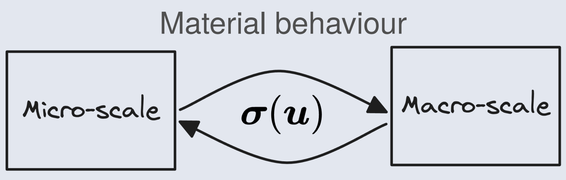
EXPRESSING GENERAL CONSTITUTIVE MODELS IN FENICSX USING EXTERNAL OPERATORS AND AUTOMATIC DIFFERENTIATION

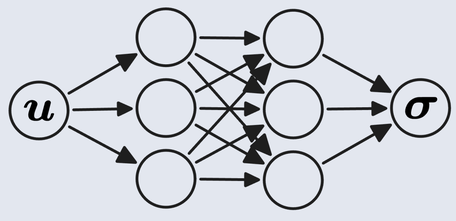
Andrey Latyshev[1,2] **Jérémy Bleyer**[3] **Jack S. Hale**[1] **Corrado Maurini**[2]

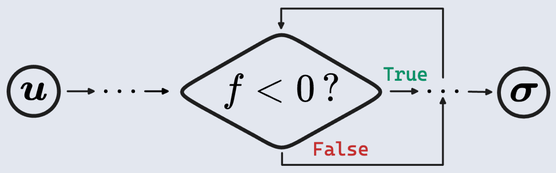
[1]Université du Luxembourg, Luxembourg, [2]Sorbonne Université, France, [3]Laboratoire Navier, École des Ponts, Université Gustave Eiffel, CNRS, France

CONSTITUTIVE MODELS OF SOLID MECHANICS DEFINE COMPLEX MATERIAL BEHAVIOUR AND CAN BE EXPRESSED IN DIFFERENT WAYS



$$\sigma(u) = J^{-1}(2C_1(F - F^{-T}) + 2D_1(J-1)JF^{-T})F^T$$





LIMITATIONS OF UNIFIED FORM LANGUAGE (UFL)

The weak form of an equilibrium problem of solid mechanics

$$F(u; v) = \int_\Omega \sigma(u) \cdot \varepsilon(v) \mathrm{d}x \quad u, v \in V,$$

can be straightforwardly expressed in *Unified Form Language* (UFL) in FEniCSx:

```
F = ufl.inner(sigma(u), epsilon(v)) * ufl.dx
```

Can we discretize and assembly the form F via DOLFINx, if the constitutive model

- $\sigma(u) =$ analytical formula?
- $\sigma(u) =$ numerical algorithm?
- $\sigma(u) =$ experimental data?
- $\sigma(u) =$ neural network?

EXTERNAL OPERATOR

An **external operator** (Bouziani and Ham, 2021) $N(\cdot)$ is a *symbolic* UFL object that is

- defined by a *computer program*, not UFL expressions.
- *differentiable* and it's derivative $\partial N(\cdot)$ is a new **external operator**.

HOW TO USE EXTERNAL OPERATORS IN FENICSX

The stress-displacement relation $\sigma(u) := \sigma(\varepsilon(u))$ is treated as an **external operator**

```
sigma = FEMExternalOperator(
    epsilon(u), # operand
    function_space=S, # quadrature space
    external_function=sigma_external) # Python function
```

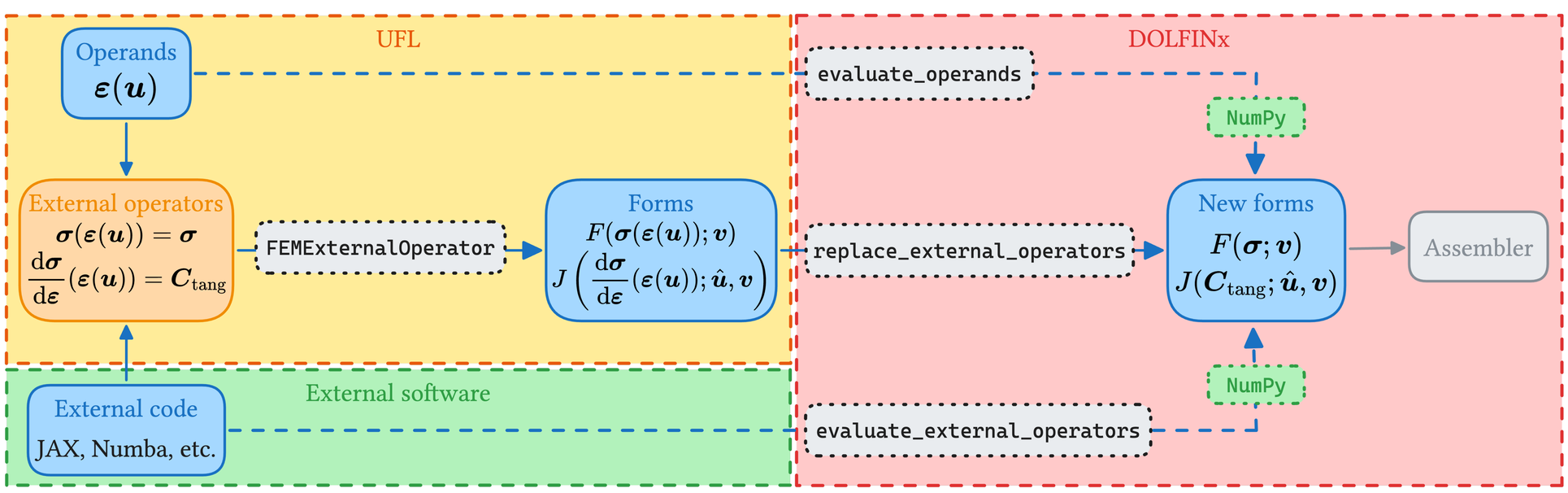
Now we can define the form F and then discretize and assemble it in DOLFINx:

```
F = ufl.inner(sigma, epsilon(v)) * dx.
```

DEFINING EXTERNAL OPERATORS BEHAVIOUR

The function `sigma_external` contains the definitions of the external operator and its derivative, which are simple Python functions:

```
def sigma_external(derivatives):
    if derivatives == (0,):
        return sigma_impl # Python function
    if derivatives == (1,):
        return C_tang_impl # Python function
```



SOLVING COMPLEX EXAMPLE

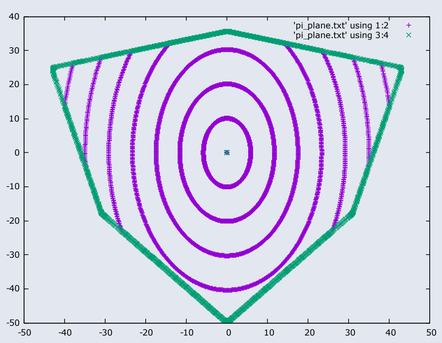
Mohr-Coulomb plasticity with apex-smoothing and non-associated flow rule

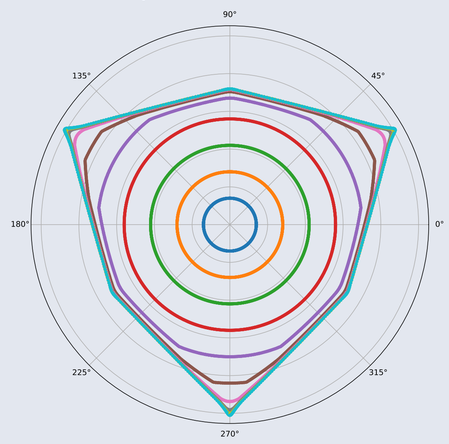
Original solution (Helfer et al., 2024)

- Manual differentiation (days)
- Hundreds of lines of code
- C++ API
- Compilable code

External operators + JAX

- Automatic differentiation (AD) (hours)
- Couple lines of code
- Python API
- JIT-compilable code





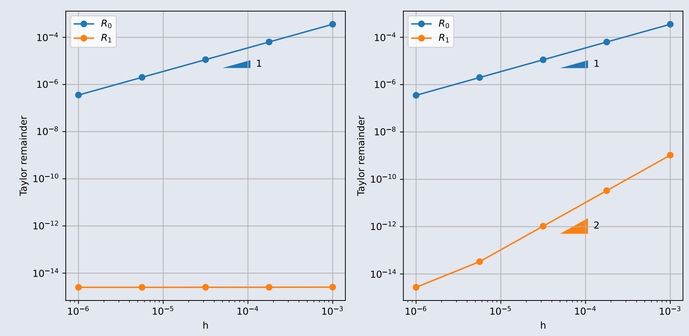
Yield surface tracing.

VERIFICATION VIA TAYLOR TEST

Taylor remainders convergence rate for Mohr-Coulomb plasticity:

$$R_0 = |F(u + h\,\delta u; v) - F(u; v)| \longrightarrow 0 \text{ at } O(h),$$

$$R_1 = |F(u + h\,\delta u; v) - F(u; v) - J(u; h\delta u, v)| \longrightarrow 0 \text{ at } O(h^2).$$



On the left: elastic phase. On the right: plastic phase.

HOW TO INSTALL AND USE

We implemented the framework `dolfinx-external-operator`

- It allows to express general constitutive models in DOLFINx via **any** external software.
- In particular, it enables the AD in DOLFINx via JAX library.

How to install + tutorials:
a-latyshev.github.io/dolfinx-external-operator/

Andrey Latyshev : andrey.latyshev@uni.lu

This research was funded in whole, or in part, by the Luxembourg National Research Fund (FNR), grant reference PRIDE/21/16747448/MATHCODA.

REFERENCES

Bouziani, Nacime and David A. Ham (2021). *Escaping the abstraction: a foreign function interface for the Unified Form Language [UFL]*. arXiv: 2111.00945 [cs.MS].

Helfer, Thomas et al. (2024). *Invariant-based implementation of the Mohr-Coulomb elasto-plastic model in OpenGeoSys using MFront*. TFEL/MFront. URL: https://thelfer.github.io/tfel/web/MohrCoulomb.html (visited on 05/10/2024).