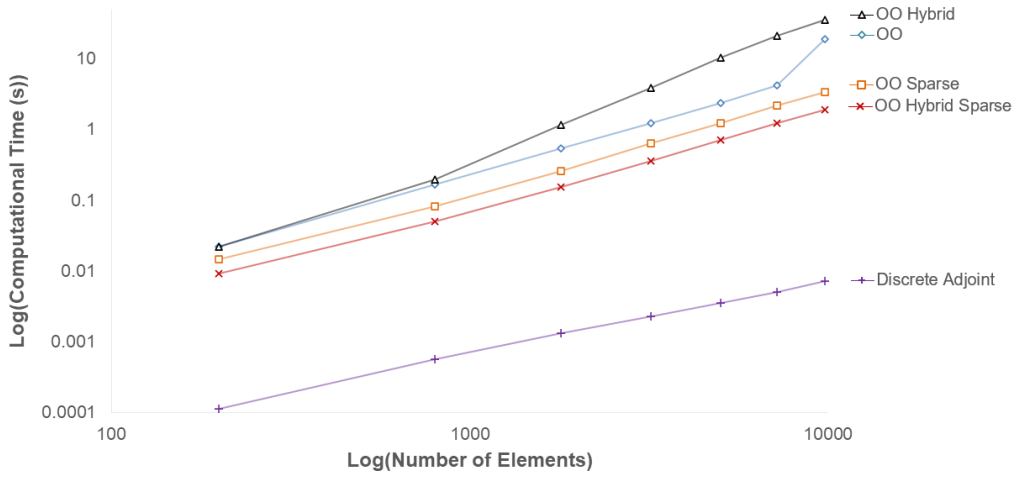


Figure 11: Comparison of the average computational time for one iteration between all the different OO implementations using Codipack.

Algorithm 3 LSTO with Hybrid and Sparse OO

$i = 0$ ▷ Iteration index
$\phi = \phi_0$ ▷ Initialize level set function
while $\Delta F \geq tolerance \quad and \quad i < MaxIter$ **do**
  area fractions ← LSTO boundary ▷ Discretize level set boundary
  Begin OO Tape 1
  $\boldsymbol{\gamma}_{OO}[N_{BSE}] \leftarrow Initialize$ ▷ $\boldsymbol{\gamma}_{\boldsymbol{OO}}$ = OO active variable vector, N_{BSE} = Number of BSE
  $\boldsymbol{\gamma}$ = area fraction ▷ Density field based on the element area fractions
  $i \leftarrow i + 1$
  for $m \leftarrow 0$ to N_e **do** ▷ Loop through the number of elements, N_e
    if m ∈ BSE **then** ▷ if element is a boundary sensitivity element
      $\boldsymbol{\gamma}_{OO,m} \leftarrow \gamma_m$
    end if
    $\mathbf{K}_{e,m} = (\gamma_{min} + \gamma_{OO,m}(1 - \gamma_{min}))\mathbf{K}_{e,0}$ ▷ Interpolation based on (14)
    $\mathbf{K} \leftarrow assemble, \mathbf{K}_{e,0}$ ▷ Assmeble the total stiffness matrix, (15)
  end for
  End Tape 1 ▷ Tape 1: Recorded the dependency on the densities, $\boldsymbol{\gamma}$
  $\mathbf{Ku} = \mathbf{F}$ ▷ Solve governing equation, (13)
  Begin OO Tape 2
  $\mathbf{u}_{OO}[DOF_{BSE}] \leftarrow Initialize$ ▷ $\mathbf{u}_{OO}$ = OO active variable vector, DOF_{BSE} = Number of $DOFs$ in BSE
  for $m \leftarrow 0$ to N_e **do** ▷ Loop through the number of elements, N_e
    $\mathbf{u}_{OO,m} \leftarrow \mathbf{u}_m$ ▷ $\mathbf{u}_m$ represents the vector of displacement $DOFs$ for the $m-$th element
  end for
  $\mathbf{R} = \mathbf{Ku_{OO}} - \mathbf{F}$ ▷ With $\mathbf{u}_{OO}$ known express residual
  objective ← $\sigma_{pn}(\boldsymbol{\gamma}_{OO}, \mathbf{u}_{OO})$ ▷ Compute Objective
  End Tape 2
  $\frac{\partial \mathbf{R}}{\partial \mathbf{u}_{OO}}, \frac{\partial \sigma_{pn}}{\partial \mathbf{u}_{OO}} \leftarrow$ Tape 2
  $\frac{\partial \mathbf{R}}{\partial \boldsymbol{\gamma}_{OO}}, \frac{\partial \sigma_{pn}}{\partial \boldsymbol{\gamma}_{OO}} \leftarrow$ Tape 1
  $\boldsymbol{\lambda}^{\boldsymbol{T}} \leftarrow \frac{\partial \mathbf{R}}{\partial \mathbf{u}_{OO}}, \frac{\partial \sigma_{pn}}{\partial \mathbf{u}_{OO}}$ ▷ Solve adjoint system, (19)
  $\frac{d\sigma_{pn}}{d\boldsymbol{\gamma}_{OO}} \leftarrow \boldsymbol{\lambda}^{\boldsymbol{T}}, \frac{\partial \mathbf{R}}{\partial \boldsymbol{\gamma}_{OO}}, \frac{\partial \sigma_{pn}}{\partial \boldsymbol{\gamma}_{OO}}$ ▷ Compute total derivative, (18)
  $\frac{d\sigma_{pn}}{\partial \Omega} \leftarrow \frac{d\sigma_{pn}}{d\boldsymbol{\gamma}} \frac{\partial \boldsymbol{\gamma}}{\partial \mathbf{z}}$ ▷ Compute shape sensitivities using perturbation mapping (6)
  $V_{n,i} \leftarrow \frac{\partial \sigma_{pn}}{\partial \Omega}$ ▷ Solve the linearized optimization in (4) problem for the velocities, $V_{n,i}$
  $\phi = \phi_{new}$ ▷ Update the level set function using (3)
end while

3.4. Hybrid OO with combination of the forward and reverse modes

In TO, typically the reverse mode is used to obtain the total sensitivity with respect to the design variables (Norgaard et al., 2017; Chaunhan and Martins, 2019). This can be justified by the fact that the number of objectives and constraints is significantly smaller than the number of design variables. However, in the hybrid mode, a combination of forward and reverse modes can be more beneficial. This is a topic discussed by Marta et al. (2007) and Mader et al. (2008) in the context of shape optimization. Here, we investigate this idea in the context of the sparse hybrid LSTO OO implementation.

We start with the partial $\frac{\partial \mathbf{R}}{\partial \boldsymbol{\gamma}}$, where the number of design variables, equal to the number of elements, is actually less than the number of output variables in $\mathbf{R}$ which are equal to the number of degrees of freedom. Thus, using the forward mode for the computation of this partial

is more efficient. Moreover, since the forward mode does not require a tape, there are also gains in terms of memory consumption. For the partial $\frac{\partial \mathbf{R}}{\partial \mathbf{u}}$ the number of active variables, which in this case are the state variables $\mathbf{u}$, either the forward or reverse modes can be utilized in terms of efficiency, however, the forward mode does not require a tape so it is more favorable in terms of memory. The partials $\frac{\partial \sigma_{pn}}{\partial \boldsymbol{\gamma}}$ and $\frac{\partial \sigma_{pn}}{d\mathbf{u}}$ have only one output variable, the p-norm stress function, and thus the reverse mode will be more efficient for their computation.

Again, we employ pseudo-code as shown in Algorithm 4 to illustrate the difference to the implementation in Algorithm 3 where only the reverse mode is used. In most of the code the implementation is the same, with the BSE elements used in lines 9-11 and 18-20 to exploit the sparsity. However, since the reverse mode is only used for the computation of $\frac{\partial \sigma_{pn}}{\partial \boldsymbol{\gamma}_{OO}}$ and $\frac{\partial \sigma_{pn}}{d\mathbf{u}_{OO}}$, a tape is only needed to record the computation of the objective in line 24 where its dependency on $\boldsymbol{\gamma}_{OO}$ and $\mathbf{u}$ appears. The tape provides these partials in line 27. The partials $\frac{\partial \mathbf{R}}{\partial \mathbf{u}_{OO}}, \frac{\partial \mathbf{R}}{\partial \boldsymbol{\gamma}_{OO}}$ are obtained via the forward mode in line 26. The comparison in terms of computational time is provided in Fig. 12.

The combined forward and reverse mode implementation results in a faster computation which may not be substantial for certain problem sizes. However, the reduction in computational time is more evident as the mesh size increases with a gain in efficiency of about 25% for the largest mesh size we studied (about 16000 elements). One of the most important benefits of this implementation is the reduction in memory requirements. In addition to the improved efficiency, all the operations associated with the partials $\frac{\partial \mathbf{R}}{\partial \mathbf{u}_{OO}}, \frac{\partial \mathbf{R}}{\partial \boldsymbol{\gamma}_{OO}}$ do not need to be stored in memory.

Algorithm 4 LSTO with Hybrid and Sparse OO, Combination of Forward and Reverse Modes

$i = 0$ ▷ Iteration index
$\phi = \phi_0$ ▷ Initialize level set function
while $\Delta F \geq tolerance \quad and \quad i < MaxIter$ **do**
  area fractions ← LSTO boundary ▷ Discretize level set boundary
  $\boldsymbol{\gamma}_{OO}[N_{BSE}] \leftarrow Initialize$ ▷ $\boldsymbol{\gamma_{OO}}$ = OO active variable vector, N_{BSE} = Number of BSE
  $\boldsymbol{\gamma}$ = area fraction ▷ Density field based on the element area fractions
  $i \leftarrow i + 1$
  for $m \leftarrow 0$ to N_e **do** ▷ Loop through the number of elements, N_e
    if m ∈ BSE **then** ▷ if element is a boundary sensitivity element
      $\boldsymbol{\gamma}_{OO,m} \leftarrow \gamma_m$
    end if
    $\mathbf{K}_{e,m} = (\gamma_{min} + \gamma_{OO,m}(1 - \gamma_{min}))\mathbf{K}_{e,0}$ ▷ Interpolation based on (14)
    $\mathbf{K} \leftarrow assemble, \mathbf{K}_{e,0}$ ▷ Assmeble the total stiffness matrix, (15)
  end for
  $\mathbf{Ku} = \mathbf{F}$ ▷ Solve governing equation, (13)
  $\mathbf{u}_{OO}[DOF_{BSE}] \leftarrow Initialize$ ▷ $\mathbf{u}_{OO}$ = OO active variable vector, DOF_{BSE} = Number of $DOFs$ in BSE
  for $m \leftarrow 0$ to N_e **do** ▷ Loop through the number of elements, N_e
    $\mathbf{u}_{OO,m} \leftarrow \mathbf{u}_m$ ▷ $\mathbf{u}_m$ represents the vector of displacement $DOFs$ for the $m-$th element
  end for
  $\mathbf{R} = \mathbf{Ku_{OO}} - \mathbf{F}$ ▷ With $\mathbf{u}_{OO}$ known express residual
  Begin OO Tape
  objective ← $\sigma_{pn}(\boldsymbol{\gamma}_{OO}, \mathbf{u}_{OO})$ ▷ Compute Objective
  End Tape
  $\frac{\partial \mathbf{R}}{\partial \mathbf{u}_{OO}}, \frac{\partial \mathbf{R}}{\partial \boldsymbol{\gamma}_{OO}} \leftarrow$ Forward Mode
  $\frac{\partial \sigma_{pn}}{\partial \mathbf{u}_{OO}}, \frac{\partial \sigma_{pn}}{\partial \boldsymbol{\gamma}_{OO}} \leftarrow$ Tape
  $\boldsymbol{\lambda}^{\boldsymbol{T}} \leftarrow \frac{\partial \mathbf{R}}{\partial \mathbf{u}_{OO}}, \frac{\partial \sigma_{pn}}{\partial \mathbf{u}_{OO}}$ ▷ Solve adjoint system, (19)
  $\frac{d\sigma_{pn}}{d\boldsymbol{\gamma}_{OO}} \leftarrow \boldsymbol{\lambda}^{\boldsymbol{T}}, \frac{\partial \mathbf{R}}{\partial \boldsymbol{\gamma}_{OO}}, \frac{\partial \sigma_{pn}}{\partial \boldsymbol{\gamma}_{OO}}$ ▷ Compute total derivative, (18)
  $\frac{d\sigma_{pn}}{\partial \Omega} \leftarrow \frac{d\sigma_{pn}}{d\boldsymbol{\gamma}} \frac{\partial \boldsymbol{\gamma}}{\partial \mathbf{z}}$ ▷ Compute shape sensitivities using perturbation mapping (6)
  $V_{n,i} \leftarrow \frac{\partial \sigma_{pn}}{\partial \Omega}$ ▷ Solve the linearized optimization in (4) problem for the velocities, $V_{n,i}$
  $\phi = \phi_{new}$ ▷ Update the level set function using (3)
end while

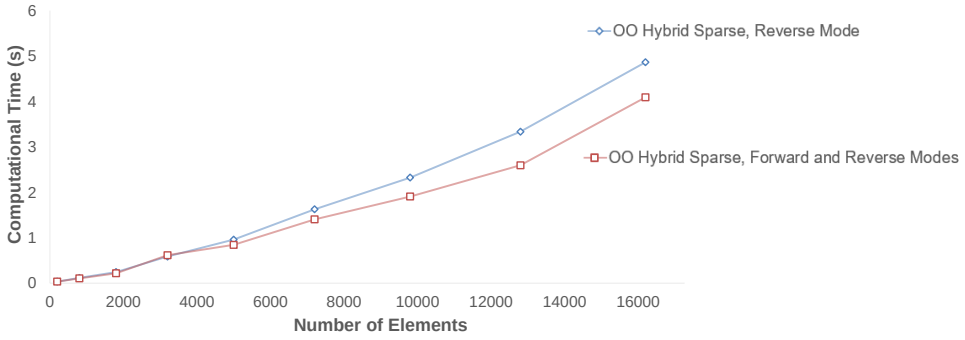


Figure 12: Comparison of the average computational time for one iteration between the OO hybrid sparse reverse mode and the OO hybrid sparse forward/reverse mode using Codipack.

4. Conclusion

In this paper, the use of automatic differentiation (AD), specifically the operator overloading (OO) implementation, is investigated in the context of level set topology optimization (LSTO). The main aim is to reduce the cost and memory consumption of the OO AD for structural and multidisciplinary optimization. To this end, we propose exploiting the sparsity of the LSTO in combination with a hybrid AD approach. We first show that the efficiency can be improved when sparsity is taken into consideration for computing the total sensitivity through the reverse mode of OO AD. However, the reduction in the number of operations recorded in the tape is not as significant. Through the investigations we conclude that more significant improvements in both storage requirements and efficiency can be obtained by employing a sparse, hybrid OO implementation where the total sensitivity expression is derived analytically and OO is used to obtain the partials considering the LSTO sparsity. Finally, we propose the hybrid, sparse OO framework, in which the forward and reverse modes can be combined and used selectively depending on each partial for further improvements. Future directions of the current research, include the application of these implementations in coupled multiphysics topology optimization problems such as fluid-structure interactions.

Acknowledgements Andreas Neofytou and H. Alicia Kim gratefully acknowledge the financial support from Honda Research Institute Europe (HRI-EU).

Competing Interests The authors declare that they have no competing interests as defined by Springer, or other interests that might be perceived to influence the results and/or discussion reported in this paper.

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

Replication of Results OpenLSTO (Kambampati et al., 2018), freely available at https://github.com/M2DOLab/OpenLSTO, was used to obtain the results above. The modifications needed to reproduce all the results are provided in pseudo-code in each section. The automatic differentiation was performed using the open-source libraries CoDiPack (Sagebaum et al., 2019) and ADOL-C (Walther and Griewank, 2012). The solution to the sub-optimization problem was computed using the open-source library GLPK available at gnu.org/software/glpk/ .

References

Allaire, G., Dapogny, C., Frey, P., 2014. Shape optimization with a level set based mesh evolution method. Computer Methods in Applied Mechanics and Engineering 282, 22–53.

Allaire, G., Jouve, F., Toader, A.M., 2004. Structural optimization using sensitivity analysis and a level-set method. Journal of Computational Physics 194, 363393.

Alnaes, M.S., Blechta, J., Hake, J., Johansson, A., Kehlet, B., Logg, A., Richardson, C., Ring, J., Rognes, M.E., Wells, G.N., 2015. The fenics project version 1.5. Archive of Numerical Software 3 .

Alonso, D.H., Rodriguez, L.F.G., Silva, E.C.N., 2021. Flexible framework for fluid topology optimization with openfoam and finite element-based high-level discrete adjoint method (fenics/dolfinadjoint). Structural and Multidisciplinary Optimization 64, 4409–4440.

Aulig, N., Nutwell, E., Menzel, S., Detwiler, D., 2018. Preference-based topology optimization for vehicle concept design with concurrent static and crash load cases. Structural and Multidisciplinary Optimization 57, 251–266.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., Zhang, Q., 2018. Jax: composable transformations of python+numpy programs. URL http://github.com/google/jax .

Bujny, M., Hung, L., Zurbrugg, N., Detwiler, D., Menzel, S., 2023. Method for structural optimization of a design and cost of a physical object. U.S. Patent Application , 17/557,052.

Bujny, M., Olhofer, M., Aulig, N., Duddeck, F., 2021. Topology optimization of 3d-printed joints under crash loads using evolutionary algorithms. Structural and Multidisciplinary Optimization 64, 6.

Cai, S., Zhang, W., Zhu, J., Gao, T., 2014. Stress constrained shape and topology optimization with fixed mesh: A b-spline finite cell method combined with level set function. Computer Methods in Applied Mechanics and Engineering 278, 361–387.

Carpenter, B., Hoffman, M.D., Brubaker, M., Lee, D., Li, P., Betancourt, M., 2015. The stan math library: Reverse-mode automatic differentiation in c++. arXiv:1509.07164v1 [cs.MS] .

Chandrasekhar, A., Sridhara, S., Suresh, K., 2021. Auto: a framework for automatic differentiation in topology optimization. Structural and Multidisciplinary Optimization 64, 43554365.

Chaunhan, S.S., Martins, J.R.R.A., 2019. Tilt-wing evtol takeoff trajectory optimization. Journal of Aircraft 57, 1–20.

Dilgen, C.B., Dilgen, S.B., Fuhrman, D.R., Sigmund, O., Lazarov, B.S., 2018. Topology optimization of turbulent flows. Computer Methods in Applied Mechanics and Engineering 331, 363–393.

Dunning, P.D., Kim, H.A., 2015. Introducing the sequential linear programming level-set methodfor topology optimization. Structural and Multidisciplinary Optimization 51.

Gangl, P., Sturm, K., Neunteufel, M., Schoberl, J., 2021. Fully and semi-automated shape differentiation in ngsolve. Structural and Multidisciplinary Optimization 63. doi:http://doi.org/10.1007/s00158-020-02742-w.

Gao, Y., Wu, Y., Xia, J., 2017. Automatic differentiation based discrete adjoint method for aerodynamic design optimization on unstructured meshes. Chinese Journal of Aeronautics 30, 611–627.

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

Griewank, A., 2000. Evaluating derivatives: Principles and techniques of algorithmic differentiation. Frontiers in Appl. Math., SIAM, Philadelphia, PA 19, 370.

Guennebaud, G., Jacob, B., et al., 2010. Eigen v3. http://eigen.tuxfamily.orghow do.

Gupta, A., Chowdhury, R., Chakrabarti, A., Rabczukb, T., 2020. A 55-line code for large-scale parallel topology optimization in 2d and 3d. arXiv:2012.08208 .

Ham, D.A., Mitchell, L., Paganini, A., Wechsung, F., 2019. Automated shape differentiation in the unified form language. Structural and Multidisciplinary Optimization 60, 1813–1829.

Hascoët, L., Pascual, V., 2013. The tapenade automatic differentiation tool: principles, model, and specification. ACM Transactions Mathematical Software 39.

He, J., Abueidda, D., 2022. Multiphysics topology optimization using automatic differentiation and adjoint method. Preprints.org 2022, 2022100071 .

Holmberg, E., Torstenfelt, B., Klarbring, A., 2013. Stress constrained topology optimization. Structural and Multidisciplinary Optimization 48, 33–47.

Hyun, J., Jauregui, C., Neofytou, A., Kim, H.A., 2022. On development of an accessible and nonintrusive level-set topology optimization framework via the discrete adjoint method. AIAA SciTech, January 3-7, San Diego, CA & Virtual .

Jauregui, C.M., Hyun, J., Neofytou, A., Gray, J.S., Kim, H.A., 2023. Avoiding reinventing the wheel: reusable opensource topology optimization software. Structural and Multidisciplinary Optimization 66.

Jenkins, N., Maute, K., 2016. An immersed boundary approach for shape and topology optimization of stationary fluid-structure interaction problems. Structural and Multidisciplinary Optimization 54, 1191–1208.

Jiang, G.S., Peng, D., 2000. Weighted eno schemes for Hamilton Jacobi equations. Society for Industrial and Applied Mathematics 21, 2126–2143.

Kambampati, S., Chung, H., Kim, H.A., 2020. A discrete adjoint based level set topology optimization method for stress constraints. Computer Methods in Applied Mechanics and Engineering 377, 113563.

Kambampati, S., Du, Z., Chung, H., Kim, H.A., Jauregui, C., Townsend, S., Picelli, R., Zhou, X.Y., Hedges, L., 2018. Openlsto: open-source software for level set topology optimization. Multidisciplinary Analysis and Optimization Conference doi:https://doi.org/10.2514/6.2018-3882.

Laniewski-Wollk, L., Rokicki, J., 2016. Adjoint Lattice Boltzmann for topology optimization on multi-gpu architecture. Computers and Mathematics with Applications 71, 833–848.

Laurain, A., 2018. A level set-based structural optimization code using fenics. Structural and Multidisciplinary Optimization .

Le, .C., Norato, J., Bruns, T., Ha, C., Tortorelli, D., 2010. Stress-based topology optimization for continua. Structural and Multidisciplinary Optimization 41, 605–620.

Liu, H., Hu, Y., Zhu, B., Matusik, W., Sifakis, E., 2018. Narrow-band topology optimization on a sparsely populated grid. ACM Transactions on Graphics (TOG) 37, 1–14.

Lopez, J., Anitescu, C., Rabczuk, T., 2021. Isogeometric structural shape optimization using automatic sensitivity analysis. Applied Mathematical Modelling 89, 1004–1024.

Mader, C.A., Martins, J.R.R.A., Alonso, J.J., van der Weide, E., 2008. Adjoint: An approach for the rapid development of discrete adjoint solvers. AIAA Journal 46.

Marta, A.C., Mader, C.A., Martins, J.R.R.A., der Weide, E.V., Alonso, J.J., 2007. A methodology for the development of discrete adjoint solvers using automatic differentiation tools. International Journal of Computational Fluid Dynamics 21, 307–327.

Martins, J.R.R.A., Hwang, J.T., 2013. Review and unification of methods for computing derivatives of multidisciplinary computational models. AIAA Journal 51.

Mitusch, S.K., Funke, S.W., Dokken, J.S., 2019. dolfin-adjoint 2018.1: automated adjoints for fenics and firedrake. Journal of Open Source Software .

Museth, K.., 2013. Vdb: High-resolution sparse volumes with dynamic topology. ACM Trans Graph 32, 27.

Norgaard, S.A., Sagebaum, M., Gauger, N.R., Lazarov, B.S., 2017. Applications of automatic differentiation in topology optimization. Structural and Multidisciplinary Optimization 56, 1135–1146.

Osher, S., Fedkiw, R., Piechor, K., 2004. Level set methods and dynamic implicit surfaces. Applied Mechanics Reviews 57, B15.

Peltzer, P., Lotz, J., Naumann, U., 2020. Eigen-ad: Algorithmic differentiation of the eigen library. Computational Science - ICCS 2020, Lecture Notes in Computer Science, Springer, Cham 12137.

Picelli, R., Ranjbarzadeh, S., Sivapuram, R., Gioria, R.S., Silva, E.C.N., 2020. Topology optimization of binary structures under design-dependent fluid-structure interaction loads. Structural and Multidisciplinary Optimization 62, 2101–2116.

Picelli, R., Townsend, S., Brampton, C., Norato, J., Kim, H., 2018. Stress-based shape and topology optimization with the level set method. Computer Methods in Applied Mechanics and Engineering 329, 1–23.

Pryce, J.D., Reid, J.K., 1998. Ad01, a Fortran 90 code for automatic differentiation. Report RAL-TR-1998-057, Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire, OX11 OQX, U.K .

Sagebaum, M., Albring, T., Gauger, N.R., 2019. High-performance derivative computations using codipack. Article in ACM Transactions on Mathematical Software, Association for Computing Machinery .

Sanchez, R., Albring, T., Palacios, R., Gauger, N.R., Economon, T.D., Alonso, J.J., 2017. Coupled adjoint-based sensitivities in large-displacement fluid-structure interaction using algorithmic differentiation. International Journal for Numerical Methods in Engineering 113, 1081–1107.

Schöberl, J., 2014. C++11 implementation of finite elements in ngsolve. Institute for Analysis and Scientific Computing, Vienna University of Technology, 30 .

Sethian, J.A., 1999. Level set methods and fast marching methods: evolving interfaces in computational geometry, fluid mechanics, computer vision, and materials science. Cambridge University Press 3.

Souza, B.C., Yamabe, P.V.M., S, L.F.N., Ranjbarzadeh, S., Picelli, R., Silva, E.C.N., 2021. Topology optimization of fluid flow by using integer linear programming. Structural and Multidisciplinary Optimization 64, 1221–1240.

Walther, A., Griewank, A., 2012. Getting started with ADOL-C. Combinatorial Scientific Computing, Chapman-Hall CRC Computational Science .

Wang, M.Y., Wang, X., Guo, D., 2003. A level set method for structural topology optimization. Computer Methods in Applied Mechanics and Engineering 192, 227–246.

Yan, J., Xiang, R., Kamensky, D., Tolley, M.T., Hwang, J.T., 2022. Topology optimization with automated derivative computation for multidisciplinary design problems. Structural and Multidisciplinary Optimization 65.