Computer Methods in Applied Mechanics and Engineering 435 (2025) 117572

Contents lists available at ScienceDirect

Comput. Methods Appl. Mech. Engrg.

journal homepage: www.elsevier.com/locate/cma

Simplifying FFT-based methods for solid mechanics with automatic differentiation

Mohit Pundir, David S. Kammer *

Institute for Building Materials, ETH Zurich, Switzerland

ARTICLE INFO

Dataset link: https://gitlab.ethz.ch/cmbm-public/papers-supp-info/2025/fft-methods-with-AD

Keywords:
Spectral method
Automatic differentiation
Multiscale
Homogenization

ABSTRACT

Fast-Fourier Transform (FFT) methods have been widely used in solid mechanics to address complex homogenization problems. However, current FFT-based methods face challenges that limit their applicability to intricate material models or complex mechanical problems. These challenges include the manual implementation of constitutive laws and the use of computationally expensive and complex algorithms to couple microscale mechanisms to macroscale material behavior. Here, we incorporate automatic differentiation (AD) within the FFT framework to mitigate these challenges. We demonstrate that AD-enhanced FFT-based methods can derive stress and tangent stiffness directly from energy density functionals, facilitating the extension of FFT-based methods to more intricate material models. Additionally, automatic differentiation simplifies the calculation of homogenized tangent stiffness for microstructures with complex architectures and constitutive properties. This enhancement renders current FFT-based methods more modular, enabling them to tackle homogenization in complex multiscale systems, especially those involving multiphysics processes. Furthermore, we illustrate the use of the AD-enhanced FFT method for problems that extend beyond homogenization, such as uncertainty quantification and topology optimization where automatic differentiation simplifies the computation of sensitivities. Our work will simplify the numerical implementation of FFT-based methods for complex solid mechanics problems.

1. Introduction

A crucial aspect of computational multiscale mechanics is understanding how microscale mechanisms impact macroscale material behavior. Features such as the distribution of heterogeneities, the formation of structures at the micro level, and physical processes occurring on the microscale can significantly influence the properties homogenized at a larger scale. The Fast-Fourier Transform (FFT) methods [1–4] are particularly well-suited for tackling such homogenization problems due to their computational efficiency [5,6], ease of implementation [7], and low memory footprint. Nonetheless, certain aspects of FFT frameworks limit their quick adaptation to various applications and extension to intricate mechanical systems.

A significant challenge with FFT-based methods is that they still rely on manually implemented stress–strain relationships, which are computed "by hand" either analytically or through numerical differentiation [5]. This makes the implementation of complex constitutive relations (*e.g.*, hyperelastic materials and elastoplastic materials) intricate and error-prone within the FFT framework, especially for cases involving multi-physical processes [8]. Hence, one of their major strength (*i.e.* ease of implementation) is lost for more complex problems. Another prominent limitation lies in determining the homogenized tangent stiffness in multiscale simulations [9–13]. Current FFT-based multiscale frameworks require auxiliary solution schemes [14] to

* Corresponding author.
E-mail addresses: mpundir@ethz.ch (M. Pundir), dkammer@ethz.ch (D.S. Kammer).

https://doi.org/10.1016/j.cma.2024.117572
Received 7 August 2024; Received in revised form 11 November 2024; Accepted 14 November 2024
Available online 5 December 2024

couple microscale mechanisms to the macroscale through a constitutive tangent stiffness, which can be computationally expensive and inaccurate for complex material models and large computational domains.

Other numerical methods have similar challenges, which have been overcome by adopting a differentiable framework, which automates the process of computing derivatives [15]. This allows the formulation of the governing equation in a high-level manner (usually as an expression of the functional energy), and the complex Jacobian and the Hessian are then strictly derived through automatic differentiation (AD) up to machine precision [8,16]. Among prominent examples of such differentiable frameworks within the field of solid mechanics is FEniCSx [17] for Finite Element Methods. Concurrently, with the rise of AD libraries such as PyTorch [18] and JAX [19], the differentiable framework has become quite appealing, which is evident by the shift in porting existing frameworks such as molecular dynamics, discrete element method, and finite element method [20–24] to a differentiation-based framework. Similarly, automatic differentiation presents great potential for FFT-based methods [25] but has yet to be developed in a general and ease-of-use manner to overcome the main challenges of FFT-based methods.

In this paper, we enhance the FFT framework by incorporating automatic differentiation. We investigate the capabilities of an AD-enhanced FFT framework and examine how automatic differentiation mitigates some of the aforementioned issues. Thereby making FFT-based frameworks more modular and easy to implement for complex solid mechanics problems.

2. How automatic differentiation works?

Consider a function $f : \mathbb{R}^n \to \mathbb{R}^m$ that maps an n-dimensional input x to an m-dimensional output. To determine the derivatives of the outputs relative to the inputs, specifically $J_{ij} = \partial f_i/\partial x_j$ where $i \in m$ and $j \in n$, one typically depends on the hand-calculated closed-form expression of the derivative of the function. In case the function f has a complex closed-form mathematical expression, one can use symbolic differentiation to get the derivatives [26]. In computational solid mechanics, f may represent a complex function consisting of numerous procedures that do not possess an explicit mathematical expression. For example, f might be a function that computes the Fourier transform of the input data with a conditional statement (if-then-else) or operates a minimization solver with for-loops on the inputs. In such cases, neither analytical nor symbolic differentiation is feasible, requiring the use of alternative approaches to estimate the derivatives $\partial f_i/\partial x_j$. A classical approach is numerical differentiation. Its results are very sensitive to the perturbations used in the calculations, which may result in inaccuracies or numerical stability issues. Alternatively, automatic differentiation overcomes such limitations by computing exact derivatives for such functions. This is achieved by decomposing a complex function f into elementary functions, for which exact derivatives are known, and interconnecting them through basic arithmetic operations such as addition and subtraction. The chain rule of calculus is then applied to compute exact partial derivatives. For instance, if the function f can be decomposed as $f = h(g(x))$, such that $x \in \mathbb{R}^n, g : \mathbb{R}^n \to \mathbb{R}^k$ and $h : \mathbb{R}^k \to \mathbb{R}^m$, then the partial derivative $\partial f_i/\partial x_j$ is computed as

$$J_{ij} = \frac{\partial f_i}{\partial x_j} = \sum_{p=1}^{k} \frac{\partial h_i}{\partial g_p} \frac{\partial g_p}{\partial x_j}. \tag{1}$$

As automatic differentiation utilizes the exact derivatives of the sub-functions to determine the derivative of f, it is not affected by issues related to numerical stability or round-off errors. The derived values retain the same level of precision as the function results, thereby enabling the accurate calculation of derivatives even in the case of nonlinear or non-monotonic functions. Moreover, automatic differentiation can be repeatedly applied to a function to derive exact higher-order derivatives without accumulating errors.

Automatic differentiation has two ways to calculate the derivative of f using the chain rule [15,19]. The first technique involves evaluating the derivatives of sub-functions (inner derivatives) simultaneously with the function evaluation. This approach is known as *forward-mode* differentiation. Fig. 1a illustrates forward-mode differentiation applied to a composite function $f = g \circ h$, mapping a scalar to a scalar, which requires only one pass to compute both the output of the function and its gradient. Alternatively, the second method involves first evaluating the function's output and then moving backwards to determine the derivatives, starting with outer derivatives followed by inner ones. This is called *reverse-mode* differentiation and requires at least two evaluations to extract the function's value and gradient. Fig. 1a depicts this backward propagation of the function to first compute outer derivatives and then inner ones. The choice of method depends on how the function f maps the input space to the output space. We demonstrate this with two simple cases for a function $f : \mathbb{R}^n \to \mathbb{R}^m$: (i) when $n < m$, *i.e.*, $f = (g_1(h(x)), g_2(h(x)))$ as shown in Fig. 1b, and (ii) when $n > m$, i.e., $f = g(h(x, y))$ as shown in Fig. 1c. In the case of $n = 1 < 2 = m$ (see Fig. 1b), forward-mode requires a single pass to calculate both partial derivatives $(\partial f_1/\partial x, \partial f_2/\partial x)$, while reverse-mode needs two backward computations to determine these derivatives. Conversely, when $n = 2 > 1 = m$, forward-mode requires two evaluations: one using x (assuming $y = 0$) and another using y (assuming $x = 0$), for computing partial derivatives $(\partial f/\partial x, \partial f/\partial y)$. However, reverse-mode achieves this with a single pass. Therefore, as a basic rule, for a function $f : \mathbb{R}^n \to \mathbb{R}^m$, forward-mode is computationally efficient when mapping from a lower-dimensional space to a higher-dimensional one, *i.e.*, when $n \ll m$. In contrast, reverse-mode is efficient when the function maps from a higher to a lower dimension space, indicating $n \gg m$. We use this guideline throughout the paper to find the most efficient way to differentiate.

For this study, we use JAX for automatic differentiation due to its effective support for both forward- and reverse-mode differentiation, just-in-time (JIT) compilation, and efficient parallelization on GPUs. JAX is particularly well suited for scientific computing tasks requiring high-order derivatives, offering both flexibility and high performance.

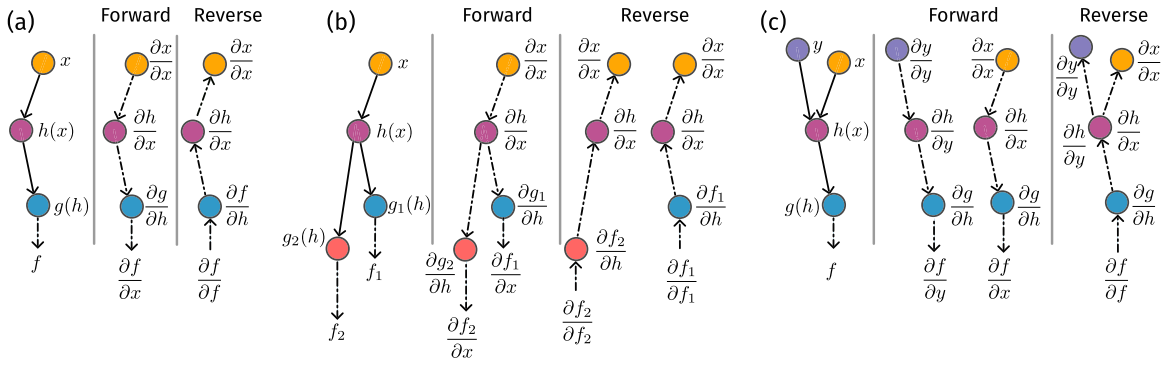

Fig. 1. Forward vs. Reverse mode for a composable function $f : g \circ h$ (a) Schematic illustration showing automatic differentiation applied to function $f : \mathbb{R} \to \mathbb{R}$. The left column shows the execution graph for computing output of f. The center column illustrates the execution graph for calculating the derivative of f via forward-mode. Observe that each node representing a function call is substituted by its derivative and is traversed in the original function call direction. The right column indicates the call direction for the reverse-mode, note the backwards arrows. (b) Schematic illustration showing automatic differentiation applied to function $f : \mathbb{R}^1 \to \mathbb{R}^2$. Notice that the forward-mode differentiation requires only a single pass to compute the 2 partial derivatives whereas reverse-mode requires 2 passes to compute the partial derivatives. The 2 passes are indicated by the 2 separate graphs. (c) Schematic illustration showing automatic differentiation applied to function $f : \mathbb{R}^2 \to \mathbb{R}^1$. Here the forward-mode requires 2 passes to compute the partial derivatives with respect 2 input values whereas reverse-mode requires a single pass.

3. Differentiable FFT-framework for solid mechanics

For the purpose of this work, we incorporate automatic differentiation within the Fourier-Galerkin approach [3] for FFT-based solvers. To this end, we explore the specific arithmetic operations made more straightforward by automatic differentiation and present the relevant pseudo-code/algorithm.

Consider a Representative Volume Element (RVE) Ω subjected to an overall strain $\bar{\boldsymbol{e}}$. The governing equation for static mechanical equilibrium in this RVE is thus given by $\nabla \cdot \boldsymbol{\sigma} = \boldsymbol{0}$, where $\boldsymbol{\sigma}$ is a function/operator that maps the overall deformation to the appropriate local stress field. Under small-strain conditions, the overall deformation $\bar{\boldsymbol{e}}$ represents small-strain $\boldsymbol{\epsilon}$, and $\boldsymbol{\sigma}$ represents a function that maps local $\boldsymbol{\epsilon}$, to the Cauchy stress tensor $\boldsymbol{\tau}$. Conversely, for finite deformations, $\bar{\boldsymbol{e}}$ represents the deformation gradient $\overline{\boldsymbol{F}}$ and $\boldsymbol{\sigma}$ represents a function that maps the local deformation gradients $\boldsymbol{F}$ to the first Piola–Kirchhoff stress tensor $\boldsymbol{P}$. Since this paper addresses both conditions, we will use the stress operator $\boldsymbol{\sigma}$ as a generic term for stresses in the description of the method throughout this section. We note that, consequently, the derivations and procedures remain consistent for both assumptions.

In this study, we adopt the Fourier-Galerkin spectral method [3,4], and do so for several reasons. Primarily, the variational structure of this approach retains the convexity of the energy density [4,5], which ensures that a unique solution to a given problem exists. Secondly, to guarantee that the solution produces a compatible strain state, the method employs a projection operator $\mathbb{G}$, which converts an arbitrary strain state to a compatible state (periodic and curl-free). This operator is independent of the material model and remains consistent for both small-strain and finite deformation scenarios [7]. We will show that these characteristics of the Fourier-Galerkin approach make integrating automatic differentiation within the FFT-based framework straightforward.

Applying the Fourier-Galerkin method, we reformulate the static equilibrium equation as

$$\mathcal{F}^{-1}\{\widehat{\mathbb{G}}(\boldsymbol{\xi}) : \widehat{\boldsymbol{\sigma}}(\boldsymbol{\xi})\} = \boldsymbol{0}\,, \tag{2}$$

where $\boldsymbol{\xi}$ is the wavevector, $\widehat{\star}(\boldsymbol{\xi})$ is the Fourier transform of quantity $\star$, and $\mathcal{F}^{-1}\{\star\}$ is the inverse Fourier transform. For details on the derivation of the above equation and the exact form of the projection operator $\mathbb{G}$, please refer to [3,4,7,27]. The above equation is solved using a Newton–Krylov solver, where the Newton–Raphson loop is employed to handle nonlinearities, and a Krylov-subspace solver, such as Conjugate Gradient, is employed to enforce the compatibility conditions (for details, see Appendix A).

Here, we will show how automatic differentiation can be applied in an FFT-based framework. We will describe three use cases of automatic differentiation, namely the stress computation in elastic and hyperelastic materials, the computation of consistent tangent operator for nonlinear problems, and the computation of homogenized stiffness operator for multiscale simulations. For all cases, we include both the arithmetic operations as well as the counterpart pseudo-code to highlight the general applicability and ease of use of automatic differentiation in FFT-based methods. To this end, we employ the open-source JAX library [19] for automatic differentiation and demonstrate that only a few lines of code are needed to implement a general solution.

3.1. Automatic computation of stress tensor from energy functional

We consider the stress computation of elastic and hyperelastic materials, for which the stress function $\boldsymbol{\sigma}$ is the first derivative of the strain energy density, as given by

$$\boldsymbol{\sigma}(\boldsymbol{x}) = \frac{\partial \psi(\boldsymbol{e})}{\partial \boldsymbol{e}}\,. \tag{3}$$

```
def strain_energy(strain):
   # compute the energy based on the input `strain`
   return energy

# create a function to compute stresses using the
# derivative of energy  w.r.t `strain` (Eq 2)
compute_stresses = jax.jacrev(strain_energy)

# using the above created function to compute stresses
sigma = compute_stresses(strain)
```

Listing 1: Automatic differentiation to compute stress from energy density functional.

```
# create a function to compute stresses
compute_stresses = jax.jacrev(strain_energy)

# computing the function to compute local tangent modulii K as given in Eq 4
compute_local_tangent = jax.jacfwd(compute_stresses)

# compute the tangent modulii
tangent_modulii = compute_local_tangent(strain)

# compute the incremental stresses as given in Eq 4
dsigma = jax.numpy.einsum('ijkl, kl -> ij', tangent_modulii, dstrain)
```

Listing 2: Automatic differentiation to compute incremental stresses using forward-mode AD.

The application of automatic differentiation on Eq. (3) using the JAX library [19] is straightforward as we show in Listing 1. Here, we define a function that receives microscale strain as input and outputs the corresponding energy density. Since the function maps a 2nd-order strain tensor to a scalar value, we use the reverse-mode technique to differentiate the function to calculate stresses. We note that the input strain to the strain–energy function may be either a small strain ($\boldsymbol{\varepsilon}$) or a deformation gradient ($\boldsymbol{F}$) based on the defined energy density. In Section 4.1, we will present examples for various material models under small-strain and finite deformation settings.

3.2. Automatic computation of tangent moduli and incremental stresses

We integrate automatic differentiation to ease the numerical implementation of the nonlinear Eq. (2). In standard practice, Eq. (2) is linearized, and the Newton–Raphson method is used to find the equilibrium. The linearization process involves computing the incremental stresses and the consistent tangent operator at each step of the Newton–Raphson scheme, which is expressed as follows,

$$\delta\boldsymbol{\sigma} = \underbrace{\frac{\partial\boldsymbol{\sigma}(\boldsymbol{e})}{\partial\boldsymbol{e}}}_{\mathbb{K}}\,\delta\boldsymbol{e}\ . \tag{4}$$

Consequently, the local tangent stiffness $\mathbb{K}$ is needed at the previous strain to calculate the incremental stresses. With automatic differentiation, there are three different ways in which one can differentiate the stress function to compute the local tangent stiffness. The straight-forward approach will be to simply differentiate the obtained stress function again. Since the stress function maps 2nd-order strain tensor to 2nd-order stress tensor, one can choose either the forward-mode or the reverse-mode for differentiation. Both will be computationally equivalent. The code-snippet for computing tangent stiffness using reverse-mode differentiation is given in Listing 2. This method explicitly computes and stores the local tangent stiffness in a matrix form which then must be multiplied with the incremental strains $\delta\boldsymbol{e}$ to get the incremental stresses $\delta\boldsymbol{\sigma}$. This results in extra memory usage and two operations.

Automatic differentiation and especially the JAX library provides two other methods, *i.e.* the jvp and linearize functions, to ease the calculation of directional derivatives. In both approaches, the local tangent stiffness can be represented as a push-forward function (or a linear operator) that maps incremental strains $\delta\boldsymbol{e}$ lying in the tangent space of $\boldsymbol{e}$ to incremental stresses $\delta\boldsymbol{\sigma}$ lying in the tangent space of the stress domain. This facilitates the direct computation of incremental stress as a Jacobian-vector product [28,29]. The jvp function (an example code-snippet is shown in Listing 3) internally constructs the push-forward map and then applies it to the incremental strains to compute the incremental stresses. Representing $\mathbb{K}$ as a push-forward map rather than storing it explicitly in matrix form can significantly reduce the memory usage of FFT-based methods. However, creating this push-forward map with each function call can prove computationally expensive, particularly when utilized in the inner solver, like the conjugate gradient, of a Newton–Krylov solver.

A computationally more efficient approach involves computing the local tangent stiffness around a fixed point, such as $\boldsymbol{e}$ from the previous converged step, within the Newton–Raphson scheme. This local stiffness can then be used to calculate incremental stresses during iterations in the inner solver. We utilize the JAX linearize function to generate the push-forward map for $\mathbb{K}$ at a specified strain point and store it as a function, which is then utilized to compute the incremental stresses. An example implementation is given in Listing 4.

```
# create a function to compute stresses
compute_stresses = jax.jacrev(strain_energy)

# computing the incremental stresses as given in Eq 4
dsigma = jax.jvp(compute_stresses, (prev_strain,), (dstrain,))[1]
```

Listing 3: Automatic differentiation to compute incremental stresses using jvp function.

```
# create a function to compute stresses
compute_stresses = jax.jacrev(strain_energy)

# compute the tangent stiffness function linearized around previous strain
f_jvp = jax.linearize(compute_stresses, prev_strain)[1]

# compute the incremental stresses as given in Eq 4
dsigma = f_jvp(dstrain)
```

Listing 4: Automatic differentiation to compute incremental stresses using linearize function.

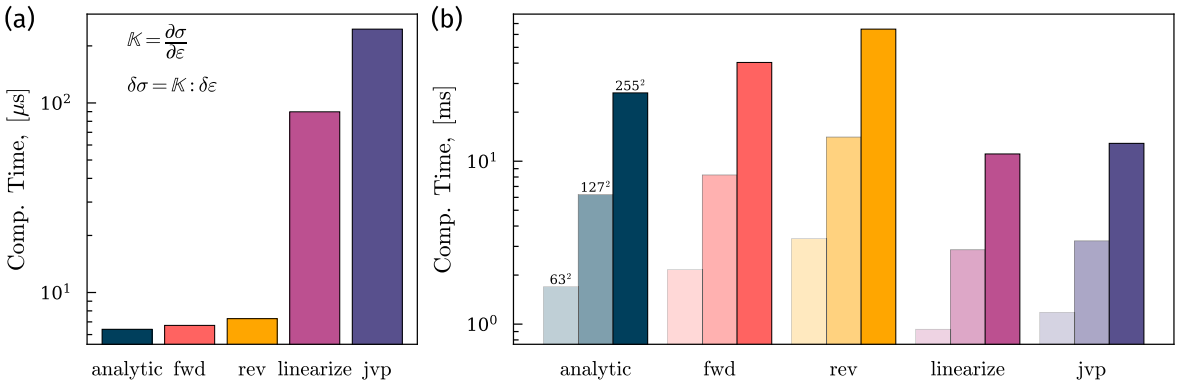

Fig. 2. Computational efficiency of AD methods for the computation of incremental stresses from tangent stiffness. (a) Computational time for calculation on a single grid point. We consider a single function evaluation. (b) Computational time for a varying number of grid points. We compute the incremental stresses simultaneously overall the grid points.

We examine the computational efficiency of these three methods to determine the incremental stresses from tangent stiffness on two scenarios: calculating incremental stresses for a single pixel (see Fig. 2a) and for a grid of pixels (see Fig. 2b). We consider a linear elastic material law for which the strain energy is given as $\psi = \lambda \mathrm{tr}(\boldsymbol{\varepsilon})^2 + \mu \mathrm{tr}(\boldsymbol{\varepsilon} : \boldsymbol{\varepsilon})$ where λ and μ are Lame's parameters. For reference, we show the computational time when the local stiffness tangent and the incremental stresses are computed from the analytical expression. We observe that the calculation time for a pixel in forward or reverse mode, which are the two options for the first approach, is nearly as fast as using the analytical method (see Fig. 2a). A small overhead occurs when using automatic differentiation due to the combined computation of derivatives and function values. However, employing the push-forward map via the linearize function or the jvp function increases the overall computation time for a single pixel (see Fig. 2a). Although direct evidence for this cost increase is unavailable, we speculate that it results from constructing the push-forward map. When considering the computation time of the incremental stresses on a pixel grid (*e.g.* 63^2, 127^2, 255^2), we observe that all the tested methods are on par with the analytic approach (see Fig. 2b). Our results show that the jvp and linearize functions are slightly faster than the forward or reverse mode differentiation.

3.3. Automatic computation of homogenized stiffness tensor

We exploit the automatic differentiation technique to compute the homogenized stiffness tensor, which is the most significant aspect of computational homogenization [30,31] or the essential requirement for performing multiscale simulations [14,32]. The effective stiffness tensor is the partial derivative of macroscale stresses with respect to macroscale strains, which reads

$$\overline{\mathbb{C}} = \frac{\partial \overline{\boldsymbol{\sigma}}}{\partial \overline{\boldsymbol{e}}}, \quad \text{where} \quad \overline{\boldsymbol{\sigma}} = \frac{\int_\Omega \boldsymbol{\sigma}(\boldsymbol{e}(\boldsymbol{x}))\, d\Omega}{\int_\Omega d\Omega}, \tag{5}$$

where $\overline{\boldsymbol{\sigma}}$ is the volume average of local stresses within the RVE. In the FFT-based framework, two methods are used to determine the effective properties from the local stress and strain distribution: (a) the perturbation approach [14,33] and (b) the algorithmic consistent approach [34]. In the perturbation approach, each component of the effective tensor is computed by numerical differentiation (finite difference) between the perturbed state and the reference state. In the algorithmic consistent approach,

```
# function to solve RVE  using FFT approach for a given macroscale strain
def solve_microscale(macro_strain):
   # solve RVE (Eq 1) using Newton-Krylov solver as given in Appendix A
   # and get the microscale strain
   local_strain = netwon_krylov_solver(macro_strain)

   # compute the microscale stresses from microscale strain
   local_sigma = compute_stresses(local_strain)

   # compute the macroscale stresses from microscale stress
   macro_sigma = mean(local_sigma)
   return macro_sigma, (macro_sigma)

# create a function to compute homogenized tangent and stresses
compute_macro_tangent_stiffness = jax.jacfwd(
   solve_microscale, argnums=0, has_aux=True
)

# calling tangent function to get macroscale stresses and tangents
macro_tangent, macro_stress = compute_macro_tangent_stiffness(macro_strain)
```

Listing 5: Automatic derivation of homogenized stiffness tensor. Note that the function returns the macroscopic stress twice. This allows us to compute the macroscopic tangent stiffness as well as the macroscopic stresses together with a single function call.

the consistent tangent operator is computed by solving the Lippmann–Schwinger equation [34]. Both approaches necessitate an auxiliary solution method alongside the primary one to determine effective properties. For instance, the perturbation method requires computing finite differences for every tensor component, leading to three computations in a 2D case and six in a 3D case. Similarly, resolving the Lippmann–Schwinger equation requires an iterative solver. Consequently, both approaches can be computationally intensive, especially for large computational domains or for nonlinear constitutive laws. In contrast to these approaches, automatic differentiation significantly simplifies the procedure by employing a single operation for the computation of the homogenized tangent stiffness, as shown in Listing 5.

Here, we directly differentiate a function that takes macro-strain as input and outputs macro-stress (after solving Eq. (3)) to compute the homogenized stiffness tensor. This function encompasses numerous complex operations or function calls, such as forward-FFT transform, inverse FFT transform, and a Newton–Krylov solver, to accurately determine the local strain and stress distribution in an RVE (for details, refer [7,27]). The ability of automatic differentiation to decompose complicated arithmetic operations into primitive ones allows for exact differentiation. Moreover, it facilitates the simultaneous computation of stresses and tangent stiffness, thus eliminating the need for any secondary solution method.

4. Applications

Incorporating automatic differentiation in an FFT-based framework can, as described in Section 3, greatly simplify the computation of complex derivatives, which is essential for many mechanical problems. Here, we employ the AD-enhanced Fourier-Galerkin method on problems of varying complexity to demonstrate its capabilities. Specifically, we examine mechanical problems with intricate material models, bodies undergoing large-strain deformation, and multiscale simulations. We show how automatic differentiation allows easy differentiation of functions with complex mathematical operations such as FFT transform or with logical operations such as conditional/unconditional-loops, which simplifies the entire numerical framework. We evaluate the computational performance and accuracy of the AD-enhanced FFT-based method. We provide the Python code for all the examples at [35]. All simulations were performed as a serial job on a Linux system using a 4-core AMD EPYC™ 9654 CPU with 16 GB RAM and a clock speed of 2.4 Hz.

4.1. Automatic derivation of stress and tangent operator

We present two examples to demonstrate the applicability of automatic differentiation for the computation of stress and tangent operators for material laws with fundamentally different characteristics, *i.e.* hyperelastic and elastoplastic materials.

4.1.1. Example on hyperelastic material

With this example, we aim to show that the reliance of FFT-based frameworks on explicit formulations for stress and tangent operators can be minimized [5]. We consider *hyperelastic* materials, where the stresses and the tangent moduli are first- and second-order derivatives of the strain energy density functional, respectively. We consider the St. Venant–Kirchhoff material for which the energy density functional is given as

$$\psi = \frac{\lambda}{2}\mathrm{tr}(\boldsymbol{E})^2 + \mu\mathrm{tr}(\boldsymbol{E} : \boldsymbol{E}) \,, \tag{6}$$

where $\boldsymbol{E} = \boldsymbol{F} \cdot \boldsymbol{F}^{\mathrm{T}} - \boldsymbol{I}$ is the Green–Lagrange strain, $\boldsymbol{F}$ is the deformation gradient, λ and μ are Lame's parameters. As described in Section 3, we apply automatic differentiation to derive the stress function (the Jacobian of the energy density) and the tangent function (the Hessian of the energy density).

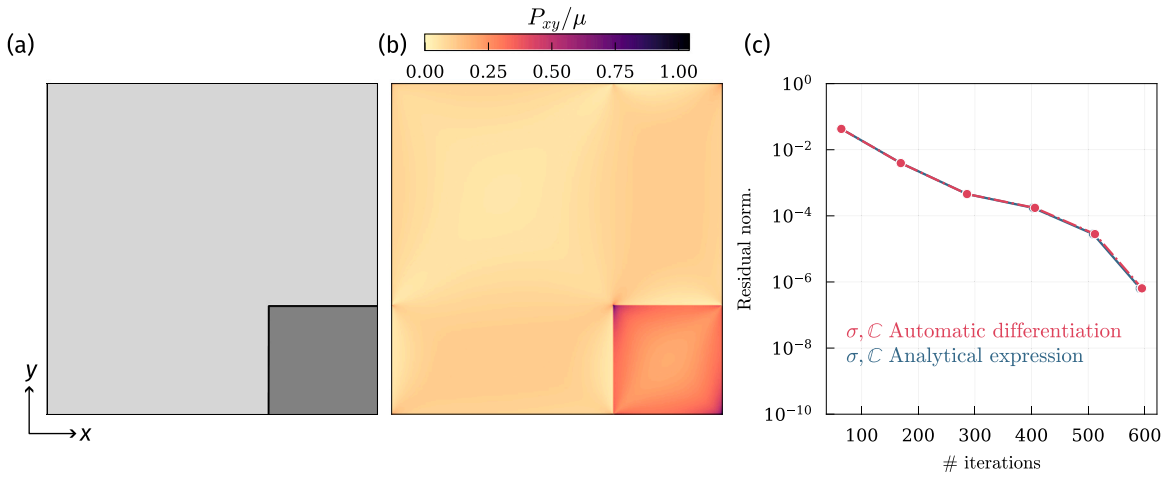

Fig. 3. Automatic computation of stress and tangent operator for a St. Venant–Kirchhoff material. (a) Schematic illustration of the microstructure of the considered RVE. (b) Local distribution of the first Piola–Kirchhoff stresses within the RVE. (c) Variation of the residual norm and the cumulative CG iterations for each NR iteration (shown as circles).

Foremost, we compare the AD implementation for St. Venant–Kirchhoff material with the implementation where the stress and tangent functions are computed from the analytical expression. Similar to [7], we consider a square RVE of dimension ℓ (see Fig. 3a) with a square stiff inclusion (of dimension $\ell/3$) at one of its corners. The RVE is subjected to a macroscopic shear loading in terms of $\overline{\boldsymbol{F}}$ which in Voigt notation is given as $[1, 1, 1+0.1\ell]$. We discretize the RVE into 299^2 grid points. An odd number of grid points is chosen to maintain the compatibility of the deformation gradient [4]. We solve the problem in one increment using a Newton–Krylov solver with a conjugate gradient (CG) solver for solving the linearized form of Eq. (2). The tolerance for the outer Newton–Raphson (NR) loop is 10^{-5}, and the tolerance for the inner loop of the CG solver is 10^{-8}. We compute the local distribution of the first-Piola Kirchhoff stresses within the RVE (see Fig. 3b) and the residual norm ($\delta \boldsymbol{F}/\|\overline{\boldsymbol{F}}\|$) for every NR iteration (see Fig. 3c). Here, we use JAX's jvp function (see Section 3.2) to compute the incremental stresses within the CG iterations. The AD implementation gives identical results to the analytical expression, as the residual norm and the iteration counts (cumulative CG iterations for every NR iteration) to reach convergence are the same. The computational times for the simulations with AD implementation and with analytical implementation are approximately the same, around 19.5 s. Since we express the tangent operator as a function rather than storing it as a matrix, the memory footprint of the AD-enhanced version of our implementation is roughly 1/2 of that of the analytical expression.

This example demonstrates that stress and the tangent operator can be easily derived from an energy functional for a material. This approach eliminates the need for FFT-based methods to rely on analytical expressions or numerical differentiation, thereby reducing computational cost (*i.e.*, memory usage) and simplifying the numerical implementation of sophisticated hyperelastic materials.

4.1.2. Example on elastoplastic material

Automatic differentiation is applicable to material laws in which the stresses and the tangent stiffness cannot be explicitly derived from the energy density. For instance, elastoplastic materials involve stress and tangent computations that depend on yielding criteria, plastic flow rule, and a return-mapping algorithm [36]. For demonstration purposes, we consider a heterogeneous RVE with hard circular inclusions embedded in a soft matrix (see Fig. 4a). We adopt an isotropic linear hardening model in a small-strain setting with each phase exhibiting distinct hardening exponents and yield stresses. The RVE is subjected to a macroscopic biaxial loading, in terms of $\overline{\boldsymbol{\epsilon}}$ which in Voigt notation is given as $[10^{-2}\ell, 10^{-2}\ell, 0]$. The RVE is discretized into 199^2 grid points. The tangent stiffness operator in elastoplastic materials depends on the elastic strain, the plastic strain, the type of yield surface, and the normals to this yield surface [36,37]. We selected J_2-plasticity for both materials. We assume a linear isotropic hardening law with the yield criterion given as $\Phi = \sqrt{3/2\boldsymbol{\sigma}_{\text{dev}} : \boldsymbol{\sigma}_{\text{dev}}} - \sigma_y$ where $\boldsymbol{\sigma}_{\text{dev}}$ is the deviatoric stress and σ_y is the yielding stress. The phase contrast between the yield stress and hardening moduli of the two phases is 2. The analytical expression for the consistent constitutive tangent is equal to the elastic tangent when the trial state is elastic and otherwise given as

$$\mathbb{K} = \frac{\partial \boldsymbol{\sigma}^{t+\Delta t}}{\partial \boldsymbol{e}^{t+\Delta t}} = \mathbb{C}_{\text{elas}} - \frac{6\mu^2 \Delta \epsilon_p}{\text{tr}\sigma_{\text{eq}}} I_d + 4\mu^2 \left(\frac{\Delta \epsilon_p}{\text{tr}\sigma_{\text{eq}}} - \frac{1}{3\mu + H} \right) \text{tr}N \otimes \text{tr}N \ .$$

where $\mathbb{C}_{\text{elas}}$ is the elastic stiffness matrix, μ is the shear modulus, ϵ_p is the cumulative plastic strain, H is the hardening modulus, σ_y is the yielding stress and N is the direction of the deviatoric strain. Automatic differentiation calculates the tangent stiffness on the fly by differentiating the stresses with respect to the strains. Unlike the previous case of hyper-elastic material, where the local stiffness tangent operator is computed as a double derivative of an energy density function, here we first implement a Python function to compute the correct stress state. The stress state is calculated using the elastic predictor–corrector approach (for details,

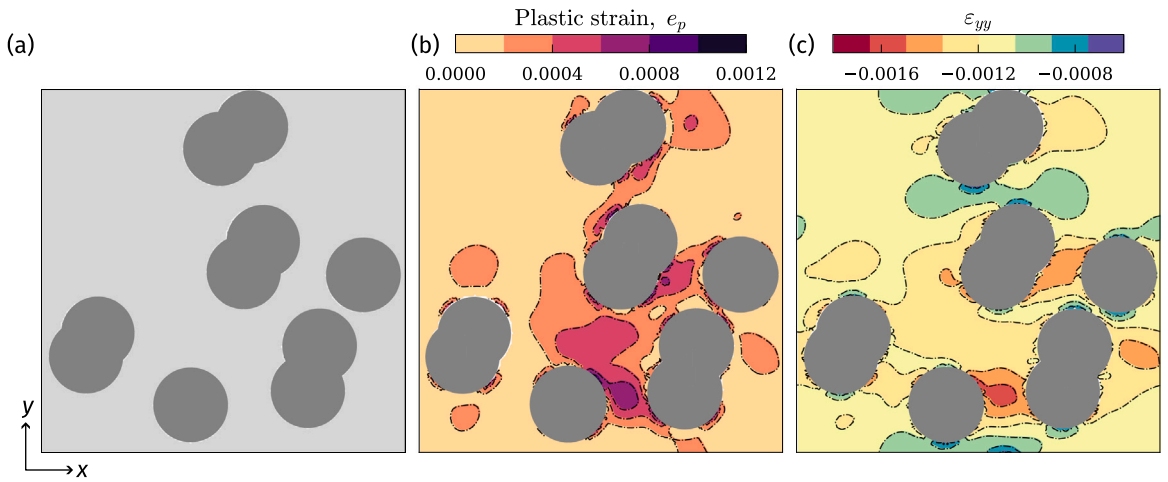

Fig. 4. Automatic computation of tangent operator for J_2-plasticity. (a) The microstructure of the considered RVE. The hard circular inclusions have a higher yielding strength than the matrix. (b) Local distribution of cumulative plastic strain ϵ_p and (c) local distribution of strain within the RVE. The color map is the result of AD-enhanced FFT simulations, and the dashed contour lines represent the results of the analytical implementation of the elastoplastic material.

see Appendix B), and a return mapping algorithm [36,37]. Then, this function is automatically differentiated with respect to the input. Here, within the NR loop, we use JAX's linearize function (see Section 3) to create the push-forward map of tangent stiffness at a fixed strain value and then use it to calculate incremental stresses within the CG iterations. For details on the numerical implementation of an elastoplastic material for an AD-enhanced FFT-based method, see the pseudo-code in Appendix B.

We observe that the local distribution of the plastic strain and the elastic strain (see Fig. 4b and Fig. 4c) for the AD-computed local tangent stiffness operator $\mathbb{K}$ matches the strains computed with the analytical expression of $\mathbb{K}$ (shown with dashed contour lines in Fig. 4). We note that the computation time for the AD implementation and the analytical implementation was approximately the same, around 0.25 s. Through this example, we show that a local stiffness operator can also be achieved for the cases where the stresses are not a direct derivative of the energy functional. This increases the applicability of automatic differentiation to various other material models where the calculation of stresses involves complex computational functions or operations.

4.1.3. Computational performance

We present scalability and memory usage (see Fig. 5) for the two cases considered above, *i.e.* hyperelastic and elastoplastic. We report the total computational time required for all stages, starting with the tangent stiffness function derivation and proceeding through Newton–Krylov iterations, up to convergence. For the hyperelastic case, we applied the push-forward map, utilizing the jvp function, to calculate incremental stresses within each iteration of the Conjugate Gradient solver. As discussed in Section 3.2, constructing the push-forward map in an inner loop incurs overhead, making the AD-enhanced FFT slightly more expensive than the conventional FFT implementation with analytic expressions. Our scaling analysis shows that even with a high number of pixels, the difference in computation time remains minimal (see Fig. 5a). However, memory consumption is reduced by approximately half since the tangent moduli are not stored (see Fig. 5b).

In the elastoplastic case, we computed and stored the push-forward map during Newton–Raphson iterations using the linearize function. The computation time here matches the standard method using analytic expressions because automatic differentiation overhead occurs only in the Newton–Raphson steps (see Fig. 5a). However, unlike before, the AD-enhanced method uses more memory compared to standard FFT, probably due to the need to store the linearized function (see Fig. 5b).

Finally, we note that while we showed in Fig. 2b that analytic computations cost more than the push-forward map, these analytic computations become cheaper than linearize and jvp functions for repeated executions. This occurs because the analytic expression, implemented in Numpy, is optimized for multiple uses, whereas the slight overhead of using automatic differentiation makes linearize or jvp marginally computationally more expensive.

4.2. Computational homogenization

We apply the AD-enhanced FFT-based framework to perform computational homogenization of an RVE. As noted earlier, current approaches, such as numerical differentiation or the algorithmic consistent approach, require an auxiliary computationally intensive method to compute the homogenized stiffness tensor of an RVE. Here, we present two examples to illustrate the efficacy and efficiency of automatic differentiation in computing effective properties.

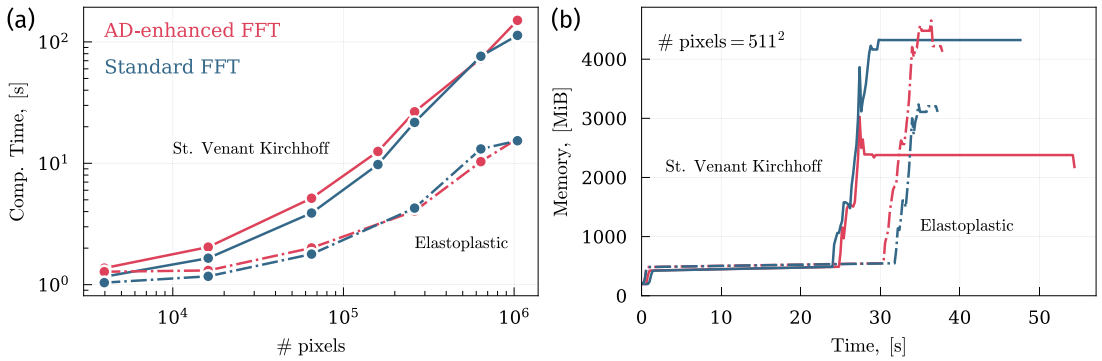

Fig. 5. Computational efficiency of AD-enhanced FFT method. (a) Scaling of computational time with increase in the number of grid points for two different constitutive laws. Computations with automatic differentiation and analytical expression are shown in red and blue, respectively. For hyperelastic material, we use JAX's jvp function to compute the incremental stresses and for elastoplastic material, we use JAX's linearize function. (b) Memory consumption for the two material laws for the case with 511^2 grid points.

Table 1
Components of the effective elastic tensor $\overline{\mathbb{C}}$ as computed numerically by automatic differentiation using FFT solvers compared against Eshelby's solution for a stiff inclusion embedded in a compliant infinite matrix.

	$\overline{\mathbb{C}}_{xxxx}$	$\overline{\mathbb{C}}_{yyyy}$	$\overline{\mathbb{C}}_{xyxy}$
Analytical	4.037	4.037	1.011
Numerical	4.035	4.035	1.010

4.2.1. Validation with eshelby's problem

First, we consider the Eshelby problem [30] as a basic example to compare the numerical result against the analytical solution. Similar to [34], we model a periodic micro-structure with a stiff circular inclusion embedded centrally in a compliant matrix. We compare the numerically computed effective elastic tensor $\overline{\mathbb{C}}$ with Eshelby's solution. We consider a very low volume fraction (0.7%) of the inclusion to match the infinite medium assumption of Eshelby, similar to [34]. We consider a linear elastic material law for both the matrix and the inclusion. Thus, the strain energy is $\psi = \lambda \mathrm{tr}(\boldsymbol{\varepsilon})^2 + \mu \mathrm{tr}(\boldsymbol{\varepsilon} : \boldsymbol{\varepsilon})$. The material parameters chosen for the matrix are $\lambda_m = 2$ GPa and $\mu_m = 1$ GPa, and for the stiff inclusion are $\lambda_i = 10$ GPa and $\mu_i = 5$ GPa. We compute the homogenized or effective elastic tensor $\overline{\mathbb{C}}$ of the material for a macro-strain of $\overline{\varepsilon} = [10^{-2}, 10^{-2}, 0]$. We observe that the results from the AD implementation are close to the analytical results (see Table 1). We note that for a computational grid of 199^2, the total computation time was around 4 s

4.2.2. Example on architected materials

We apply the AD-enhanced computational homogenization method to an RVE (of dimension ℓ) with intricate microstructure. We chose architected materials as our example. Architected materials are composed of two or more distinct phase materials arranged in a specific pattern that results in exceptional mechanical properties [38–40]. We examine a two-phase linear elastic architected material as illustrated in Fig. 6a, comprising a compliant phase and a stiff phase, where the stiff phase is arranged in a hexagonal pattern. We vary the phase contrast $\mathcal{X} = \mathrm{compliant/stiff}$ from 1 to 10^{-4} where a value of 1 represents a homogeneous material and value $\ll 1$ approximates a lattice metamaterial [41]. We apply a macroscopic compressive strain $\overline{\varepsilon}$ in x and y-directions under a small-strain setting. The macroscopic strain tensor in Voigt notation is given as $[-10^{-3}\ell, -10^{-3}\ell, 0]$. For the case where $\mathcal{X} = 1$, *i.e.* a homogeneous material, the computed macro stiffness values match the analytical solution (see Fig. 6c). With increasing phase contrast, however, the macroscopic stiffness reduces, as expected, and the obtained values ($\overline{\mathbb{C}}_{xxxx} = 0.51, \overline{\mathbb{C}}_{yyyy} = 0.42$ and $\overline{\mathbb{C}}_{xyxy} = 0.059$) saturates to the values that are indicative of a lattice metamaterials. The values obtained are similar to the values ($\overline{\mathbb{C}}_{xxxx} = 0.57, \overline{\mathbb{C}}_{yyyy} = 0.52$ and $\overline{\mathbb{C}}_{xyxy} = 0.15$) obtained from a full-field Finite Element homogenization under affine displacement boundary conditions (for details, please refer to Appendix C). The properties obtained under periodic boundary conditions are expected to be lower than the properties obtained under affine displacement boundary conditions [42].

While not depicted in Fig. 6 we calculated the homogenized properties for a phase contrast of 10^{-10} to approximate zero stiffness in the compliant phase to machine precision. The obtained homogenized values remain consistent, illustrating that the AD-enhanced approach is not affected by significant contrast (approaching zero) in phase properties.

4.3. Application to multiscale problem

In this section, we showcase how the ease of computation of homogenized stiffness tensor allows seamless integration of the AD-enhanced FFT-based framework with an additional solver. To this end, we explore a multiscale framework where the macroscale

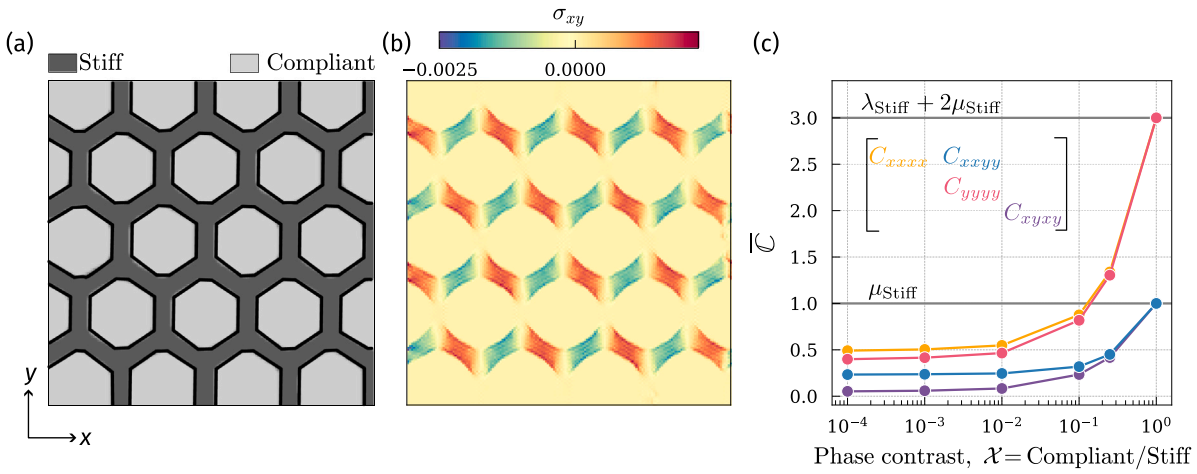

Fig. 6. Computational homogenization of an architected lattice material. (a) The architected RVE with a stiff phase arranged in a hexagonal pattern within a compliant phase. (b) Local distribution of shear stresses within the RVE for a phase contrast of $\mathcal{X} = 0.5$ and a macroscopic strain $\overline{\boldsymbol{e}} = [-10^{-3}\ell, -10^{-3}\ell, 0]$. (c) Components of the macroscale constitutive tangent stiffness as a function of phase contrast $\mathcal{X}$. The dark gray horizontal lines show the stiffness component for the homogeneous case *i.e.* $\mathcal{X} = 1$.

problem is tackled using the Finite Element Method (FEM), while the microscale problem employs the FFT-based approach. This example demonstrates the flexibility of the AD-enhanced FFT-based method by integrating it with a FEM solver.

The macroscale problem involves a beam with dimensions $L \times H$, where the left end is fixed in both the x and y directions. A displacement of $10^{-2}L$ is applied to the right edge along the y-axis (see Fig. 7a). At the microscale, we use the same composite RVE as in Section 4.2 but with the St. Venant–Kirchhoff material model to capture the geometric nonlinearity (see Eq. (6)). A phase contrast of 0.01 is chosen between the compliant and stiff phases to represent voids, thus modeling a hexagonal-lattice metamaterial. We solve the macroscale problem in reference configuration with $\boldsymbol{P}$ and $\boldsymbol{F}$ as the stress–strain measures. The weak form at the macroscale thus reads,

$$\int_V \underbrace{\overline{\mathbb{C}}(\boldsymbol{F}) : \overline{\boldsymbol{F}}}_{\overline{\boldsymbol{P}}} : \delta\overline{\boldsymbol{F}} \cdot \mathrm{d}V = 0, \quad \text{where} \quad \overline{\mathbb{C}} = \frac{\partial \overline{\boldsymbol{P}}}{\partial \overline{\boldsymbol{F}}}, \tag{7}$$

where the $\delta\overline{\boldsymbol{F}}$ is the virtual deformation gradient (or test function) and the $\overline{\mathbb{C}}$ is the homogenized tangent operator computed from microscale. We discretize the beam into 16 quadrilateral finite elements (16 Gauss points) as shown in Fig. 7b, where each Gauss point is linked to the RVE with 169^2 voxels. The deformation gradient $\overline{\boldsymbol{F}}$ at each Gauss point within the macroscale domain is transferred as input to the corresponding RVE associated with that specific Gauss point. Similar to the example in Section 4.2, we determine the homogenized tangent operator $\overline{\mathbb{C}}$ through automatic differentiation and pass it together with the average first-Piola Kirchhoff stress $\overline{\boldsymbol{P}}$ to the FEM solver (see Fig. 7a). We note that due to a non-symmetric deformation gradient, one needs to use a full deformation gradient (4 components) [33]. This means that $\overline{\mathbb{C}}$ is a 4×4 matrix. We perform the simulation in one displacement increment under static conditions and use FEniCSx to manage the FEM component [17,24].

At the microscale, we apply the Newton–Krylov solver (with an outer tolerance of 10^{-5} and an inner tolerance of 10^{-6}) to accurately capture the nonlinearity within the lattices. At the macroscale, a separate Newton–Raphson solver (with a tolerance of 10^{-8}) is used to ensure proper convergence. We observe that the chosen contrast of 0.01 between the two phases sufficiently replicates the voids, which is demonstrated by the stresses in the compliant phases dropping to 0, as shown by the stress profile across the cross-section in Fig. 7c. We report that this simulation takes approx. 20 min in total (for roughly 1 million degrees of freedom in total). A large part of the computational time is spent in the CG-solver because of the high contrast in the phase properties. This is a known issue for the Fourier-Galerkin approach [43], which can be improved using pre-conditioned solvers [6]. A traditional FE-FFT framework [14], such as finite-difference-based computation of homogenized stiffness tensor, would require 4 operations per Gauss point, one for each component of non-symmetric deformation gradient, to compute $\overline{\mathbb{C}}$. Conversely, the AD-enhanced multiscale framework is notably advantageous in this scenario, as it demands only one operation per Gauss point for the same.

4.4. Beyond stress and tangent computations

The application of automatic differentiation to compute derivatives extends beyond stress and tangent calculations. Several challenges in solid mechanics require derivatives that are difficult to compute, such as evaluating the sensitivity of a complex computational model to its input parameters. In this section, we illustrate two use cases where automatic differentiation can aid in computing sensitivities of complex computational models such as in uncertainty quantification and topology optimization. Here, we employ the FFT-based method, as presented in earlier sections, as the complex computational model.

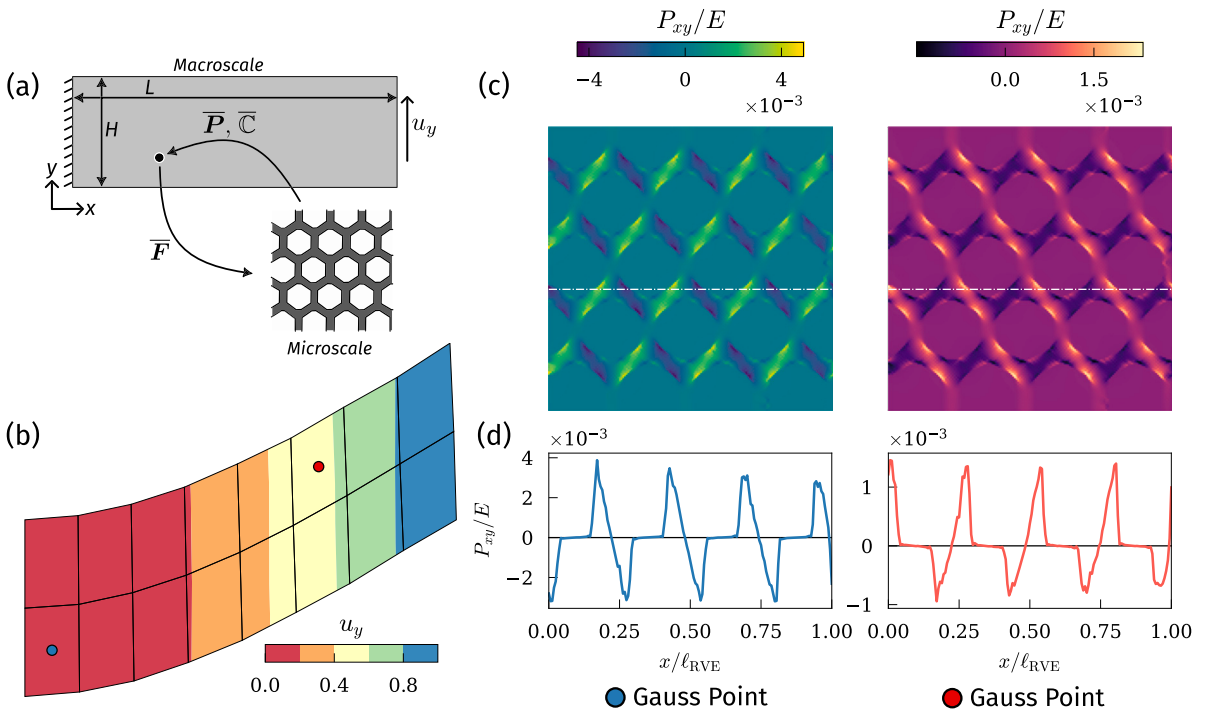

Fig. 7. Multiscale simulation of a hyperelastic lattice material. (a) Schematic illustration of the beam at the macroscale and the hexagonal lattice structure at the microscale. (b) The deformed state of the beam for an applied displacement of $10^{-2}L$. The color corresponds to the macroscale displacement along the y-axis. The finite element discretization is shown by black lines and two Gauss points are highlighted by colored (red and blue) points. (c) The local shear stress distribution within the RVEs associated with the 2 Gauss points highlighted in (b). (d) Normalized shear stress profile along line highlighted by white dashes in (c).

Table 2
The expected and the standard deviation values of the elasticity modulus and the Poisson ratio for the inclusion (E_{i} and ν_{i}) and the matrix (E_{m} and ν_{m}).

	E_{i} [MPa]	ν_{i}	E_{m} [MPa]	ν_{m}
$\mathbb{E}$	5.7	0.386	0.57	0.386
$\sqrt{\mathrm{Cov}}$	0.01	0.03	0.01	0.03

4.4.1. Uncertainty quantification

We investigate how automatic differentiation can assist in optimizing uncertainty quantification within deterministic processes. Uncertainties may arise from variations in material parameters or inconsistencies in material arrangement. A common method to propagate these uncertainties (*i.e.*, calculate the expected and covariance values of the output) employs Monte Carlo simulations, which, while being effective, are computationally demanding. When the input uncertainties are small, the expected value of the output $\mathbb{E}[y]$ and its covariance $\mathrm{Cov}[y]$ can be approximated using the Taylor series expansion. For example, the second-order approximation for the expected value and the first-order approximation for the variance can be expressed as

$$\mathbb{E}[y] \approx f(\mathbb{E}[\boldsymbol{x}]) + \frac{1}{2} C_{ij} \frac{\partial^2 f(\boldsymbol{x})}{\partial x_i \partial x_j}\Big|_{\boldsymbol{x}=\mathbb{E}[\boldsymbol{x}]}, \quad \mathrm{Cov}[y] \approx C_{ij} \frac{\partial f(\boldsymbol{x})}{\partial x_i}\Big|_{\boldsymbol{x}=\mathbb{E}[\boldsymbol{x}]} \frac{\partial f(\boldsymbol{x})}{\partial x_j}\Big|_{\boldsymbol{x}=\mathbb{E}[\boldsymbol{x}]}, \tag{8}$$

where C_{ij} is covariance matrix of input parameters, $\mathbb{E}[\boldsymbol{x}]$ is the expected value of the input variables $\boldsymbol{x}$, $\partial f/\partial x_i$ and $\partial^2 f/\partial x_i \partial x_j$ are the first and the second order derivative of a objective function with respect to inputs x_i. Typically, the covariance matrix of the input variables is known and therefore the task is to find the derivatives. Here, we employ automatic differentiation to derive the first derivative and the second derivative of the objective function f with respect to its input. To this end, we consider the same two-phase material as in Section 4.1.1. The material is *hyperelastic* and the expected values of the material parameters and their variance are given in Table 2. We assume that the material parameters are normally distributed.

To simplify the analysis, we consider a function f that accepts material parameters as inputs, determines static equilibrium for a macroscopic deformation gradient $\overline{\boldsymbol{F}} = [0, 0, 0.1]$, and then outputs the total strain energy of the two-phase material. Given that f maps a 4-dimensional space to a scalar space, we use reverse-mode AD to efficiently obtain the first derivative. We differentiate this newly derived function using forward-mode to compute the second-order derivative. Since the function that evaluates the first-order derivative maps a 4-dimensional space to another 4-dimensional space, forward-mode proves to be efficient.

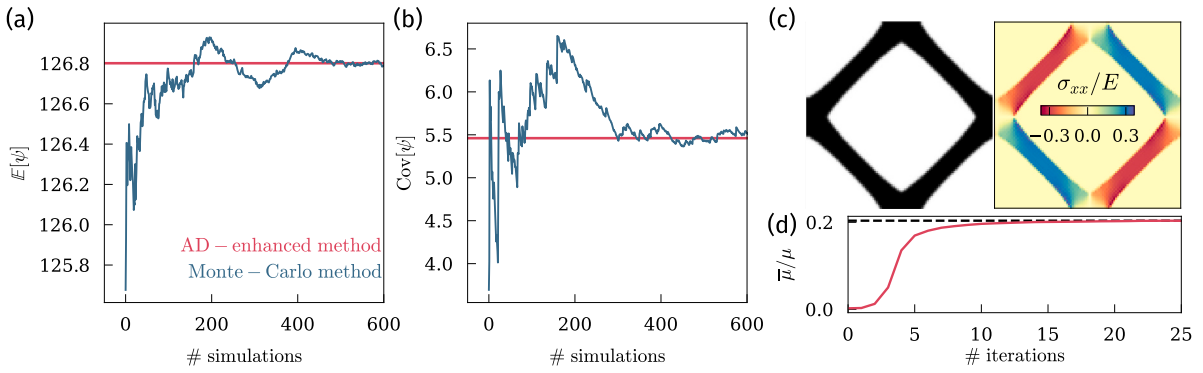

Fig. 8. Sensitivity calculation using automatic differentiation for uncertainty quantification and topology optimization (a) Computation of expected value of strain energy for two-phase hyperelastic material using the 2nd-order approximation via AD (red line) and using the Monte Carlo approach on 600 samples as a reference (blue curve). (b) The covariance in strain energy computed through first-order approximation via AD (red) and through the Monte Carlo approach (blue) on 600 samples as a reference. (c) Microstructure optimized for a maximal homogenized shear modulus at a volume fraction of 30% (left), and the associated stress distribution (right). (d) The value of the homogenized shear modulus $\overline{\mu}$ as it evolves over optimization iterations. The value is normalized by the shear modulus of the parent material.

We assume that the covariance C_{ij} between the different parameters is known and is computed by drawing 600 samples randomly from a normal distribution. Using the first-order function, the second-order function and the known covariance matrix, we calculate the expected value of the total strain energy and its variance. We note that we only need 2 runs, first for the first-order derivative and second for second-order derivative. The results thus obtained are shown in Fig. 8a &b. For reference, we also determine the expected values and variance through Monte-Carlo simulations using the drawn sample of 600. We observe that the AD-computed values match closely with the Monte-Carlo results (see Fig. 8)a &b. Additionally, the total time required for the Monte-Carlo method to achieve a saturated value was approximately 7 min, whereas the AD-computed uncertainties took around 20 s.

4.4.2. Microstructure optimization

Here, we utilize AD-enhanced FFT-based method for topology optimization and use automatic differentiation to simplify the sensitivity calculation. Topology optimization is frequently utilized to design microstructures with tailored homogenized properties, such as achieving the highest bulk modulus or shear modulus with a specific volume fraction. The microstructure, represented as a density field $\rho(\boldsymbol{x})$ varying from 1 (material) to 0 (void), is optimized to achieve the desired properties under specified constraints. The optimization problem thus reads:

$$\begin{aligned} \underset{\rho(\boldsymbol{x})}{\text{minimize}} \; : \;& f(\rho(\boldsymbol{x}), \overline{\boldsymbol{\varepsilon}}) \\ \text{s.t} \; : \;& \mathcal{F}^{-1}\{\widehat{\mathbb{G}}(\boldsymbol{\xi}) : \widehat{\boldsymbol{\sigma}}(\boldsymbol{\xi})\} = \mathbf{0} \, , \\ : \;& \textstyle\int \rho(\boldsymbol{x}) d\Omega / \int d\Omega = \vartheta \, , \\ : \;& 0 \leq \rho(\boldsymbol{x}) \leq 1, \forall \boldsymbol{x} \in \Omega \end{aligned}$$

where f is the objective function to be minimized. The first condition ensures static equilibrium. The second condition ensures the imposed volumetric constraint and the last condition ensures that the density field $\rho(\boldsymbol{x})$ remains within physical limits *i.e.* $\rho(\boldsymbol{x}) \in [0, 1]$. Irrespective of the solution strategy (optimality criteria [44], MMA [45] or Lagrange multipliers) used for the aforementioned optimization problem, information about the gradients of both the objective function and the constraints with respect to the input is required. Recently, automatic differentiation has been adopted to simplify the computation of these gradients [46]. Building on this concept, here we use automatic differentiation to determine both homogenized properties (*i.e.* homogenized shear modulus $\overline{\mu}$) and their sensitivities to the density field within a FFT framework.

Similar to [46–48], we consider the classic microstructure optimization problem of maximizing the homogenized shear modulus $\overline{\mu}$ of a microstructure (*i.e.* objective function $f(\rho(\boldsymbol{x})) = -\overline{\mu}$) with a given volume fraction. We examine a square domain under macroscopic strain $\overline{\boldsymbol{\varepsilon}}$. The material behaves as linear elastic with a Young's modulus $E = 1$ MPa and a Poisson ratio $\nu = 0.3$. Additionally, we enforce a volumetric constraint of 0.3. We follow the optimality criteria method [44] to solve the above constrained optimization problem. This method requires the sensitivity of the objective function $\partial f / \partial \rho$ to update the density field ρ in successive iterations. Since the function f maps a high-dimensional space (ρ) to a scalar value ($\overline{\mu}$), we use reverse-mode AD to compute the sensitivities. For each update of the density field, we use the AD-enhanced FFT framework to calculate the homogenized stiffness tensor $\overline{\mathbb{C}}$ of the microstructure for a macroscopic strain of $\overline{\boldsymbol{\varepsilon}} = [0, 0, 1.]$ under plane-stress conditions, as discussed in Section 4.2.

The optimized microstructure (see Fig. 8b) and the converged shear modulus $\overline{\mu} \approx 0.075$ match well the results from the literature [46,47,49]. Thus, integrating AD-enhanced computation of homogenized properties with AD-derived sensitivities streamlines the implementation of optimization problems. This approach allows for the simultaneous calculation of all homogenized components, differing from previous studies that required solving the static equilibrium for three separate test strains [50]. Furthermore, integrating simultaneous homogenization of the microstructure and computation of sensitivities within an FFT-based framework facilitates the optimization of computationally intensive cases.

5. Discussion & outlook

The integration of automatic differentiation (AD) with Fast-Fourier Transform (FFT) techniques can considerably simplify the numerical implementation of complex solid mechanics problems. A key advantage is the ease of deriving stress and tangent operators, thereby minimizing human error and enhancing precision. This work combines automatic differentiation with the Fourier-Galerkin method, whose variational structure facilitates straightforward integration of automatic differentiation. Here, we investigated the potential of automatic differentiation in FFT-based methods for solid mechanics, demonstrating how the automatic derivation of stress and tangent operators enables modular FFT approaches. This strengthens the use of FFT-based methods in more advanced models, particularly in multiscale scenarios where computing homogenized tangent stiffness has traditionally been a significant challenge. While we have emphasized potential applications in Section 4, several other research areas could also greatly benefit from combining automatic differentiation with FFT-based methods.

A notable application in which we expect AD-based FFT-based methods to be beneficial is computational inelasticity. As shown in Section 4.1, automatic differentiation greatly simplifies the extraction of the local tangent stiffness for inelastic models and is computationally equivalent to analytical expressions. Therefore, one can leverage automatic differentiation for complex plasticity models such as multi-yield surface J_2-plasticity [51], Simo-plasticity [36] or crystal plasticity [52,53] for which extracting the tangent stiffness is intricate and often relies on numerical differentiation [33]. Automatic differentiation can also streamline the return-mapping algorithm by boosting the robustness of trial-state projections to the yield surface [36,37]. Furthermore, multiphysical problems (or coupled problems) offer another promising avenue for the application of AD-enhanced FFT techniques [27,54,55]. For example, automatic differentiation can support the creation of coupled or staggered operators, leading to more precise and efficient solutions for such problems. As discussed in [8], linearizing a thermo-mechanical problem requires four-tangent stiffness matrices; using automatic differentiation, one can greatly simplify this number to 1 or 2. Additionally, expressing them as operators can significantly reduce the memory footprint. We demonstrated that employing AD-derived operators for the St. Venant–Kirchhoff material model, which has a straightforward analytical form (with only a few mathematical operations) for tangent stiffness, can halve memory usage.

We showed that automatic differentiation capabilities go beyond just computing stress and tangent values in an FFT-based framework. In Section 4.4, we utilized an AD-enhanced FFT-based method for topology optimization, where an accurate and efficient computation of sensitivity is necessary for a correct and fast optimization. Automatic differentiation simplifies the calculation of the sensitivity by directly differentiating the objective function with respect to the input parameters, thus eliminating the need for adjoint calculations [56]. Likewise, the high computational cost associated with uncertainty quantification, often due to the employed Monte Carlo approach, can be significantly reduced by using higher-order approximations to calculate expected values and variances of an output parameter. Automatic differentiation enables computing higher-order derivatives in a single simulation, thereby significantly reducing the computational cost. Furthermore, by broadening AD-FFT techniques to encompass second-order homogenization and strain gradient models, one can boost the precision and predictive power of multiscale simulations, thereby expanding the limits of computational mechanics and materials science. This adaptability highlights the potential of automatic differentiation in enhancing FFT-based methods for numerous mechanical applications.

The biggest challenge in adding AD techniques to existing codes/libraries is selecting the appropriate AD framework. Many AD frameworks are language-specific, so it is often easier to do so if the code is already in that language. For example, transforming existing Python-based code to work with differentiable frameworks such as JAX is generally straightforward (as is evident in the code provided at [35]). JAX's functional programming makes it easy to fit into scientific workflows with minimal changes. In contrast, modifying C++ code to function with AD frameworks such as ADOL-C [57], autodiff [58] or Sacado [59] typically requires more changes to handle automatic differentiation structures. However, these C++ libraries, built for performance, excel in tasks with high performance requirements, unlike Python-based framework such as JAX or PyTorch, which incur a slight computational overhead. Although frameworks such as ADOL-C and Sacado enable efficient automatic differentiation for basic arithmetic and algebra operations, allowing the derivation of stresses or tangent moduli from energy density functions, they lack, to the best of authors' knowledge, native support for FFT differentiation. Therefore, enabling automatic differentiation for FFT functions in these frameworks requires either manual implementation of custom adjoint routines or defining the chain rule for FFT operations. In contrast, JAX and PyTorch natively support automatic differentiation through FFTs, as shown in this paper, making them suitable for FFT-based methods. Furthermore, JAX compensates for AD overhead with its just-in-time compilation and advanced parallelization capabilities (CPUs and GPUs). Recently, frameworks such as Enzyme-AD [60,61], an experimental differentiable library that works across languages, enable automatic differentiation without altering the original code or libraries. This offers both flexibility, differentiation of FFT functions and high performance and, therefore, could be a great balance between ease of integration and computational efficiency.

Another important factor in selecting an AD framework is the range of differentiation techniques it offers, such as forward-mode and reverse-mode. As discussed in Section 2, both methods are crucial and can provide considerable efficiency depending on the function being differentiated. Although PyTorch provides AD support, mainly in reverse-mode, it does not offer as extensive support for both modes as JAX does (forward-mode, jvp function and linearize function). Therefore, the choice of a suitable framework for integrating automatic differentiation into existing code largely hinges on the user's specific requirements and preferences such as integration ease, flexibility, computing efficiency and support for parallelization. In this research, we opted for JAX to augment FFT-based methods with automatic differentiation. Nonetheless, it is feasible to employ any available AD framework, as the principles addressed in this paper remain applicable.

Finally, in this paper, we employed the Fourier-Galerkin method for the FFT approach; however, the AD technique, as demonstrated here, is applicable to other FFT-based methods [1,62,63] as well.

6. Conclusion

In this paper, we investigated the use of automatic differentiation (AD) within the FFT-based framework for solid mechanics. We have shown that automatic differentiation allows the computation of stresses and tangent operators directly from the energy density functional, which makes FFT-based methods more accessible for complex hyperelastic materials. Furthermore, we have demonstrated that automatic differentiation allows differentiation of intricate tensor operations, FFT transforms, and Newton–Krylov solvers, which enable the computation of tangent operators for materials where local stiffness is not derivable from strain density, *e.g.*, elastoplastic materials. The incorporation of automatic differentiation into the FFT-based framework decreases the reliance on explicit expressions of stresses and tangent operators while preserving accuracy and computational efficiency. We further employed the AD-enhanced FFT-based method to simplify the computation of homogenized stiffness tensors, a notable computational bottleneck in multiscale problems. We show that the AD-enhanced FFT-based approach is applicable beyond stress and tangent calculations, extending to problems like uncertainty quantification and topology optimization by simplifying sensitivity computations. Our findings indicate that applying automatic differentiation within an FFT-based method is highly beneficial and greatly simplifies the implementation of complex solid mechanics problems, as is also evident by the simple code that we provide at [35]. We believe this research will encourage further studies using automatic differentiation within the FFT-based frameworks and will help in advancing the field of computational solid mechanics.

Code availability

The code [35] for the simulation is written in Python and is an extension of the code provided in [27].

CRediT authorship contribution statement

Mohit Pundir: Writing – original draft, Visualization, Software, Methodology, Investigation, Formal analysis, Conceptualization. **David S. Kammer:** Writing – review & editing, Supervision, Resources, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

DSK and MP acknowledge support from the Swiss National Science Foundation under the SNSF starting grant (TMSGI2_211655). We thank Jan Zeman and Antoine Sanner for fruitful discussions and Alessandra Lingua for providing the scans of the lattice structure. We also thank Luca Michel for his feedback on the manuscript.

Appendix A. Netwon-Krylov solver

Algorithm 1 Newton–Krylov Solver

For a given macroscopic strain $\overline{\boldsymbol{e}}$

$\boldsymbol{r}(\boldsymbol{x}) = \boldsymbol{\sigma}(\boldsymbol{e}(\boldsymbol{x})^i),\ \boldsymbol{e}(\boldsymbol{x})^i = \overline{\boldsymbol{e}}$ ▷ $\boldsymbol{\sigma}(\boldsymbol{\varepsilon}) = \mathbb{C} : \boldsymbol{\varepsilon}$

while true **do**

  solve : $\mathcal{F}^{-1}\{\widehat{\mathbb{G}}(\boldsymbol{\xi}) : \widehat{\boldsymbol{\sigma}(\delta \boldsymbol{e})}(\boldsymbol{\xi})\} = -\mathcal{F}^{-1}\{\widehat{\mathbb{G}}(\boldsymbol{\xi}) : \widehat{\boldsymbol{\sigma}(\boldsymbol{r})}(\boldsymbol{\xi})\}$ ▷ Linear iterative solver

  update : $\boldsymbol{e}(\boldsymbol{x})^{i+1} = \boldsymbol{e}(\boldsymbol{x})^i + \mathcal{F}^{-1}\{\widehat{\delta \boldsymbol{e}}(\boldsymbol{\xi})\}$

  if $\delta \boldsymbol{e}(\boldsymbol{x})/\|\overline{\boldsymbol{e}}\| <$ tol **then** break

  else

    $\boldsymbol{r}(\boldsymbol{x}) = \boldsymbol{\sigma}(\boldsymbol{e}(\boldsymbol{x})^{i+1})$

Appendix B. Computation of incremental stresses for an elastoplastic material

For a given total strain $\boldsymbol{\epsilon}^{t+\Delta t}$, we aim to find the stress state ($\boldsymbol{\sigma}$)and the cumulative plastic strain (ϵ_p). The calculation of the stress state and the cumulative plastic strain for J2 plasticity using predictor–corrector approach is given as,

$$
\begin{aligned}
\Delta\boldsymbol{\epsilon} &= \boldsymbol{\epsilon}^{t+\Delta t} - \boldsymbol{\epsilon}^{t} \\
\boldsymbol{\sigma}^{\mathrm{tr}}_{\mathrm{elas}} &= \mathbb{C}_{\mathrm{elas}} : (\boldsymbol{\epsilon}^{t}_{\mathrm{elas}} + \Delta\boldsymbol{\epsilon}) \\
\sigma^{\mathrm{tr}}_{\mathrm{eq}} &= \sqrt{\frac{3}{2}\boldsymbol{\sigma}^{\mathrm{tr}}_{\mathrm{dev}} : \boldsymbol{\sigma}^{\mathrm{tr}}_{\mathrm{dev}}} \\
\Phi &= \sigma^{\mathrm{tr}}_{\mathrm{eq}} - \underbrace{(\sigma_y + H p^{t})}_{\text{yield function}} \\
\Delta\epsilon_{\mathrm{p}} &= \frac{\langle\Phi\rangle_{+}}{3\mu + H}, \quad \text{where } \langle\star\rangle_{+} = \frac{1}{2}(\Phi + |\Phi|) \\
\boldsymbol{\sigma}^{t+\Delta t} &= \boldsymbol{\sigma}^{\mathrm{tr}}_{\mathrm{elas}} - 2\mu\Delta\epsilon_{\mathrm{p}}\frac{\boldsymbol{\sigma}^{tr}_{\mathrm{dev}}}{\sigma^{\mathrm{tr}}_{\mathrm{eq}}} \\
\epsilon_{\mathrm{p}}^{t+\Delta t} &= \epsilon_{\mathrm{p}}^{t} + \Delta\epsilon_{\mathrm{p}} \\
\boldsymbol{\epsilon}^{t+\Delta t}_{\mathrm{elas}} &= \boldsymbol{\epsilon}^{t}_{\mathrm{elas}} + \Delta\boldsymbol{\epsilon} - \Delta\epsilon_{\mathrm{p}}\frac{\boldsymbol{\sigma}^{tr}_{\mathrm{dev}}}{\sigma^{\mathrm{tr}}_{\mathrm{eq}}}
\end{aligned}
$$

The above calculation is implemented as Python function that takes the total strain at time $t + \Delta t$ and computes the total stress at time $t + \Delta t$. We then use the Jacobian-Vector product (using the jvp function of JAX) to compute the incremental stresses. The implementation for which is given as

```
# explicit implementation of a Python function to compute stresses
def compute_stresses(strain):
   # Compute the trial state stresses
   # evaluate yield surface, set to zero if elastic (or stress-free)
   # plastic multiplier, based on hardening law
   # apply return-map algorithm to project stresses to a feasible state
   return sigma
# computing the incremental stresses for an elasto-plastic material
dsigma = jax.jvp(compute_stresses, (prev_strain,), (dstrain,))[1]
```

Appendix C. Computation of homogenized properties for architected material using FE

We perform a full-field computational homogenization using Finite Element method. Unlike, FFT method where we have periodic boundary condition, here we apply Kinematic Uniform Boundary Conditions (KUBC) for computational homogenization [42,64]. It assumes that the strain field is uniform on the boundary of the RVE. For a homogenized material the strain, strain relation in Voigt notation is given as $\overline{\sigma} = \overline{\mathbb{C}} : \overline{\varepsilon}$. We express the macroscopic strain as a Identity tensor such that:

$$
\overline{\varepsilon} = \begin{bmatrix} \varepsilon_{xx} & 0 & 0 \\ 0 & \varepsilon_{yy} & 0 \\ 0 & 0 & 2\varepsilon_{xy} \end{bmatrix} = \boldsymbol{I} \tag{C.1}
$$

This means that under such condition the macroscopic compliance matrix will be equal to average stress. Therefore,

$$
\overline{\mathbb{C}} = \begin{bmatrix} C_{xxxx} & & \\ & C_{yyyy} & \\ & & C_{xyxy} \end{bmatrix} = \begin{bmatrix} \overline{\sigma}_{xx} & & \\ & \overline{\sigma}_{yy} & \\ & & \overline{\sigma}_{xy} \end{bmatrix} \tag{C.2}
$$

To compute the values of $\overline{\sigma}_{xx}, \overline{\sigma}_{yy}$ and $\overline{\sigma}_{xy}$, we perform 3 FE simulations with 3 different boundary conditions. The microscale stresses for each boundary conditions are then average over the volume of the RVE as $\overline{\sigma}_{ij} = \int \sigma_{ij}(x) dV / \Omega_{\mathrm{RVE}}$. For the 2D case as considered in Section 4.2, we only need 3 boundary conditions: uniaxial strain along y-axis, uniaxial strain along x-axis and simple shear strain. We used FEniCSx for the Finite Element solution.

Data availability

The source code is freely available at: URL https://gitlab.ethz.ch/cmbm-public/papers-supp-info/2025/fft-methods-with-AD.

References

[1] H. Moulinec, P. Suquet, A numerical method for computing the overall response of nonlinear composites with complex microstructure, Comput. Methods Appl. Mech. Engrg. (ISSN: 0045-7825) 157 (1) (1998) 69–94, http://dx.doi.org/10.1016/S0045-7825(97)00218-1, URL https://www.sciencedirect.com/science/article/pii/S0045782597002181.

[2] J.C. Michel, H. Moulinec, P. Suquet, A computational scheme for linear and non-linear composites with arbitrary phase contrast, Internat. J. Numer. Methods Engrg. (ISSN: 1097-0207) 52 (1–2) (2001) 139–160, http://dx.doi.org/10.1002/nme.275, URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.275.

[3] Jaroslav Vondřejc, Jan Zeman, Ivo Marek, An FFT-based Galerkin method for homogenization of periodic media, Comput. Math. Appl. (ISSN: 0898-1221) 68 (3) (2014) 156–173, http://dx.doi.org/10.1016/j.camwa.2014.05.014, URL https://www.sciencedirect.com/science/article/pii/S0898122114002077.

[4] J. Zeman, T.W.J. de Geus, J. Vondřejc, R.H.J. Peerlings, M.G.D. Geers, A finite element perspective on nonlinear FFT-based micromechanical simulations, Internat. J. Numer. Methods Engrg. (ISSN: 1097-0207) 111 (10) (2017) 903–926, http://dx.doi.org/10.1002/nme.5481, URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.5481.

[5] Matti Schneider, A review of nonlinear FFT-based computational homogenization methods, Acta Mech. 232 (6) (2021) 2051–2100, (ISSN: 0001-5970, 1619-6937), http://dx.doi.org/10.1007/s00707-021-02962-1 URL https://link.springer.com/10.1007/s00707-021-02962-1.

[6] Martin Ladecký, Richard J. Leute, Ali Falsafi, Ivana Pultarová, Lars Pastewka, Till Junge, Jan Zeman, An optimal preconditioned FFT-accelerated finite element solver for homogenization, Appl. Math. Comput. (ISSN: 0096-3003) 446 (2023) 127835, http://dx.doi.org/10.1016/j.amc.2023.127835, URL https://www.sciencedirect.com/science/article/pii/S0096300323000048.

[7] T.W.J. de Geus, J. Vondřejc, J. Zeman, R.H.J. Peerlings, M.G.D. Geers, Finite strain FFT-based non-linear solvers made simple, Comput. Methods Appl. Mech. Engrg. (ISSN: 0045-7825) 318 (2017) 412–430, http://dx.doi.org/10.1016/j.cma.2016.12.032, URL https://www.sciencedirect.com/science/article/pii/S0045782516318709.

[8] Steffen Rothe, Stefan Hartmann, Automatic differentiation for stress and consistent tangent computation, Arch. Appl. Mech. (ISSN: 1432-0681) 85 (8) (2015) 1103–1125, http://dx.doi.org/10.1007/s00419-014-0939-6, URL https://doi.org/10.1007/s00419-014-0939-6.

[9] Matthias Rambausek, Felix Selim Göküzüm, Lu Trong Khiem Nguyen, Marc-André Keip, A two-scale FE-FFT approach to nonlinear magneto-elasticity, Internat. J. Numer. Methods Engrg. (ISSN: 1097-0207) 117 (11) (2019) 1117–1142, http://dx.doi.org/10.1002/nme.5993, URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.5993.

[10] Christian Gierden, Julian Kochmann, Johanna Waimann, Tobias Kinner-Becker, Jens Sölter, Bob Svendsen, Stefanie Reese, Efficient two-scale FE-FFT-based mechanical process simulation of elasto-viscoplastic polycrystals at finite strains, Comput. Methods Appl. Mech. Engrg. (ISSN: 0045-7825) 374 (2021) 113566, http://dx.doi.org/10.1016/j.cma.2020.113566, URL https://www.sciencedirect.com/science/article/pii/S0045782520307519.

[11] Julian Kochmann, Lisa Ehle, Stephan Wulfinghoff, Joachim Mayer, Bob Svendsen, Stefanie Reese, Efficient multiscale FE-FFT-based modeling and simulation of macroscopic deformation processes with non-linear heterogeneous microstructures, in: Jurica Sorić, Peter Wriggers, Olivier Allix (Eds.), Multiscale Modeling of Heterogeneous Structures, Springer International Publishing, Cham, ISBN: 978-3-319-65463-8, 2018, pp. 129–146, http://dx.doi.org/10.1007/978-3-319-65463-8_7, URL https://doi.org/10.1007/978-3-319-65463-8_7.

[12] Sebastian Felder, Julian Kochmann, Stephan Wulfinghoff, Stefanie Reese, Multiscale FE-FFT-based thermo-mechanically coupled modeling of viscoplastic polycrystalline materials.

[13] Thien Tran-Duc, J.E. Bunder, A.J. Roberts, Efficient computational homogenisation of 2D beams of heterogeneous elasticity using the patch scheme, Int. J. Solids Struct. (ISSN: 0020-7683) 292 (2024) 112719, http://dx.doi.org/10.1016/j.ijsolstr.2024.112719, URL https://www.sciencedirect.com/science/article/pii/S0020768324000763.

[14] Christian Gierden, Julian Kochmann, Johanna Waimann, Bob Svendsen, Stefanie Reese, A review of FE-FFT-based two-scale methods for computational modeling of microstructure evolution and macroscopic material behavior, Arch. Comput. Methods Eng. 29 (6) (2022) 4115–4135, (ISSN: 1134-3060, 1886-1784), http://dx.doi.org/10.1007/s11831-022-09735-6, URL https://link.springer.com/10.1007/s11831-022-09735-6.

[15] Charles C. Margossian, A review of automatic differentiation and its efficient implementation, WIREs Data Min. Knowl. Discov. (ISSN: 1942-4795) 9 (4) (2019) e1305, http://dx.doi.org/10.1002/widm.1305, URL https://onlinelibrary.wiley.com/doi/abs/10.1002/widm.1305.

[16] Andrea Vigliotti, Ferdinando Auricchio, Automatic differentiation for solid mechanics, Arch. Comput. Methods Eng. (ISSN: 1886-1784) 28 (3) (2021) 875–895, http://dx.doi.org/10.1007/s11831-019-09396-y, URL https://doi.org/10.1007/s11831-019-09396-y.

[17] Igor A. Baratta, Joseph P. Dean, Jørgen S. Dokken, Michal Habera, Jack S. Hale, Chris N. Richardson, Marie E. Rognes, Matthew W. Scroggs, Nathan Sime, Garth N. Wells, DOLFINx: the next generation FEniCS problem solving environment, 2023, preprint.

[18] Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, Geeta Chauhan, Anjali Chourdia, Will Constable, Alban Desmaison, Zachary DeVito, Elias Ellison, Will Feng, Jiong Gong, Michael Gschwind, Brian Hirsh, Sherlock Huang, Kshiteej Kalambarkar, Laurent Kirsch, Michael Lazos, Mario Lezcano, Yanbo Liang, Jason Liang, Yinghai Lu, C.K. Luk, Bert Maher, Yunjie Pan, Christian Puhrsch, Matthias Reso, Mark Saroufim, Marcos Yukio Siraichi, Helen Suk, Michael Suo, Phil Tillet, Eikan Wang, Xiaodong Wang, William Wen, Shunting Zhang, Xu Zhao, Keren Zhou, Richard Zou, Ajit Mathews, Gregory Chanan, Peng Wu, Soumith Chintala, PyTorch 2: Faster machine learning through dynamic Python bytecode transformation and graph compilation, 2024, 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, vol. 2 (ASPLOS '24) original-date: 2016-08-13T05:26:41Z, URL https://pytorch.org/assets/pytorch2-2.pdf.

[19] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, Qiao Zhang, JAX: composable transformations of Python+NumPy programs, 2018, URL http://github.com/google/jax.

[20] Samuel S. Schoenholz, Ekin D. Cubuk, JAX, M.D.: A Framework for Differentiable Physics, 2020, URL arXiv:1912.04232 [cond-mat, physics:physics, stat].

[21] Manuel Carrer, Henrique Musseli Cezar, Sigbjørn Løland Bore, Morten Ledum, Michele Cascella, Learning force field parameters from differentiable particle-field molecular dynamics, J. Chem. Inf. Model. (ISSN: 1549-9596) (2024) http://dx.doi.org/10.1021/acs.jcim.4c00564, URL https://doi.org/10.1021/acs.jcim.4c00564.

[22] Artur P. Toshev, Harish Ramachandran, Jonas A. Erbesdobler, Gianluca Galletti, Johannes Brandstetter, Nikolaus A. Adams, JAX-SPH: A Differentiable Smoothed Particle Hydrodynamics Framework, 2024, URL arXiv:2403.04750 [physics].

[23] Tianju Xue, Shuheng Liao, Zhengtao Gan, Chanwook Park, Xiaoyu Xie, Wing Kam Liu, Jian Cao, JAX-FEM: A differentiable GPU-accelerated 3D finite element solver for automatic inverse design and mechanistic data science, Comput. Phys. Comm. (ISSN: 0010-4655) 291 (2023) 108802, http://dx.doi.org/10.1016/j.cpc.2023.108802, URL https://www.sciencedirect.com/science/article/pii/S0010465523001479.

[24] Jeremy Bleyer, Numerical Tours of Computational Mechanics with FEniCSx, 2024, URL https://doi.org/10.5281/zenodo.10470942.

[25] Johannes Blühdorn, Nicolas R. Gauger, Matthias Kabel, AutoMat: automatic differentiation for generalized standard materials on GPUs, Comput. Mech. (ISSN: 1432-0924) 69 (2) (2022) 589–613, http://dx.doi.org/10.1007/s00466-021-02105-2, URL https://doi.org/10.1007/s00466-021-02105-2.

[26] Wolfram Research Inc. Mathematica, Version 14.1. URL https://www.wolfram.com/mathematica.

[27] Mohit Pundir, David S. Kammer, Ueli Angst, An FFT-based framework for predicting corrosion-driven damage in fractal porous media, J. Mech. Phys. Solids (ISSN: 0022-5096) 179 (2023) 105388, http://dx.doi.org/10.1016/j.jmps.2023.105388, URL https://linkinghub.elsevier.com/retrieve/pii/S0022509623001928.

[28] M.W. Hirsch, S. Smale, R.L. Devaney, Differential Equations, Dynamical Systems, and an Introduction to Chaos, third ed., Elsevier, ISBN: 978-0-12-382010-5, 2013, URL https://linkinghub.elsevier.com/retrieve/pii/C20090611600.

[29] Randall Balestriero, Richard Baraniuk, Fast Jacobian-Vector Product for Deep Networks, 2021, URL arXiv:2104.00219 [cs].

[30] John Douglas Eshelby, Rudolf Ernst Peierls, The determination of the elastic field of an ellipsoidal inclusion, and related problems, Proc. R. Soc. Lond. Ser. A Math. Phys. Sci. 241 (1226) (1997) 376–396, http://dx.doi.org/10.1098/rspa.1957.0133, URL https://royalsocietypublishing.org/doi/10.1098/rspa.1957.0133.

[31] Xavier Blanc, Claude Le Bris, Homogenization Theory for Multiscale Problems: An Introduction, in: MS & A, vol. 21, Springer Nature Switzerland, Cham, 2023, (ISBN: 978-3-031-21832-3, 978-3-031-21833-0), http://dx.doi.org/10.1007/978-3-031-21833-0, URL https://link.springer.com/10.1007/978-3-031-21833-0.

[32] M.G.D. Geers, V.G. Kouznetsova, W.A.M. Brekelmans, Multi-scale computational homogenization: Trends and challenges, J. Comput. Appl. Math. (ISSN: 0377-0427) 234 (7) (2010) 2175–2182, http://dx.doi.org/10.1016/j.cam.2009.08.077, URL https://www.sciencedirect.com/science/article/pii/S0377042709005536.

[33] Felix Meier, Cornelia Schwarz, Ewald Werner, Determination of the tangent stiffness tensor in materials modeling in case of large deformations by calculation of a directed strain perturbation, Comput. Methods Appl. Mech. Engrg. (ISSN: 0045-7825) 300 (2016) 628–642, http://dx.doi.org/10.1016/j.cma.2015.11.034, URL https://www.sciencedirect.com/science/article/pii/S0045782515003989.

[34] Felix Selim Göküzüm, Marc-André Keip, An algorithmically consistent macroscopic tangent operator for FFT-based computational homogenization, Internat. J. Numer. Methods Engrg. (ISSN: 1097-0207) 113 (4) (2018) 581–600, http://dx.doi.org/10.1002/nme.5627, URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.5627.

[35] Mohit Pundir, David S. Kammer, Supplementary Code - Simplifying FFT-based methods for mechanics with automatic differentiation, 2024, URL https://gitlab.ethz.ch/cmbm-public/papers-supp-info/2024/simplifying-fft-based-methods-for-mechanics-with-automatic-differentiation.

[36] J.C. Simo, T.J.R. Hughes, Computational Inelasticity, vo. 7, Springer-Verlag, New York, ISBN: 978-0-387-97520-7, 1998, http://dx.doi.org/10.1007/b98904, URL http://link.springer.com/10.1007/b98904.

[37] Ted Belytschko, W.K. Liu, B. Moran, K.I. Elkhodary, Nonlinear Finite Elements for Continua and Structures, second ed., Wiley, 2014.

[38] Madhu Balan P., Johnney Mertens A., M.V.A. Raju Bahubalendruni, Auxetic mechanical metamaterials and their futuristic developments: A state-of-art review, Mater. Today Commun. (ISSN: 2352-4928) 34 (2023) 105285, http://dx.doi.org/10.1016/j.mtcomm.2022.105285, URL https://www.sciencedirect.com/science/article/pii/S2352492822021262.

[39] Konstantinos Karapiperis, Dennis M. Kochmann, Prediction and control of fracture paths in disordered architected materials using graph neural networks, Commun. Eng. (ISSN: 2731-3395) 2 (1) (2023) 1–9, http://dx.doi.org/10.1038/s44172-023-00085-0, URL https://www.nature.com/articles/s44172-023-00085-0.

[40] Tommaso Magrini, Chelsea Fox, Adeline Wihardja, Athena Kolli, Chiara Daraio, Control of mechanical and fracture properties in two-phase materials reinforced by continuous, irregular networks, Adv. Mater. (ISSN: 1521-4095) 36 (6) (2024) 2305198, http://dx.doi.org/10.1002/adma.202305198, URL https://onlinelibrary.wiley.com/doi/abs/10.1002/adma.202305198.

[41] Angkur Jyoti Dipanka Shaikeea, Huachen Cui, Mark O'Masta, Xiaoyu Rayne Zheng, Vikram Sudhir Deshpande, The toughness of mechanical metamaterials, Nature Mater. (ISSN: 1476-4660) 21 (3) (2022) 297–304, http://dx.doi.org/10.1038/s41563-021-01182-1, URL https://www.nature.com/articles/s41563-021-01182-1.

[42] C. Miehe, A. Koch, Computational micro-to-macro transitions of discretized microstructures undergoing small strains, Arch. Appl. Mech. (ISSN: 1432-0681) 72 (4) (2002) 300–317, http://dx.doi.org/10.1007/s00419-002-0212-2, URL https://doi.org/10.1007/s00419-002-0212-2.

[43] J. Zeman, J. Vondřejc, J. Novák, I. Marek, Accelerating a FFT-based solver for numerical homogenization of periodic media by conjugate gradients, J. Comput. Phys. (ISSN: 0021-9991) 229 (21) (2010) 8065–8071, http://dx.doi.org/10.1016/j.jcp.2010.07.010, URL arXiv:1004.1122 [cond-mat, physics:physics].

[44] Martin P. Bendsøe, Ole Sigmund, Topology Optimization, Springer, Berlin, Heidelberg, 2004, (ISBN: 978-3-642-07698-5 978-3-662-05086-6), http://dx.doi.org/10.1007/978-3-662-05086-6, URL http://link.springer.com/10.1007/978-3-662-05086-6.

[45] Krister Svanberg, The method of moving asymptotes—a new method for structural optimization, Internat. J. Numer. Methods Engrg. (ISSN: 1097-0207) 24 (2) (1987) 359–373, http://dx.doi.org/10.1002/nme.1620240207, URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.1620240207.

[46] Aaditya Chandrasekhar, Saketh Sridhara, Krishnan Suresh, AuTO: A framework for automatic differentiation in topology optimization, Struct. Multidiscip. Optim. 64 (6) (2021) 4355–4365, (ISSN: 1615-147X, 1615-1488), http://dx.doi.org/10.1007/s00158-021-03025-8, URL arXiv:2104.01965 [cs, math].

[47] Liang Xia, Piotr Breitkopf, Design of materials using topology optimization and energy-based homogenization approach in Matlab, Struct. Multidiscip. Optim. (ISSN: 1615-1488) 52 (6) (2015) 1229–1241, http://dx.doi.org/10.1007/s00158-015-1294-0, URL https://doi.org/10.1007/s00158-015-1294-0.

[48] Erik Andreassen, Anders Clausen, Mattias Schevenels, Boyan S. Lazarov, Ole Sigmund, Efficient topology optimization in MATLAB using 88 lines of code, Struct. Multidiscip. Optim. (ISSN: 1615-1488) 43 (1) (2011) 1–16, http://dx.doi.org/10.1007/s00158-010-0594-7, URL https://doi.org/10.1007/s00158-010-0594-7.

[49] S. Amstutz, S.M. Giusti, A.A. Novotny, E.A. de Souza Neto, Topological derivative for multi-scale linear elasticity models applied to the synthesis of microstructures, Internat. J. Numer. Methods Engrg. (ISSN: 1097-0207) 84 (6) (2010) 733–756, http://dx.doi.org/10.1002/nme.2922, URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.2922.

[50] Zeyao Chen, Baisheng Wu, Yi Min Xie, Xian Wu, Shiwei Zhou, FFT-based inverse homogenization for cellular material design, Int. J. Mech. Sci. (ISSN: 0020-7403) 231 (2022) 107572, http://dx.doi.org/10.1016/j.ijmecsci.2022.107572, URL https://www.sciencedirect.com/science/article/pii/S0020740322004635.

[51] Q. Gu, J.P. Conte, Z. Yang, A. Elgamal, Consistent tangent moduli for multi-yield-surface J2 plasticity model, Comput. Mech. 48 (1) (2011) 97–120.

[52] S. Lucarini, F.P.E. Dunne, E. Martínez-Pañeda, An FFT-based crystal plasticity phase-field model for micromechanical fatigue cracking based on the stored energy density, Int. J. Fatigue (ISSN: 0142-1123) 172 (2023) 107670, http://dx.doi.org/10.1016/j.ijfatigue.2023.107670, URL https://www.sciencedirect.com/science/article/pii/S0142112323001718.

[53] T. Takaki, A. Yamanaka, Y. Higa, Y. Tomita, Phase-field model during static recrystallization based on crystal-plasticity theory, J. Comput.-Aided Mater. Des. (ISSN: 1573-4900) 14 (1) (2007) 75–84, http://dx.doi.org/10.1007/s10820-007-9083-8, URL https://doi.org/10.1007/s10820-007-9083-8.

[54] L. Sharma, R.H.J. Peerlings, P. Shanthraj, F. Roters, M.G.D. Geers, An FFT-based spectral solver for interface decohesion modelling using a gradient damage approach, Comput. Mech. 65 (4) (2020) 925–939, (ISSN: 0178-7675 1432-0924), http://dx.doi.org/10.1007/s00466-019-01801-4, URL http://link.springer.com/10.1007/s00466-019-01801-4.

[55] Yang Chen, Dmytro Vasiukov, Lionel Gélébart, Chung Hae Park, A FFT solver for variational phase-field modeling of brittle fracture, Comput. Methods Appl. Mech. Engrg. (ISSN: 0045-7825) 349 (2019) 167–190, http://dx.doi.org/10.1016/j.cma.2019.02.017, URL https://www.sciencedirect.com/science/article/pii/S004578251930088X.

[56] Indre Jödicke, Richard J. Leute, Till Junge, Lars Pastewka, Efficient topology optimization using compatibility projection in micromechanical homogenization, 2022, URL arXiv:2107.04123 [cs].

[57] Andreas Griewank, David Juedes, Jean Utke, Algorithm 755: ADOL-C: a package for the automatic differentiation of algorithms written in C/C++, ACM Trans. Math. Software (ISSN: 0098-3500) 22 (2) (1996) 131–167, http://dx.doi.org/10.1145/229473.229474, URL https://dl.acm.org/doi/10.1145/229473.229474.

[58] autodiff/autodiff, 2024, URL https://github.com/autodiff/autodiff. original-date: 2018-07-19T07:28:38Z.

[59] Eric Phipps, Roger Pawlowski, Christian Trott, Automatic differentiation of C++ codes on emerging manycore architectures with sacado, ACM Trans. Math. Software 48 (4) (2022).

[60] William Moses, Valentin Churavy, Instead of rewriting foreign code for machine learning, automatically synthesize fast gradients, in: H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, H. Lin (Eds.), Advances in Neural Information Processing Systems, vol. 33, Curran Associates, Inc., 2020, pp. 12472–12485, URL https://proceedings.neurips.cc/paper/2020/file/9332c513ef44b682e9347822c2e457ac-Paper.pdf.

[61] William S. Moses, Sri Hari Krishna Narayanan, Ludger Paehler, Valentin Churavy, Michel Schanen, Jan Hückelheim, Johannes Doerfert, Paul Hovland, Scalable Automatic Differentiation of Multiple Parallel Paradigms through Compiler Augmentation, in: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC '22, IEEE Press, Dallas, Texas, 2022, (ISBN: 9784665454445).

[62] Sanath Keshav, Felix Fritzen, Matthias Kabel, FFT-based Homogenization at Finite Strains using Composite Boxels (ComBo), 2022.

[63] S. Lucarini, J. Segurado, DBFFT: A displacement based FFT approach for non-linear homogenization of the mechanical behavior, Internat. J. Engrg. Sci. (ISSN: 0020-7225) 144 (2019) 103131, http://dx.doi.org/10.1016/j.ijengsci.2019.103131, URL https://www.sciencedirect.com/science/article/pii/S0020722519304446.

[64] Mikhail Kuts, James Walker, Pania Newell, Computational homogenization of linear elastic properties in porous non-woven fibrous materials, Mech. Mater. (ISSN: 0167-6636) 189 (2024) 104868, http://dx.doi.org/10.1016/j.mechmat.2023.104868, URL https://www.sciencedirect.com/science/article/pii/S0167663623003149.