CSMA 2026
17ème Colloque National en Calcul des Structures
18-22 Mai 2026, Presqu'île de Giens (Var)

DOLFINx-External-Operators: integrating externally defined constitutive models into FEniCSx

A. Latyshev[1,2], J.S. Dokken[3], J. Bleyer[4], J. S. Hale[1], C. Maurini[2]

[1] *Department of Engineering, Faculty of Science, Technology and Medicine, Université du Luxembourg, Luxembourg. andrey.latyshev@uni.lu, jack.hale@uni.lu*
[2] *Institut Jean Le Rond d'Alembert, Sorbonne Université, UMR CNRS 7190, France. corrado.maurini@sorbonne-universite.fr*
[3] *Department of Numerical Analysis and Scientific Computing, Simula Research Laboratory, Norway. dokken@simula.no*
[4] *Laboratoire Navier, École des Ponts ParisTech, Université Gustave Eiffel, UMR CNRS 8205, France. jeremy.bleyer@enpc.fr*

Résumé — Many solid mechanics problems involve constitutive models that are difficult to express in variational form. We introduce a concise methodology and open-source software framework that couples variational forms with externally defined operators written in general-purpose programming languages, enabling seamless integration of complex constitutive models into FEniCSx. The approach is demonstrated on a Mohr–Coulomb elastoplastic model with apex smoothing implemented in JAX, where algorithmic automatic differentiation provides the derivatives required for the solution process.
Mots clés — external operators, constitutive models, automated finite element solvers, algorithmic automatic differentiation, JAX, Numba, FEniCSx.

1 Background

A number of methods have been proposed to incorporate general constitutive models into automated finite element solvers such as DOLFINx [4]. One method is to write programmatic interfaces from the finite element solver to frameworks specifically designed for implementing non-standard constitutive models such as MFront [8] and ZMAT [1, 12]. Although this approach does not require any changes to the existing functionality of Unified Form Language (UFL) [3], it requires significant effort to develop interfaces between the finite element solver and the interface provided by each constitutive modelling package. Furthermore, programmatic approaches often feel *ad hoc* as they do not integrate with the abstractions and automatic differentiation tools provided by UFL to write the variational part of the problem.

A more recent approach, and one that forms the basis for this work, is the introduction of the *external operator* extension to UFL [5]. The external operator is a symbolic UFL object that represents a general mapping between finite element quantities in a form; for a formal definition, see [5, Definition 1]. The action of the operator itself can then be concretely defined externally, using 'any' programming language. In essence, the external operator concept creates a bridge between the language of forms and the broader possibilities provided by general programming languages. Another key feature of external operators is that they can be symbolically differentiated by UFL, producing new symbolic external operators representing the action of the derivative of the original operator - this opens up numerous possibilities for using programming languages that support *algorithmic automatic differentiation* (AD) to automatically create the necessary derivatives.

2 Contributions

During the software session, we will demonstrate two main contributions. Firstly, we describe a methodology and software framework that extends the open source library DOLFINx to support the recently introduced symbolic external operator in UFL [5]. In turn, this provides DOLFINx/FEniCSx users with an interface that allows the definition of general constitutive models via a wide variety of programming languages.

Building on this software framework, our second main contribution is to explore the use of program-

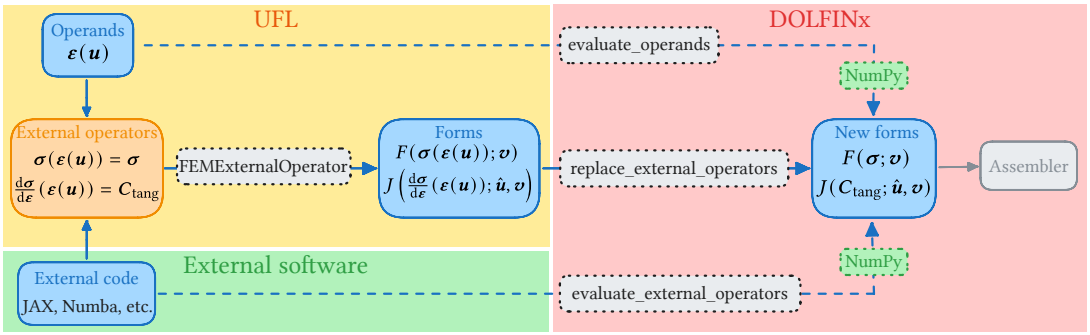

FIGURE 1 – The diagram summarizes the workflow of implementing a solid mechanics constitutive model within the framework. The stress $\boldsymbol{\sigma}(\boldsymbol{\varepsilon}(\boldsymbol{u}))$ and its derivative the tangent stiffness $\boldsymbol{C}_{\text{tang}}(\boldsymbol{\varepsilon}(\boldsymbol{u}))$ are wrapped with the `FEMExternalOperator` objects, which depend on the UFL-expression of the strain field $\boldsymbol{\varepsilon}(\boldsymbol{u})$ and some external code. Once the forms containing `FEMExternalOperator` objects are defined, they can be replaced with their representatives from the DOLFINx environment, the `Function` objects $\boldsymbol{\sigma}$ and $\boldsymbol{C}_{\text{tang}}$. The values of the UFL-expression of $\boldsymbol{\varepsilon}(\boldsymbol{u})$ are computed at Gauss points via the `evaluate_operands` function using FFCx generated code and stored in an `ndarray`. The values of the external operators and their derivatives are then evaluated at the Gauss points via the `evaluate_external_operators` function and stored in `ndarray`. In the final step, the standard DOLFINx assembler is used to assemble the forms with the `Function` objects containing $\boldsymbol{\sigma}$ and $\boldsymbol{C}_{\text{tang}}$.

ming languages with algorithmic automatic differentiation capabilities for expressing constitutive models. To demonstrate this we implement a Mohr-Coulomb elastoplastic model with apex smoothing [2] that has previously been implemented using MFront [8]. By leveraging JAX [7], a Python library for high-performance array computations with composable transformations for automatic differentiation, we completely avoid both the manual expression and implementation of the necessary derivatives. The resulting implementation of the complete Mohr-Coulomb finite element solver in DOLFINx is remarkably compact and its correctness is verified against an existing solution from the literature.

The software framework is openly available [9] and includes further fully documented examples. Fig. 1 schematically visualizes the process of the main steps of the workflow. Fig. 2 provides a "minimal" code example of the framework application to a solid mechanics problem.

3 Results

We implement the non-associative plasticity model of Mohr-Coulomb with apex-smoothing and solve a soil slope stability problem. We use the JAX package to define constitutive relations including the differentiation of certain terms. This example demonstrates how AD techniques may be used to define constitutive models that require differentiation of expressions without significant differentiation by hand.

3.1 Problem formulation

We solve a soil slope stability problem of a domain Ω represented by a rectangle $[0,L] \times [0,H]$ under plane strain assumptions loaded by an external force on a part of its boundary $\partial\Omega_N$ with outward facing normal $\boldsymbol{n}$, and V be a space of admissible displacements. The equilibrium state of the solid body is described by the following variational problem : find the displacement field $\boldsymbol{u} \in V$ such that the following weak residual equation is satisfied

$$F(\boldsymbol{u}; \boldsymbol{v}) = \int_{\Omega} \boldsymbol{\sigma}(\boldsymbol{\varepsilon}(\boldsymbol{u})) \cdot \boldsymbol{\varepsilon}(\boldsymbol{v}) \, \mathrm{d}\boldsymbol{x} - F_{\text{ext}}(\boldsymbol{v}) = 0, \quad \forall \boldsymbol{v} \in V, \tag{1}$$

where $\varepsilon(\boldsymbol{u}) = \frac{1}{2}(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T)$ is the small strain tensor of the displacement field $\boldsymbol{u}$ and $\boldsymbol{\sigma}$ the stress in a Cartesian coordinate system $\boldsymbol{x} = (x, y, z)$. For this problem the homogeneous Dirichlet boundary conditions for the displacement field $\boldsymbol{u} = \boldsymbol{0}$ on the right side $x = L$ and the bottom $y = 0$. The loading

```python
def sigma_external(
    derivatives: Tuple[int, ...]
) -> Callable[[np.ndarray], np.ndarray]:
    if derivatives == (0,):
        return sigma_impl # user-defined function (external code is inside)
    elif derivatives == (1,):
        return C_tang_impl # user-defined function (external code is inside)
    else:
        raise NotImplementedError

# Define the output function space of the external operator
S = fem.functionspace(mesh, quadrature_element)

sigma = FEMExternalOperator(
    epsilon(u), # operand: UFL Expression of strains
    function_space=S, # Output function space
    external_function=sigma_external # Python callable
)

# Define the form `F` and its Jacobian `J`
F = ufl.inner(sigma, epsilon(v)) * dx - F_ext(v)
# UFL's SD creates a new `FEMExternalOperator` wrapping the
# derivative of `sigma` aka `C_tang`
J = ufl.derivative(F, u, u_hat)
J_expanded = ufl.algorithms.expand_derivatives(J)

# Create new forms with `FEMExternalOperator` replaced with `Function` objects
# appropriately sized to hold the result of evaluating the external operator
F_replaced, F_external_operators = replace_external_operators(F)
J_replaced, J_external_operators = replace_external_operators(J_expanded)

# Define final forms to assemble
F_form = fem.form(F_replaced)
J_form = fem.form(J_replaced)

# Loop implementing iterative solution algorithm.
# e.g. Newton method, fixed-point iteration etc.
for _ in range(0, 10):
    # Evaluate values of `epsilon(u)`
    evaluated_operands = evaluate_operands(F_external_operators)
    # Evaluate and update values of `sigma` via 'sigma_impl'
    evaluate_external_operators(J_external_operators, evaluated_operands)
    # Evaluate and update values of `C_tang` via 'C_tang_impl'
    evaluate_external_operators(F_external_operators, evaluated_operands)

    # Assemble Jacobian into matrix
    A_matrix = fem.assemble_matrix(J_form)
    # Assemble residual into vector
    b_vector = fem.assemble_vector(F_form)
```

FIGURE 2 – Minimal and abbreviated code example of the framework applied to a non-specific solid mechanics problem. It shows how the main features of the framework (class `FEMExternalOperator` and functions `replace_external_operators`, `evaluate_operands`, `evaluate_external_operators`) are used to define the problem. Note how external operator allows for the concise and unified expression of models involving variational and non-variational terms in UFL.

consists of a gravitational body force $\boldsymbol{q} = [0, -\gamma]^T$ with γ being the soil self-weight

$$F_{\text{ext}}(\boldsymbol{v}) = \int_{\Omega} \boldsymbol{q} \cdot \boldsymbol{v} \, \mathrm{d}\boldsymbol{x}. \tag{2}$$

We progressively increase the soil self-weight γ until a plateau on the loading-displacement curve is reached.

The constitutive model of the soil is described by a non-associative plasticity law without hardening that is defined by the Mohr-Coulomb yield surface f and the plastic potential g. Both quantities may be expressed through the following function h

$$h(\boldsymbol{\sigma}, \alpha) = \frac{I_1(\boldsymbol{\sigma})}{3} \sin\alpha + \sqrt{J_2(\boldsymbol{\sigma}) K^2(\theta, \alpha) + a^2(\alpha) \sin^2\alpha} - c\cos\alpha, \tag{3}$$

$$f(\boldsymbol{\sigma}) = h(\boldsymbol{\sigma}, \phi), \tag{4}$$

$$g(\boldsymbol{\sigma}) = h(\boldsymbol{\sigma}, \psi), \tag{5}$$

where c is a cohesion, ϕ, ψ and θ are friction, dilatancy and Lode angles respectively, $I_1(\boldsymbol{\sigma}) = \mathrm{tr}\boldsymbol{\sigma}$ is the first invariant of the stress field and $J_2(\boldsymbol{\sigma}) = \frac{1}{2}\boldsymbol{s} \cdot \boldsymbol{s}$ is the second invariant of the deviatoric part of the stress.

Partial code for implementing this model using JAX is shown in fig. 3. We perform two verification tests. The first checks the correct tracing of the yield surface. The second checks that our solution of the slope stability problem defined by matches an existing result found in the literature.

Yield surface tracing In this part, we verify that the Mohr-Coulomb model is implemented correctly by visually tracing its yield surface. We generate several stress paths and check whether they remain within the yield surface after passing through the `return_mapping` function, which depends on the derivatives $\frac{\mathrm{d}g(\boldsymbol{\sigma})}{\mathrm{d}\boldsymbol{\sigma}}$ and $\frac{\mathrm{d}\boldsymbol{r}(\boldsymbol{y})}{\mathrm{d}\boldsymbol{y}}$. The yield surface with the stress paths projected onto the deviatoric plane is shown in fig. 4, where we observe that the yield surface of Mohr-Coulomb with apex smoothing is reached along different stress paths (colored lines). Moreover, the obtained yield surface lies along the standard Mohr-Coulomb one without smoothing (black contour line). These results justify the correct implementation of the plasticity model and the derivatives `dgdsigma` and `drdx` obtained by JAX's AD.

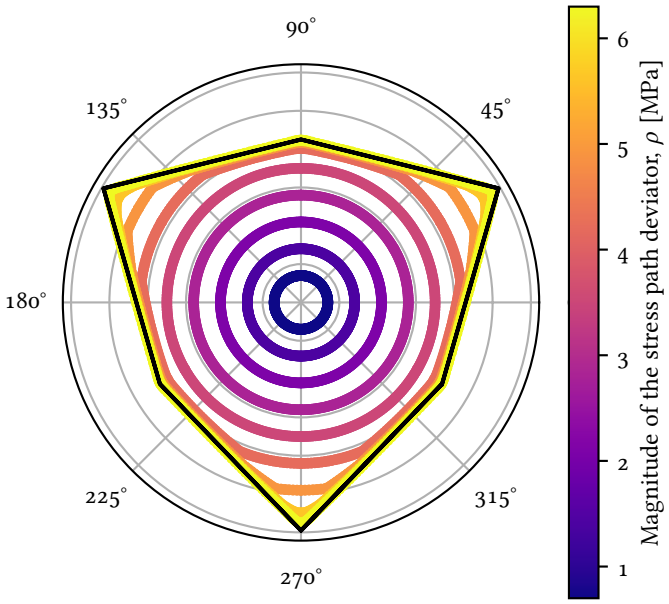

FIGURE 4 – Tracing of the Mohr-Coulomb with apex smoothing yield surface. It is obtained by passing several stress paths projected onto the deviatoric plane (ρ, θ), where $\rho = \sqrt{2J_2(\boldsymbol{\sigma})}$ and θ is the Lode angle. Each colour represents one loading step along the stress paths. The circles are associated with the loading during the elastic phase. Once the loading reaches the elastic limit, the circles start outlining the yield surface, which in the limit lay along the standard Mohr-Coulomb one without smoothing (black contour).

```python
def r(y_local, deps_local, sigma_n_local):
    sigma_local = y_local[:stress_dim]
    dlambda_local = y_local[-1]

    # The definitions r_g and r_f are not shown
    res_g = r_g(sigma_local, dlambda_local, deps_local, sigma_n_local)
    res_f = r_f(sigma_local, dlambda_local, deps_local, sigma_n_local)

    res = jnp.c_["0,1,-1", res_g, res_f]  # concatenates an array and a scalar
    return res

drdy = jax.jacfwd(r)

def return_mapping(deps_local: np.ndarray, sigma_n_local: np.ndarray):
    niter = 0

    dlambda = ZERO_SCALAR
    sigma_local = sigma_n_local
    y_local = jnp.concatenate([sigma_local, dlambda])

    res = r(y_local, deps_local, sigma_n_local)
    norm_res0 = jnp.linalg.norm(res)

    def cond_fun(state):
        norm_res, niter, _ = state
        return jnp.logical_and(norm_res / norm_res0 > tol, niter < Nitermax)

    def body_fun(state):
        norm_res, niter, history = state

        y_local, deps_local, sigma_n_local, res = history

        j = drdy(y_local, deps_local, sigma_n_local)
        j_inv_vp = jnp.linalg.solve(j, -res)
        y_local = y_local + j_inv_vp

        res = r(y_local, deps_local, sigma_n_local)
        norm_res = jnp.linalg.norm(res)
        history = y_local, deps_local, sigma_n_local, res

        niter += 1

        return (norm_res, niter, history)

    history = (y_local, deps_local, sigma_n_local, res)

    norm_res, niter_total, y_local = jax.lax.while_loop(cond_fun, body_fun,
    (norm_res0, niter, history))

    sigma_local = y_local[0][:stress_dim]
    dlambda = y_local[0][-1]
    sigma_elas_local = C_elas @ deps_local
    yielding = f(sigma_n_local + sigma_elas_local)

    return sigma_local, (sigma_local, niter_total, yielding, norm_res, dlambda)

dsigma_ddeps = jax.jacfwd(return_mapping, has_aux=True)
```

FIGURE 3 – The implementation of the return-mapping procedure of the Mohr-Coulomb plasticity with apex smoothing via JAX package. The function `return_mapping` receives NumPy arrays of values of the strains $\boldsymbol{\varepsilon}(\Delta \boldsymbol{u})$ and the stresses $\boldsymbol{\sigma}^n$ evaluated at a given Gauss point. The return-mapping procedure is implemented via the Newton method wrapped by the JAX's `while_loop`. It returns a tuple of the new stress field values at a given Gauss point and another tuple containing the same values of the stress field plus some auxiliary data related to the Newton method. Then we apply the JAX's AD `jacfwd` to compute the derivative of the function `return_mapping` with respect to its first input. This results in the creation of a new function `dsigma_ddeps` that returns both the stress field and its derivative at a given Gauss point.

Solution of slope stability problem We now demonstrate that our numerical solution of the slope stability problem matches the results found in the literature. We progressively increase the second component of the gravitational body force $\boldsymbol{q} = [0, -\gamma]^T$ from eq. 2, the soil self-weight γ, up to the critical value $\gamma_{\text{lim}}^{\text{num}}$, when the perfect plasticity plateau is reached on the loading-displacement curve at the top left corner $(0, H)$. Then we compare $\gamma_{\text{lim}}^{\text{num}}$ with analytical γ_{lim} found through the formula of the slope stability factor l_{lim}

$$l_{\text{lim}} = \gamma_{\text{lim}} H / c. \tag{6}$$

The orange loading-displacement curve shown in fig. 5 confirms that the numerically estimated yield strength limit reached for $\gamma_{\text{lim}}^{\text{num}}$ is close to γ_{lim}. Additionally, we demonstrate the magnitude of the displacement field at the last loading step in fig. 6, where the slip of the rectangular slope can be observed on the left side $x = 0$.

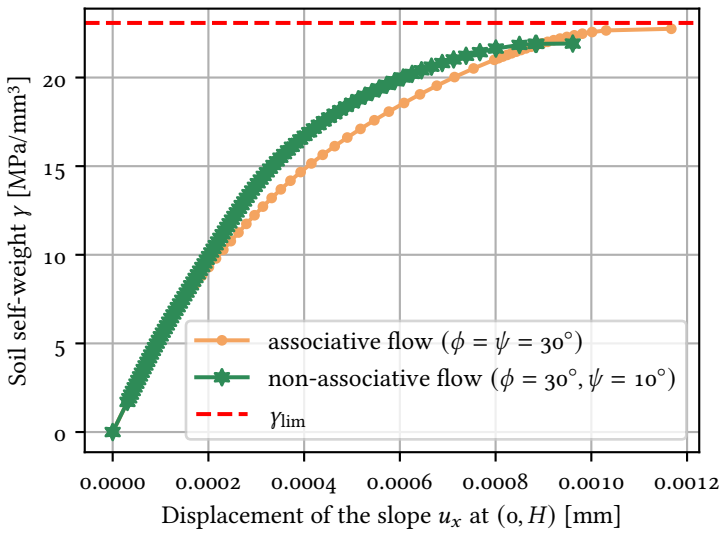

FIGURE 5 – Displacement of the slope $u_x(0, H)$ with respect to the soil self-weight γ in the Mohr-Coulomb model with apex smoothing for the associative plastic flow ($\phi = \psi = 30°$) and the non-associative one ($\phi = 30°, \psi = 10°$). Reaching the yield strength limit $\gamma_{\text{lim}}^{\text{num}}$ (the plateau) is associated with losing the stability by the slope. The γ_{lim} for the associative flow is obtained via an analytical solution using the standard Mohr-Coulomb model without smoothing [6, p. 368].

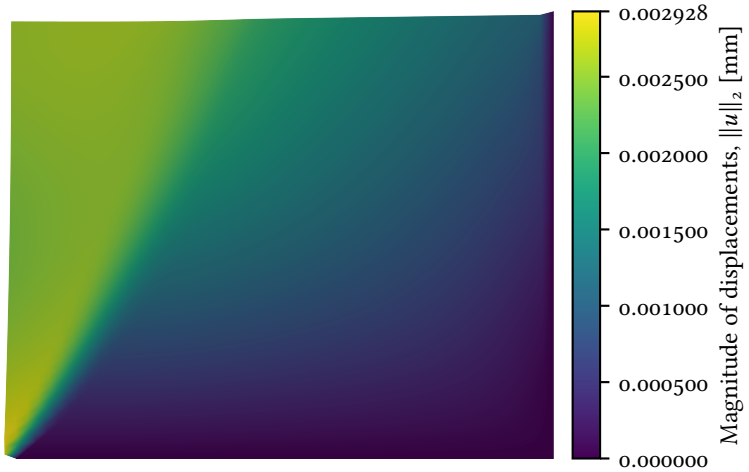

FIGURE 6 – Slip of the slope for the Mohr-Coulomb problem. The domain is warped by the displacement field (magnified). The magnitude of the displacement field is shown by the colour and reaches its maximum at the bottom left corner.

4 Summary

We described a methodology to define a range of constitutive models in FEniCSx / DOLFINx and a supporting software framework. Our framework provides the user with an interface that wraps a constitutive model via an external operator [5]. Data is passed between the external operator and DOLFINx using standard array-like data structures, allowing a wide range of different programming languages and environments to be used. Together, these developments enable the definition of a general class of constitutive models in FEniCSx.

Funding

This research was funded in whole, or in part, by the Luxembourg National Research Fund (FNR), grant reference PRIDE/21/16747448/MATHCODA. For the purpose of open access, and in fulfilment of the obligations arising from the grant agreement, the author has applied a Creative Commons Attribution 4.0 International (CC BY 4.0) license to any Author Accepted Manuscript version arising from this submission.

Acknowledgements

Text and figures in this article were adapted from our published journal article [11] under the CC BY 4.0 license. We also note that an earlier version of this work on implementing external operators in DOLFINx was published in the 2024 proceedings [10].

Author's contributions

AL : Conceptualisation, Formal analysis, Investigation, Methodology, Software, Validation, Visualisation, Writing - original draft, Writing - review and editing. JSD : Software, since publication of [11], Writing - review and editing. JB : Conceptualisation, Methodology, Supervision (Masters thesis of AL), Validation, Writing - review and editing. CM : Conceptualisation, Supervision (Masters and PhD thesis of AL), Project administration, Writing - review and editing. JSH : Conceptualisation, Formal analysis, Funding acquisition, Methodology, Project administration, Software, Supervision (PhD thesis of AL), Writing - review and editing.

Références

[1] Mohamed ABATOUR et al. "A generic formulation of anisotropic thermo-elastoviscoplasticity at finite deformations for finite element codes". en. In : *Computational Mechanics* (oct. 2024). ISSN : 1432-0924. DOI : 10.1007/s00466-024-02543-8. (Visité le 17/01/2025).

[2] A.J. ABBO et S.W. SLOAN. "A Smooth Hyperbolic Approximation to the Mohr-Coulomb Yield Criterion". In : *Computers & Structures* 54.3 (fév. 1995), p. 427-441. ISSN : 00457949. DOI : 10.1016/0045-7949(94)00339-5.

[3] Martin S. ALNÆS et al. "Unified Form Language : A Domain-Specific Language for Weak Formulations of Partial Differential Equations". In : *ACM Transactions on Mathematical Software* 40.2 (5 mars 2014), 9 :1-9 :37. ISSN : 0098-3500. DOI : 10.1145/2566630.

[4] Igor A. BARATTA et al. *DOLFINx : The next generation FEniCS problem solving environment*. Déc. 2025. DOI : 10.5281/zenodo.10447665.

[5] Nacime BOUZIANI et David A. HAM. "Escaping the Abstraction : A Foreign Function Interface for the Unified Form Language [UFL]". In : First Workshop on Differentiable Programming (NeurIPS 2021) (13 déc. 2021). 2021. DOI : 10.48550/arXiv.2111.00945.

[6] W. F. CHEN et X. L. LIU. *Limit Analysis in Soil Mechanics*. T. 52. Developments in Geotechnical Engineering. Elsevier Science, 1990. 477 p. ISBN : 0-444-43042-3.

[7] Roy FROSTIG, Matthew James JOHNSON et Chris LEARY. “Compiling Machine Learning Programs via High-Level Tracing”. In : *Systems for Machine Learning*. SysML Conference 2018 (Stanford, CA, United States, 31 mars-2 avr. 2019). 2018. URL : https://mlsys.org/Conferences/doc/2018/146.pdf.

[8] Thomas HELFER et al. “Introducing the Open-Source Mfront Code Generator : Application to Mechanical Behaviours and Material Knowledge Management within the PLEIADES Fuel Element Modelling Platform”. In : *Computers & Mathematics with Applications* 70.5 (sept. 2015), p. 994-1023. ISSN : 08981221. DOI : 10.1016/j.camwa.2015.06.027.

[9] Andrey LATYSHEV et Jack S. HALE. *dolfinx-external-operator*. Oct. 2024. DOI : 10.5281/zenodo.10907417.

[10] Andrey LATYSHEV et al. “A Framework for Expressing General Constitutive Models in FEniCSx”. In : 16ème Colloque National En Calcul de Structures (13-17 mai 2024). Giens, France : CNRS, CSMA, ENS Paris-Saclay, CentraleSupélec, mai 2024. URL : https://hal.science/hal-04610881.

[11] Andrey LATYSHEV et al. “Expressing general constitutive models in FEniCSx using external operators and algorithmic automatic differentiation”. In : *Journal of Theoretical, Computational and Applied Mechanics* (oct. 2025), p. 14449. ISSN : 2726-6141. DOI : 10.46298/jtcam.14449.

[12] Z-SET SOFTWARE. *ZMAT*. 2023. URL : http://zset-software.com/products/z-mat/.