AuTO: A Framework for Automatic differentiation in Topology Optimization

Aaditya Chandrasekhar
Department of Mechanical Engineering
University of Wisconsin-Madison
Madison, WI
achandrasek3@wisc.edu

Saketh Sridhara
Department of Mechanical Engineering
University of Wisconsin-Madison
Madison, WI
ssridhara@wisc.edu

Krishnan Suresh
Department of Mechanical Engineering
University of Wisconsin-Madison
Madison, WI
ksuresh@wisc.edu

April 2, 2021

Abstract

A critical step in topology optimization (TO) is finding sensitivities. Manual derivation and implementation of the sensitivities can be quite laborious and error-prone, especially for non-trivial objectives, constraints and material models. An alternate approach is to utilize automatic differentiation (AD). While AD has been around for decades, and has also been applied in TO, wider adoption has largely been absent.

In this educational paper, we aim to reintroduce AD for TO, and make it easily accessible through illustrative codes. In particular, we employ JAX, a high-performance Python library for automatically computing sensitivities from a user defined TO problem. The resulting framework, referred to here as AuTO, is illustrated through several examples in compliance minimization, compliant mechanism design and microstructural design.

1 Introduction

Fueled by improvements in manufacturing capabilities and computational modeling, the field of topology optimization (TO) has witnessed tremendous growth in recent years. To further accelerate the development of TO, we consider here automating a critical step in TO, namely computing the sensitivities, i.e., computing the derivatives of objectives, constraints, material models, projections and filters, with respect to the design variables, typically the elemental pseudo-densities, in the popular density-based TO.

Conceptually, there are four different methods for computing sensitivities [1]: (1) numerical, (2) symbolic, (3) manual, and (4) automatic differentiation. Numerical, i.e., finite difference, based sensitivity computation suffers from truncation and floating-point errors, and is therefore not recommended. Symbolic differentiation using software packages such as SymPy [2] or Mathematica [3] is a reasonable choice for simple expressions. However, it is impractical when the quantity of interest involves loops (such as when assembling stiffness matrices), and/or flow-control (if-then-else). The default method today for computing sensitivities is manual. While theoretically straightforward, the manual process is unfortunately cumbersome and error prone; it is often the bottle-neck in the development of new TO modules and exploratory studies. In this educational paper, we therefore *illustrate and promote the use of automatic differentiation for computing sensitivities in TO*.

Automatic differentiation (AD), is a collection of methods for efficiently and accurately computing derivatives of numeric functions expressed as computer programs [1]. AD has been around for decades [4] and has been exploited in a wide range of problems ranging from molecular dynamics simulations [5] to the design of photonic crystals [6]; see [7] for a critical review. AD in the context of finite element analysis is reviewed in [8] and [9]. More recently, AD was demonstrated for for shape optimization in [10] using the Firedrake framework [11]. AD has also been exploited in TO of turbulent fluid flow systems [12], [13].

Despite the pioneering research, AD is not widely used in TO. The objective of this educational paper *is to accelerate the adoption of AD in TO by providing standalone codes for popular TO problems.* In particular, we employ JAX [14], a high-performance Python library for end-to-end AD. Its NumPy [15] like syntax, low memory footprint and support of just-in-time (JIT) compilation for accelerated code performance makes it an ideal candidate for the task. We demonstrate the use of AD within the popular density-based TO framework [16], by replicating existing educational TO codes for compliance minimization [17], compliant mechanism design [18] and microstructural design [19]. Critical code snippets are highlighted in this article; the complete codes are available at https://github.com/UW-ERSL/AuTO

2 Compliance minimization

2.1 Problem Formulation

First we consider compliance minimization as modeled [17] in subject to a volume constraint; this TO problem is very popular due to its self-adjoint nature. In a mesh discretized form, the problem can be posed as:

$$\underset{\boldsymbol{\rho}}{\text{minimize}} \quad J = \boldsymbol{u}^{\mathsf{T}}\boldsymbol{K}(\boldsymbol{\rho})\boldsymbol{u} \tag{1a}$$

$$\text{subject to} \quad \boldsymbol{K}(\boldsymbol{\rho})\boldsymbol{u} = \boldsymbol{f} \tag{1b}$$

$$\sum_e \rho_e v_e \leq V^* \tag{1c}$$

where $\boldsymbol{u}$ is the displacement (in structural problems) or temperature (in thermal problems), $\boldsymbol{K}$ is the stiffness matrix, $\boldsymbol{\rho}$ is the pseudo-density design variables, $\boldsymbol{f}$ is the structural/thermal load and V^* is the volume constraint. To solve this problem, one must define the material model (see below), rely on finite element analysis to solve Equation 1b, and use design update schemes such as MMA [20] or Optimality Criteria [21].

A critical ingredient for the design update schemes is the sensitivity, i.e., derivative, of the objective and constraint with respect to the pseudo-density variables. As mentioned earlier, this is typically carried out manually. For the above self-adjoint problem, the sensitivity of the compliance, for the solid isotropic material with penalization (SIMP) [21] material model, can be easily derived:

$$\frac{\partial J}{\partial \rho_e} = -\boldsymbol{u}^T \frac{\partial \boldsymbol{K}}{\partial \rho_e}\boldsymbol{u} = -p\rho^{p-1}{u_e}^T K u_e \tag{2}$$

However, in this paper, we will rely on automatic differentiation (AD) framework for sensitivity analysis.

2.2 AuTO Framework

The algorithm for solving the above compliance minimization is illustrated in 1. Code snippets that illustrate the use of AD are discussed below.

Algorithm 1 Compliance Minimization

1: **procedure** COMPLIANCEMIN(mesh, material, filter, BC, V^*)
2: $i = 0$ ▷ Iteration index
3: $\rho = V^*$ ▷ Design variable initialization
4: $\Delta = 1.0$ ▷ Design change
5: **while** $\Delta > \epsilon$ and $i \leq$ MaxIter **do**
6: $i \leftarrow i + 1$
7: $E \leftarrow \rho$ ▷ Material model
8: $\boldsymbol{K} \leftarrow E$ ▷ Compute stiffness matrix and assemble
9: u via $Ku = f$ ▷ Solve with imposed BC
10: $J \leftarrow (K, u)$ ▷ Objective
11: $\frac{\partial \mathcal{J}}{\partial \rho} \leftarrow AD(\rho \rightarrow J)$ ▷ Automatic differentiation of objective
12: $g \leftarrow (\bar{\rho}, V^*)$ ▷ Vol. Constraint
13: $\frac{\partial g}{\partial \rho} \leftarrow AD(\rho \rightarrow g)$ ▷ Automatic differentiation of constraint
14: $\phi^i \leftarrow (J, g, \frac{\partial J}{\partial \rho}, \frac{\partial g}{\partial \rho})$ ▷ MMA Solver [20]
15: $\Delta = (||\rho^i - \rho^{i-1}||)$
16: **end while**
17: **end procedure**

Steps 7-10 are captured through the following Python code, where the @jit directive refers to the "just-in-compilation", i.e., the compiler translates the Python functions to optimized machine code at run-time, approaching the speeds of C or FORTRAN [22].

```
@jit
def computeCompliance(rho):
  E = MaterialModel(rho)
  K = assembleK(E)
  u = solveKuf(K)
  J = jnp.dot(u.T, jnp.dot(K,u))
  return J
```

SIMP [21] is a typical material model, and implemented as follows (the @jit directive has been removed here to avoid repetition).

```
def MaterialModel(rho):
    E = Emin + (Emax-Emin)*rho**penal # SIMP
    return E
```

The stiffness matrix is assembled in a compact manner as follows.

```
def assembleK(E):
    K = jnp.zeros((ndof,ndof))
    sK = D0.flatten()[np.newaxis]*E.T.flatten()
    K = jax.ops.index_add(K, idx, sK)
    return K;
```

where $D0 = \int_{\Omega_e} [B]^T [C_0][B] d\Omega_e$ is the element base stiffness matrix [23] with $E = 1$ and prescribed ν; idx reflects the global numbering of the element nodes. The underlying linear system is solved using a direct solver.

```
def solveKuf(K):
    u_free = jax.scipy.linalg.solve(K[free,:][:,free],force[free])
    u = jnp.zeros((ndof))
    u = jax.ops.index_add(u, free,u_free.reshape(-1))
    return u;
```

Finally, to compute the compliance and its sensitivity in step 11, we simply request for the function and its derivative as follows. The JAX environment automatically traces the chain of function calls, and ensures an end-to-end automatic differentiation.

```
J,  gradJ = value_and_grad(computeCompliance)(rho)
```

The global volume constraint (in step 11) is defined as follows,

```
@jit
def globalVolumeConstraint(rho):
    vc = jnp.mean(rho)/vf - 1.
    return vc;
```

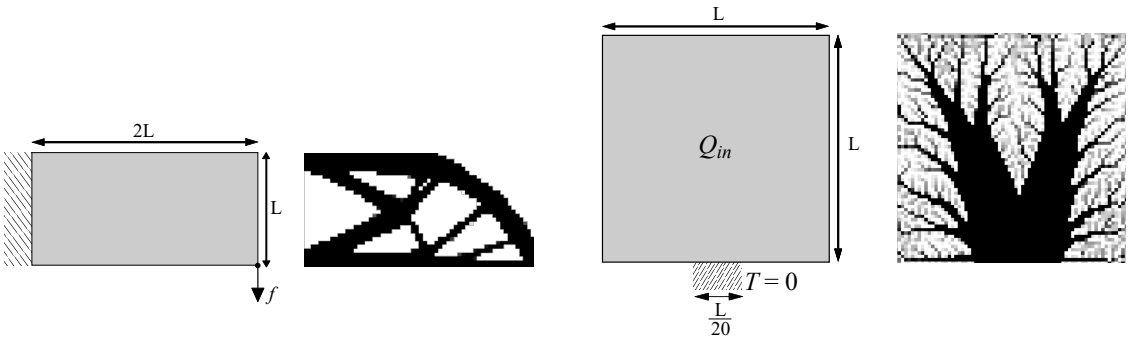
As before, the value and its gradient (via AD) can be computed via

```
g, gradg = value_and_grad(globalVolumeConstraint)(rho)
```

As summarized in step 14 of the algorithm, the computed objective, objective gradient, constraint, constraint gradient are then passed to standard optimizers (MMA in our case) [20]. The reader is referred to the complete code provided.

2.3 Illustrative Examples

We illustrate the above AD framework using two popular examples of compliance minimization [16]: (a) minimizing structural compliance of a tip-loaded cantilever (see Figure 1a) and (b) minimizing thermal compliance of a square plate under a uniform heat load (see Figure 1b). The mesh was chosen to be 60×30 grid for the structural problem, and 60×60 grid for the thermal problem. The target volume fraction in both problems is $V^* = 0.5$. The material properties are $E = 1$, $\nu = 0.3$, $k = 1$, and MMA was used as the design update scheme, with default parameters. The computed designs illustrated in 1a and 1b matches those in the literature [16].



(a) Structural problem (b) Thermal problem

Figure 1: Compliance minimization examples: (a) Tip loaded cantilever and optimized topology at $V^* = 0.5$ (b) Heat conduction on a square plate and optimized topology at $V^* = 0.5$

The totla time taken for optimization using analytical derivatives and AD are compared in Figure 2. We observe that AD is marginally more expensive, but we will observe later that this is not always the case.

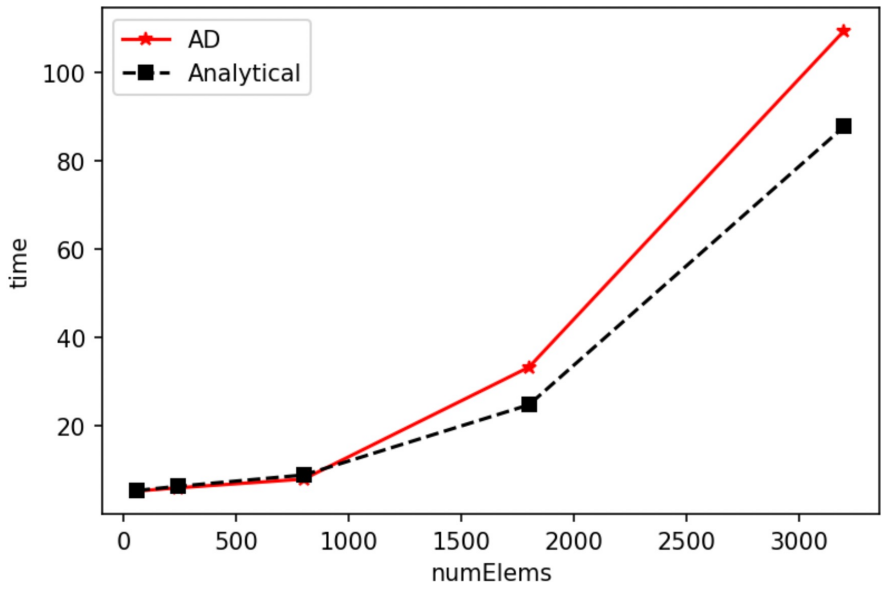


Figure 2: Computational cost for compliance minimization using AD and analytical implementation.

2.4 Advantages of AD

While SIMP is a popular material model, other models have been proposed [24]. The advantage of AD is that one can easily replace SIMP with, for example, RAMP [25], by simply changing the material model.

```
def MaterialModel(rho):
    E = Emax*rho/(1.+S*(1.-rho)) # RAMP
    return Y
```

All downstream sensitivity computations are handled automatically.

Often additional filters and projections are used in TO. For instance, they can be used to remove checkerboard patterns [26], impose minimum length scale [27], limit gray elements [28], etc. The filters apart from being complex in their own right, they are often used in tandem. For instance, in [29], eight such schemes were compounded to obtain shell-infill type structures. This results in highly complicated sensitivity expressions that can be laborious to derive. However, using an AD framework, the user simply needs to include the desired projections in the pipeline and the sensitivity is taken care of. For instance, we can introduce the following filter to reduce grayness in design, just before computing the material model.

```
def projectionFilter(rho):
    if(projection['isOn']):
        nmr = np.tanh(c0*beta) + jnp.tanh(beta*(rho-c0))
        dnmr = np.tanh(c0*beta) + np.tanh(beta*(1-c0))
        rho_tilde = nmr/dnmr
        return rho_tilde
    else:
        return rho
```

Finally, manufacturing constraints [30], [31] are often imposed in TO; these include limiting overhang of structures [32], connectivity [33], material utilization [34], and length scale control [35]. Such constraints are easy to impose within the AD framework. For example, the local volume constraint proposed in [35] may be implemented as follows.

```
def maxLengthScaleConstraint(rho):
    v = jnp.matmul(L, (1.01-rho)**n); # L averaged prior
    cons = 1 - jnp.power(jnp.sum(v**p),1./p)/vstar;
    return cons;
```

As before, one can calculate the value and gradient of the constraint via

```
vc, gradvc = value_and_grad(maxLengthScaleConstraint)(rho)
```

The computed constraint and gradient can then be passed on to MMA. To illustrate, for the tip cantilever problem in Figure 1(a), with the additional max length scale radius of $r = 30$, and maximum void volume at $0.75\pi r^2$, the resulting topology is illustrated in Figure 3.

Figure 3: Tip cantilever beam with length scale control.

3 Compliant Mechanism Design

We next illustrate the AuTO framework using compliant mechanisms (CMs) [36]; see [37] for a comprehensive review on TO for CMs.

3.1 Problem Formulation

Consider the displacement inverter considered in the 104-line educational MATLAB code [16]. The objective is to maximize the output displacement u_{out} at the point of interest when a force f_{in} is applied, as illustrated in Figure 4. The spring constants are specified by the user to control the behavior of the CM.

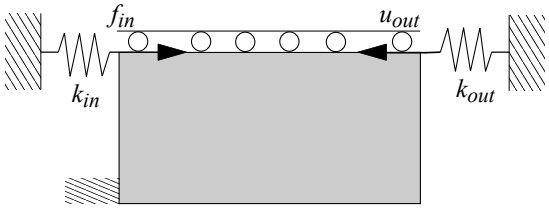


Figure 4: The displacement inverter compliant mechanism.

This TO problem can be written as:

$$\begin{aligned} \underset{\boldsymbol{\rho}}{\text{maximize}} \quad & \boldsymbol{u}_{out} && \text{(3a)} \\ \text{subject to} \quad & \boldsymbol{K}(\boldsymbol{\rho})\boldsymbol{u} = \boldsymbol{f}_{\boldsymbol{in}} && \text{(3b)} \\ & \sum_e \rho_e v_e \leq V^* && \text{(3c)} \end{aligned}$$

The standard implementation entails computing the elemental sensitivity $\frac{\partial u_{out}}{\partial \rho_e}$ given by:

$$\frac{\partial u_{out}}{\partial \rho_e} = \boldsymbol{\lambda}^T \frac{\partial \boldsymbol{K}}{\partial \rho_e}\boldsymbol{u} = p\rho^{p-1}{\lambda_e}^T K u_e \tag{4}$$

where $\boldsymbol{\lambda}$ is the solution to the adjoint load problem $\boldsymbol{K\lambda} = -\boldsymbol{l}$. $\boldsymbol{l}$ is a vector with the value 1 at the degree of freedom corresponding to the output point, and with zeros elsewhere. Observe that, in the manual method, two sets of analysis (one to compute $\boldsymbol{u}$, and the other to compute $\boldsymbol{\lambda}$) are required per iteration for evaluating sensitivities.

3.2 AuTO Framework

The implementation in AuTO for CM design is similar to the compliance minimization problem, with two minor changes: (a) the stiffness matrix assembly includes the spring constants, and (b) the objective is the displacement at the output node. The relevant code snippets are provided below.

```
def assembleKWithSprings(E):
    K = jnp.zeros((ndof,ndof))
    sK = DO.flatten()[np.newaxis]*E.T.flatten()
    K = jax.ops.index_add(K, idx, sK)
    # springs at input and output nodes
    K = jax.ops.index_add(K, jax.ops.index[nodeIn, nodeIn], kspringIn)
    K = jax.ops.index_add(K, jax.ops.index[nodeOut, nodeOut], kspringOut)
    return K;
```

```
def CompliantMechanism(rho):
  E = MaterialModel(rho)
  K = assembleKWithSprings(E)
  u = solveKuf(K)
  return u[bc['nodeOut']]
```

To compute the objective and its gradient, we rely on JAX as follows.

```
J,  gradJ = value_and_grad(CompliantMechanism)(rho)
```

The design update using MMA is as per Section 2. Using the problem specification in [18], the resulting topology for the inverter is illustrated in Figure 5a; this is in agreement with the result in [18].

3.3 Advantages of AD

For the design of CMs using TO, a key advantage of AD stems from the following observation [37] "*no universally accepted objective formulation exists*". For example, consider two additional objectives:

1. min : $-\omega MSE + (1-\omega)SE$ [38]
2. min : $-MSE/SE$ [39]

where $MSE = \boldsymbol{v}^T\boldsymbol{K}\boldsymbol{u}$ is the mutual strain energy, which describes the flexibility of the designed mechanism and $SE = \boldsymbol{u}^T\boldsymbol{K}\boldsymbol{u}$ is the strain energy, $\boldsymbol{v}$ is the output displacement when a unit dummy load applied at the degree of freedom corresponding to the output point.

In the AuTO framework, one can easily explore various objectives as follows.

```
def CompliantMechanism(rho):
  E = MaterialModel(rho)
  K = assembleKWithSprings(E)
  u = solveKuf(K)
  v = solve_dummy(K)
  MSE = jnp.dot(v.T, jnp.dot(K,u))
  SE = jnp.dot(u.T, jnp.dot(K,u));
  J = -MSE/SE # or
  # w = 0.9
  # J = -w*MSE + (1 - w)*SE
  return J
```

The topologies obtained with the two additional objectives are illustrated in Figure 5b and Figure 5c.

(a) min : $-u_{out}$ (b) min : $-\omega MSE + (1-\omega)SE$ (c) min : $-MSE/SE$

Figure 5: Displacement inverter design using three formulations at $V^* = 0.35$

For the objective of maximizing output displacement, the computational costs using analytical and AD methods are illustrated in Figure 6. Observe that the the analytical method is more expensive since one must solve an adjoint problem explicitly. On the other hand, JAX internally optimizes the code for computing sensitivities via AD.

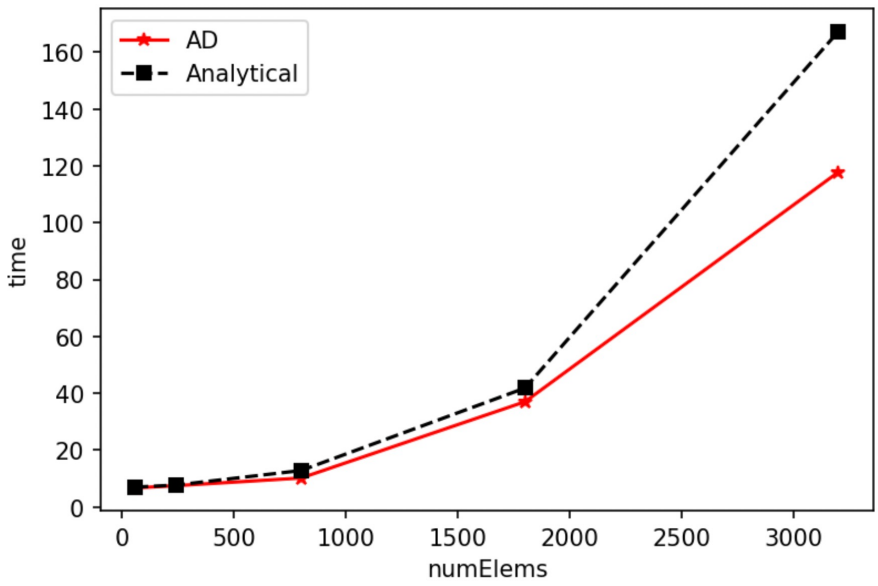


Figure 6: Computational cost of optimization using AD vs analytical implementation.

4 Design of Materials

In this section, we replicate the educational article [19] for the design of microstructures using AuTO. In particular, we consider (a) maximizing bulk modulus (b) maximizing shear modulus, and (c) designing microstructures with negative Poisson's ratio.

4.1 Problem setup

The mathematical formulation is as follows [19]:

$$\begin{aligned} \underset{\rho}{\text{minimize}} \quad & c(E^H_{ijkl}(\boldsymbol{\rho})) && \text{(5a)} \\ \text{subject to} \quad & \boldsymbol{K}(\boldsymbol{\rho})\boldsymbol{U}^{A(kl)} = \boldsymbol{F}^{(kl)}, k, l = 1, 2, \ldots, d && \text{(5b)} \\ & \sum_e \rho_e v_e \leq V^* && \text{(5c)} \end{aligned}$$

where the objective $c(E^H_{ijkl})$ represents the material property we intend to minimize, $\boldsymbol{K}$ is the stiffness matrix, $\boldsymbol{U}^{A(kl)}$ and $\boldsymbol{F}^{(kl)}$ are the displacement vector and the external force vector for test case (kl) respectively. The different test cases correspond to the unit strain tests along different directions, where d is the spatial dimension.

In 2D, maximization of bulk modulus corresponds to:

$$c = -(E_{1111} + E_{1122} + E_{2211} + E_{2222}) \tag{6}$$

and maximization of shear modulus corresponds to:

$$c = -E_{1212} \tag{7}$$

Finally, for the design of materials with negative Poisson's ratio, the following was proposed [19]:

$$c = -E_{1122} - \beta^l(E_{1111} + E_{2222}) \tag{8}$$

where $\beta \in (0, 1)$ is a user-defined fixed parameter and l is the design iteration number. Observe that, in the manual method, computing the sensitivity requires solving for the adjoint [19],

4.2 Implementation on AuTO

In AuTO, the bulk modulus objective, for example, can be captured as follows:

```
def MicrosrucuturalDesign(rho):
    E = MaterialModel(rho)
    K = assembleK(E)
    Kr, F = computeSubMatrices(K)
    U = performFE(Kr, F)
    EMatrix = homogenizedMatrix(U, rho)
    bulkModulus = -EMatrix['0_0']-EMatrix['0_1']-EMatrix['1_1']-EMatrix['1_0']
    return bulkModulus
```

Other objectives can be similarly captured. Figure 7 illustrates three different microstuctures for the three different objectives, for a volume fraction of 0.25.

Figure 7: Maximization of bulk modulus, shear modulus and design of material with negative Poisson's ratio, with $v^*_f = 0.25$.

5 Conclusion

In this paper, we demonstrated the simplicity and benefits of AD in TO. Possible extensions include multi-physics [40, 41, 42] and non-linear problems [43, 44]. In the current implementation, direct solvers were employed. For large scale problems, sparse pre-conditioned iterative solvers [45], [46] will be critical (but not fully supported by JAX). One of the advantages of the manual approach (that is lost in AD) is that the expressions can provide key insights to the problem.

Replication of results

The Python code pertinent to this paper is available at https://github.com/UW-ERSL/AuTO.

Acknowledgments

The authors would like to thank the support of National Science Foundation through grant CMMI 1561899.

Compliance with ethical standards

The authors declare that they have no conflict of interest.

References

[1] Atılım Günes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *The Journal of Machine Learning Research*, 18(1):5595–5637, 2017.

[2] Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, January 2017.

[3] Wolfram Research, Inc. Mathematica, Version 12.2. Champaign, IL, 2020.

[4] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[5] Samuel S Schoenholz and Ekin D Cubuk. Jax md: End-to-end differentiable, hardware accelerated, molecular dynamics in pure python. 2019.

[6] Momchil Minkov, Ian AD Williamson, Lucio C Andreani, Dario Gerace, Beicheng Lou, Alex Y Song, Tyler W Hughes, and Shanhui Fan. Inverse design of photonic crystals through automatic differentiation. *ACS Photonics*, 7(7):1729–1741, 2020.

[7] Louis B Rall. Perspectives on automatic differentiation: past, present, and future? In *Automatic Differentiation: Applications, Theory, and Implementations*, pages 1–14. Springer, 2006.

[8] I Ozaki, F Kimura, and M Berz. Higher-order sensitivity analysis of finite element method by automatic differentiation. *Computational mechanics*, 16(4):223–234, 1995.

[9] F Van Keulen, RT Haftka, and NH Kim. Review of options for structural design sensitivity analysis. part 1: Linear systems. *Computer methods in applied mechanics and engineering*, 194(30-33):3213–3243, 2005.

[10] Alberto Paganini and Florian Wechsung. Fireshape: a shape optimization toolbox for firedrake. *Structural and Multidisciplinary Optimization*, pages 1–17, 2021.

[11] Peter Gangl, Kevin Sturm, Michael Neunteufel, and Joachim Schöberl. Fully and semi-automated shape differentiation in ngsolve. *Structural and multidisciplinary optimization*, pages 1–29, 2020.

[12] Cetin B Dilgen, Sumer B Dilgen, David R Fuhrman, Ole Sigmund, and Boyan S Lazarov. Topology optimization of turbulent flows. *Computer Methods in Applied Mechanics and Engineering*, 331:363–393, 2018.

[13] Sumer B Dilgen, Cetin B Dilgen, David R Fuhrman, Ole Sigmund, and Boyan S Lazarov. Density based topology optimization of turbulent flow heat transfer systems. *Structural and Multidisciplinary Optimization*, 57(5):1905–1918, 2018.

[14] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[15] Charles R. Harris, K. Jarrod Millman, St'efan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fern'andez del R'10, Mark Wiebe, Pearu Peterson, Pierre G'erard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[16] Martin Philip Bendsoe and Ole Sigmund. *Topology optimization: theory, methods, and applications*. Springer Science & Business Media, 2013.

[17] Ole Sigmund. A 99 line topology optimization code written in matlab. *Structural and multidisciplinary optimization*, 21(2):120–127, 2001.

[18] M P Bendsoe and Ole. Sigmund. *Topology optimization: theory, methods, and applications*. Springer Berlin Heidelberg, 2 edition, 2003.

[19] Liang Xia and Piotr Breitkopf. Design of materials using topology optimization and energy-based homogenization approach in matlab. *Structural and multidisciplinary optimization*, 52(6):1229–1241, 2015.

[20] Krister Svanberg. The method of moving asymptotes—a new method for structural optimization. *International journal for numerical methods in engineering*, 24(2):359–373, 1987.

[21] Martin P Bendsøe and Ole Sigmund. *Optimization of structural topology, shape, and material*, volume 414. Springer, 1995.

[22] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. Numba: A llvm-based python jit compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, pages 1–6, 2015.

[23] Klaus-Jürgen Bathe. *Finite element procedures*. Klaus-Jurgen Bathe, 2006.

[24] Grzegorz Dzierżanowski. On the comparison of material interpolation schemes and optimal composite properties in plane shape optimization. *Structural and Multidisciplinary Optimization*, 46(5):693–710, 2012.

[25] Mathias Stolpe and Krister Svanberg. An alternative interpolation scheme for minimum compliance topology optimization. *Structural and Multidisciplinary Optimization*, 22(2):116–124, 2001.

[26] O. Sigmund and J. Petersson. Numerical instabilities in topology optimization: A survey on procedures dealing with checkerboards, mesh-dependencies and local minima. *Structural Optimization*, 16(1):68–75, 1998.

[27] James K Guest, Jean H Prévost, and Ted Belytschko. Achieving minimum length scale in topology optimization using nodal design variables and projection functions. *International journal for numerical methods in engineering*, 61(2):238–254, 2004.

[28] Jun Wu, Niels Aage, Ruediger Westermann, and Ole Sigmund. Infill Optimization for Additive Manufacturing – Approaching Bone-like Porous Structures. *IEEE Transactions on Visualization and Computer Graphics*, 24(2):1127–1140, 2018.

[29] Jun Wu, Anders Clausen, and Ole Sigmund. Minimum compliance topology optimization of shell–infill composites for additive manufacturing. *Computer Methods in Applied Mechanics and Engineering*, 326:358–375, Nov 2017.

[30] Sandro L. Vatanabe, Tiago N. Lippi, Cícero R.de Lima, Glaucio H. Paulino, and Emílio C.N. Silva. Topology optimization with manufacturing constraints: A unified projection-based approach. *Advances in Engineering Software*, 100:97–112, oct 2016.

[31] Jikai Liu, Andrew T Gaynor, Shikui Chen, Zhan Kang, Krishnan Suresh, Akihiro Takezawa, Lei Li, Junji Kato, Jinyuan Tang, Charlie CL Wang, et al. Current and future trends in topology optimization for additive manufacturing. *Structural and Multidisciplinary Optimization*, 57(6):2457–2483, 2018.

[32] Xiaoping Qian. Undercut and overhang angle control in topology optimization: A density gradient based integral approach. *International Journal for Numerical Methods in Engineering*, 111(3):247–272, Jul 2017.

[33] Quhao Li, Wenjiong Chen, Shutian Liu, and Liyong Tong. Structural topology optimization considering connectivity constraint. *Structural and Multidisciplinary Optimization*, 54(4):971–984, 2016.

[34] Emily D. Sanders, Miguel A. Aguiló, and Glaucio H. Paulino. Multi-material continuum topology optimization with arbitrary volume and mass constraints. *Computer Methods in Applied Mechanics and Engineering*, 340:798–823, oct 2018.

[35] James K. Guest. Imposing maximum length scale in topology optimization. *Structural and Multidisciplinary Optimization*, 37(5):463–473, 2009.

[36] Larry L Howell. Compliant mechanisms. In *21st century kinematics*, pages 189–216. Springer, 2013.

[37] Benliang Zhu, Xianmin Zhang, Hongchuan Zhang, Junwen Liang, Haoyan Zang, Hai Li, and Rixin Wang. Design of compliant mechanisms using continuum topology optimization: a review. *Mechanism and Machine Theory*, 143:103622, 2020.

[38] Shinji Nishiwaki, Mary I Frecker, Seungjae Min, and Noboru Kikuchi. Topology optimization of compliant mechanisms using the homogenization method. *International journal for numerical methods in engineering*, 42(3):535–559, 1998.

[39] A Saxena and GK Ananthasuresh. On an optimal property of compliant topologies. *Structural and multidisciplinary optimization*, 19(1):36–49, 2000.

[40] Joe Alexandersen and Casper Schousboe Andreasen. A review of topology optimisation for fluid-based problems. *Fluids*, 5(1):29, 2020.

[41] Johannes Semmler, Lukas Pflug, and Michael Stingl. Material optimization in transverse electromagnetic scattering applications. *SIAM Journal on Scientific Computing*, 40(1):B85–B109, 2018.

[42] Shiguang Deng and Krishnan Suresh. Topology optimization under thermo-elastic buckling. *Structural and Multidisciplinary Optimization*, 55(5):1759–1772, 2017.

[43] Fengwen Wang, Ole Sigmund, and Jakob Søndergaard Jensen. Design of materials with prescribed nonlinear properties. *Journal of the Mechanics and Physics of Solids*, 69:156–174, 2014.

[44] Anders Clausen, Fengwen Wang, Jakob S Jensen, Ole Sigmund, and Jennifer A Lewis. Topology optimized architectures with programmable poisson's ratio over large deformations. *Adv. Mater*, 27(37):5523–5527, 2015.

[45] Martin S Andersen, Joachim Dahl, and Lieven Vandenberghe. Cvxopt: A python package for convex optimization. *abel. ee. ucla. edu/cvxopt*, 2013.

[46] Praveen Yadav and Krishnan Suresh. Assembly-Free Large-Scale Modal Analysis on the Graphics-Programmable Unit. *Journal of Computing and Information Science in Engineering*, 13(1):011003, Jan 2013.