Forward-Mode Enzyme in Developing Constitutive Models with Ratel

Leila Ghaffari, William Moses, Jeremy L Thompson, Karen Stengel, Rezgar Shakeri, and Jed Brown

Department of Computer Science
University of Colorado Boulder

SIAM CSE23
Feb 28, 2023

Differentiation in Solid Mechanics

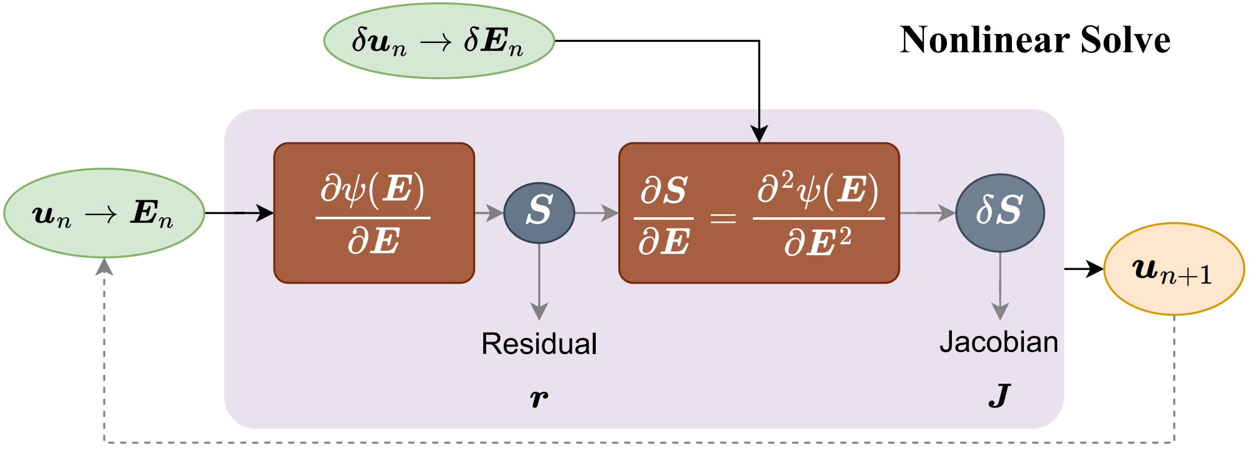
Free energy functional and PDE solvers

$$\psi\left(\boldsymbol{E}\right) = \frac{\lambda}{4}\left(J^2 - 1 - 2\log J\right) - \mu\left(\log J + \operatorname{trace}\boldsymbol{E}\right), \quad J = \sqrt{\left|\boldsymbol{I} + 2\boldsymbol{E}\right|}$$

Differentiation in Solid Mechanics

Free energy functional and PDE solvers

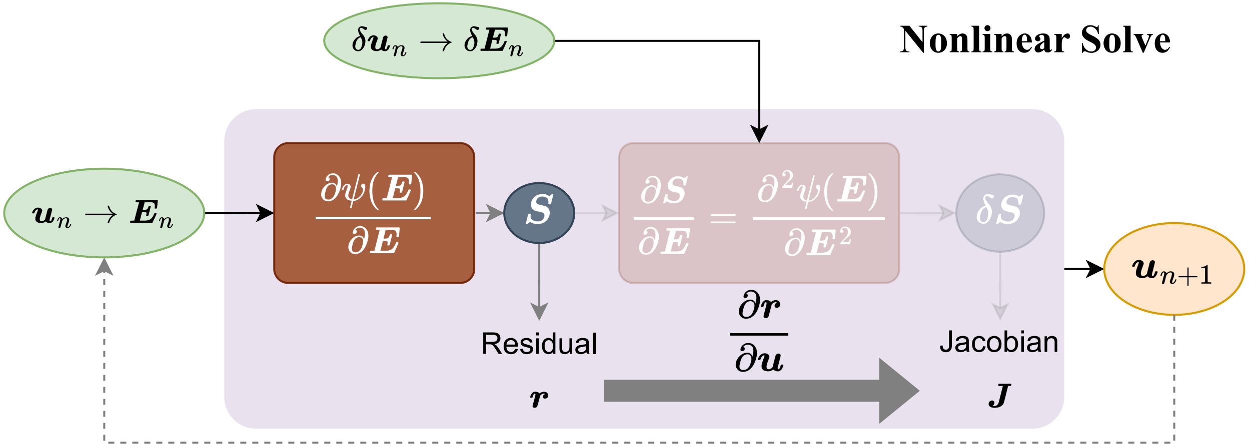
$$\psi\left(\boldsymbol{E}\right) = \frac{\lambda}{4}\left(J^2 - 1 - 2\log J\right) - \mu\left(\log J + \operatorname{trace}\boldsymbol{E}\right), \quad J = \sqrt{|\boldsymbol{I} + 2\boldsymbol{E}|}$$



Differentiation in Solid Mechanics

Free energy functional and PDE solvers

$$\psi(\boldsymbol{E}) = \frac{\lambda}{4}\left(J^2 - 1 - 2\log J\right) - \mu\left(\log J + \operatorname{trace} \boldsymbol{E}\right), \quad J = \sqrt{|\boldsymbol{I} + 2\boldsymbol{E}|}$$

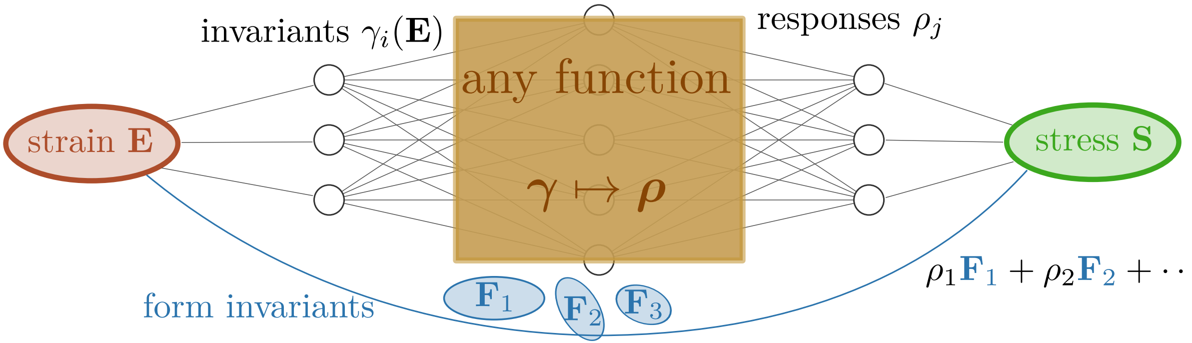


$$\psi\left(\boldsymbol{E}\right) = \frac{\lambda}{4}\left(J^2 - 1 - 2\log J\right) - \mu\left(\log J + \operatorname{trace}\boldsymbol{E}\right), \quad J = \sqrt{\left|\boldsymbol{I} + 2\boldsymbol{E}\right|}$$

Differentiation in Solid Mechanics

Free energy functional and Inverse problems

$$\psi(\boldsymbol{E}) = \frac{\lambda}{4}\left(J^2 - 1 - 2\log J\right) - \mu\left(\log J + \operatorname{trace}\boldsymbol{E}\right), \quad J = \sqrt{|\boldsymbol{I} + 2\boldsymbol{E}|}$$



ABAQUS

UHYPER: USER SUBROUTINE

Variables to be defined

`U(1)`

U, strain energy density function. For a compressible material, at least one derivative involving J should be nonzero. For an incompressible material, all derivatives involving J will be ignored. The strain invariants—$\bar{I}_1$, $\bar{I}_2$, and J—are defined in Hyperelastic behavior of rubberlike materials.

`U(2)`

$\tilde{U}_{dev}$, the deviatoric part of the strain energy density of the primary material response. This quantity is needed only if the current material definition also includes Mullins effect (see Mullins effect).

`UI1(1)`

$\partial U / \partial \bar{I}_1$.

`UI1(2)`

$\partial U / \partial \bar{I}_2$.

`UI1(3)`

$\partial U / \partial J$.

`UI2(1)`

$\partial^2 U / \partial \bar{I}_1^{\,2}$.

`UI2(2)`

$\partial^2 U / \partial \bar{I}_2^{\,2}$.

`UI2(3)`

$\partial^2 U / \partial J^2$.

`UI2(4)`

$\partial^2 U / \partial \bar{I}_1 \partial \bar{I}_2$.

`UI2(5)`

$\partial^2 U / \partial \bar{I}_1 \partial J$.

`UI2(6)`

$\partial^2 U / \partial \bar{I}_2 \partial J$.

`UI3(1)`

$\partial^3 U / \partial \bar{I}_1^{\,2} \partial J$.

`UI3(2)`

$\partial^3 U / \partial \bar{I}_2^{\,2} \partial J$.

`UI3(3)`

$\partial^3 U / \partial \bar{I}_1 \partial \bar{I}_2 \partial J$.

`UI3(4)`

$\partial^3 U / \partial \bar{I}_1 \partial J^2$.

`UI3(5)`

$\partial^3 U / \partial \bar{I}_2 \partial J^2$.

`UI3(6)`

$\partial^3 U / \partial J^3$.

- Fully automated commercial package (Solid Mechanics, FEM)

Abaqus

UHYPER: User Subroutine

Variables to be defined

`U(1)`

U, strain energy density function. For a compressible material, at least one derivative involving J should be nonzero. For an incompressible material, all derivatives involving J will be ignored. The strain invariants—$\bar{I}_1$, $\bar{I}_2$, and J—are defined in Hyperelastic behavior of rubberlike materials.

`U(2)`

$\tilde{U}_{dev}$, the deviatoric part of the strain energy density of the primary material response. This quantity is needed only if the current material definition also includes Mullins effect (see Mullins effect).

`UI1(1)`

$\partial U / \partial \bar{I}_1$.

`UI1(2)`

$\partial U / \partial \bar{I}_2$.

`UI1(3)`

$\partial U / \partial J$.

`UI2(1)`

$\partial^2 U / \partial \bar{I}_1^{\,2}$.

`UI2(2)`

$\partial^2 U / \partial \bar{I}_2^{\,2}$.

`UI2(3)`

$\partial^2 U / \partial J^2$.

`UI2(4)`

$\partial^2 U / \partial \bar{I}_1 \partial \bar{I}_2$.

`UI2(5)`

$\partial^2 U / \partial \bar{I}_1 \partial J$.

`UI2(6)`

$\partial^2 U / \partial \bar{I}_2 \partial J$.

`UI3(1)`

$\partial^3 U / \partial \bar{I}_1^{\,2} \partial J$.

`UI3(2)`

$\partial^3 U / \partial \bar{I}_2^{\,2} \partial J$.

`UI3(3)`

$\partial^3 U / \partial \bar{I}_1 \partial \bar{I}_2 \partial J$.

`UI3(4)`

$\partial^3 U / \partial \bar{I}_1 \partial J^2$.

`UI3(5)`

$\partial^3 U / \partial \bar{I}_2 \partial J^2$.

`UI3(6)`

$\partial^3 U / \partial J^3$.

- Fully automated commercial package (Solid Mechanics, FEM)
- Complex interface (too many inputs)

Abaqus

UHYPER: User subroutine

Variables to be defined

`U(1)`

U, strain energy density function. For a compressible material, at least one derivative involving J should be nonzero. For an incompressible material, all derivatives involving J will be ignored. The strain invariants—$\bar{I}_1$, $\bar{I}_2$, and J—are defined in Hyperelastic behavior of rubberlike materials.

`U(2)`

$\tilde{U}_{dev}$, the deviatoric part of the strain energy density of the primary material response. This quantity is needed only if the current material definition also includes Mullins effect (see Mullins effect).

`UI1(1)`

$\partial U / \partial \bar{I}_1$.

`UI1(2)`

$\partial U / \partial \bar{I}_2$.

`UI1(3)`

$\partial U / \partial J$.

`UI2(1)`

$\partial^2 U / \partial \bar{I}_1^{\,2}$.

`UI2(2)`

$\partial^2 U / \partial \bar{I}_2^{\,2}$.

`UI2(3)`

$\partial^2 U / \partial J^2$.

`UI2(4)`

$\partial^2 U / \partial \bar{I}_1 \partial \bar{I}_2$.

`UI2(5)`

$\partial^2 U / \partial \bar{I}_1 \partial J$.

`UI2(6)`

$\partial^2 U / \partial \bar{I}_2 \partial J$.

`UI3(1)`

$\partial^3 U / \partial \bar{I}_1^{\,2} \partial J$.

`UI3(2)`

$\partial^3 U / \partial \bar{I}_2^{\,2} \partial J$.

`UI3(3)`

$\partial^3 U / \partial \bar{I}_1 \partial \bar{I}_2 \partial J$.

`UI3(4)`

$\partial^3 U / \partial \bar{I}_1 \partial J^2$.

`UI3(5)`

$\partial^3 U / \partial \bar{I}_2 \partial J^2$.

`UI3(6)`

$\partial^3 U / \partial J^3$.

- Fully automated commercial package (Solid Mechanics, FEM)
- Complex interface (too many inputs)
- Unstable for small deformation due to the choice of interface design

$$\boldsymbol{F} = \boldsymbol{I} + \nabla_X \boldsymbol{u}$$

Abaqus

UHYPER: User Subroutine

Variables to be defined

`U(1)`

U, strain energy density function. For a compressible material, at least one derivative involving J should be nonzero. For an incompressible material, all derivatives involving J will be ignored. The strain invariants—$\bar{I}_1$, $\bar{I}_2$, and J—are defined in Hyperelastic behavior of rubberlike materials.

`U(2)`

$\tilde{U}_{dev}$, the deviatoric part of the strain energy density of the primary material response. This quantity is needed only if the current material definition also includes Mullins effect (see Mullins effect).

`UI1(1)`

$\partial U / \partial \bar{I}_1$.

`UI1(2)`

$\partial U / \partial \bar{I}_2$.

`UI1(3)`

$\partial U / \partial J$.

`UI2(1)`

$\partial^2 U / \partial \bar{I}_1^{\,2}$.

`UI2(2)`

$\partial^2 U / \partial \bar{I}_2^{\,2}$.

`UI2(3)`

$\partial^2 U / \partial J^2$.

`UI2(4)`

$\partial^2 U / \partial \bar{I}_1 \partial \bar{I}_2$.

`UI2(5)`

$\partial^2 U / \partial \bar{I}_1 \partial J$.

`UI2(6)`

$\partial^2 U / \partial \bar{I}_2 \partial J$.

`UI3(1)`

$\partial^3 U / \partial \bar{I}_1^{\,2} \partial J$.

`UI3(2)`

$\partial^3 U / \partial \bar{I}_2^{\,2} \partial J$.

`UI3(3)`

$\partial^3 U / \partial \bar{I}_1 \partial \bar{I}_2 \partial J$.

`UI3(4)`

$\partial^3 U / \partial \bar{I}_1 \partial J^2$.

`UI3(5)`

$\partial^3 U / \partial \bar{I}_2 \partial J^2$.

`UI3(6)`

$\partial^3 U / \partial J^3$.

- Fully automated commercial package (Solid Mechanics, FEM)
- Complex interface (too many inputs)
- Unstable for small deformation due to the choice of interface design

$$\boldsymbol{F} = \boldsymbol{I} + \nabla_X \boldsymbol{u}$$

- Not easy to change the interface

Abaqus
UHYPER: User Subroutine

Variables to be defined

U(1)

U, strain energy density function. For a compressible material, at least one derivative involving J should be nonzero. For an incompressible material, all derivatives involving J will be ignored. The strain invariants—$\bar{I}_1$, $\bar{I}_2$, and J—are defined in Hyperelastic behavior of rubberlike materials.

U(2)

$\tilde{U}_{dev}$, the deviatoric part of the strain energy density of the primary material response. This quantity is needed only if the current material definition also includes Mullins effect (see Mullins effect).

UI1(1)

$\partial U / \partial \bar{I}_1$.

UI1(2)

$\partial U / \partial \bar{I}_2$.

UI1(3)

$\partial U / \partial J$.

UI2(1)

$\partial^2 U / \partial \bar{I}_1^{\,2}$.

UI2(2)

$\partial^2 U / \partial \bar{I}_2^{\,2}$.

UI2(3)

$\partial^2 U / \partial J^2$.

UI2(4)

$\partial^2 U / \partial \bar{I}_1 \partial \bar{I}_2$.

UI2(5)

$\partial^2 U / \partial \bar{I}_1 \partial J$.

UI2(6)

$\partial^2 U / \partial \bar{I}_2 \partial J$.

UI3

∂

UI3(

∂^3

UI3(3)

$\partial^3 U$

UI3(4)

$\partial^3 U / \partial \bar{I}_1 \partial J^2$.

UI3(5)

$\partial^3 U / \partial \bar{I}_2 \partial J^2$.

UI3(6)

$\partial^3 U / \partial J^3$.

AD helps us to create a more generic interface (one input function, $\psi(\mathbf{E})$).

- Fully automated commercial package (Solid Mechanics, FEM)
- Complex interface
  many inputs)
  able for small
  mation due to the
  e of interface design
  $= \boldsymbol{I} + \nabla_X \boldsymbol{u}$
- Not easy to change the interface

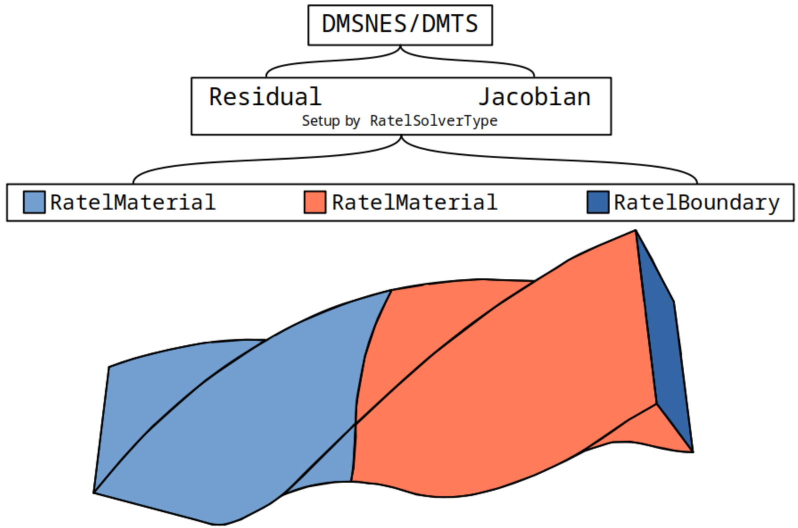
RATEL: EXTENSIBLE, PERFORMANCE-PORTABLE SOLID MECHANICS

HTTPS://GITLAB.COM/MICROMORPH/RATEL

GitLab-CI passed | License BSD 2-Clause | Documentation latest | coverage 96.05%

Features:

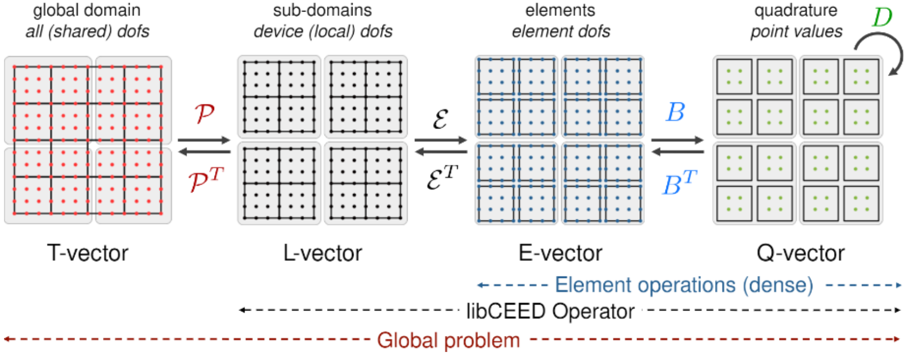
- Linear elasticity
- Neo-Hookean and Mooney-Rivlin Hyperelasticity
- Multi-material
- Static, Quasistatic, Dynamic
- Initial and Current configurations



Ratel: Extensible, performance-portable solid mechanics

Composition and Abstraction - libCEED: https://libceed.org/en/latest/

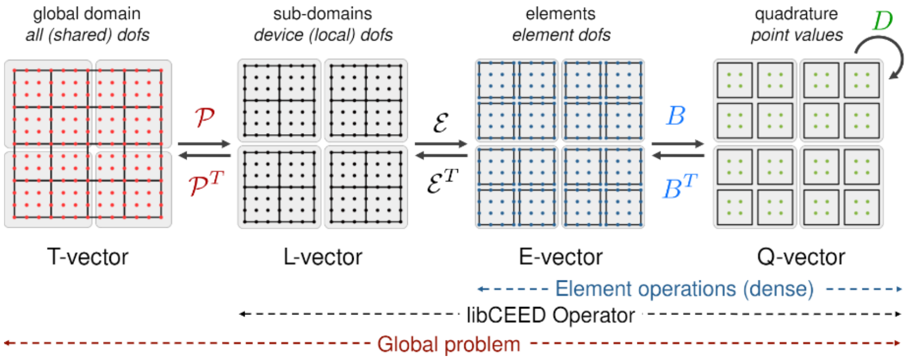
$$A = \mathcal{P}^T \mathcal{E}^T B^T D B \mathcal{E} \mathcal{P}$$



Ratel: Extensible, performance-portable solid mechanics

Composition and Abstraction - libCEED: https://libceed.org/en/latest/

$$A = \mathcal{P}^T \mathcal{E}^T B^T D B \mathcal{E} \mathcal{P}$$

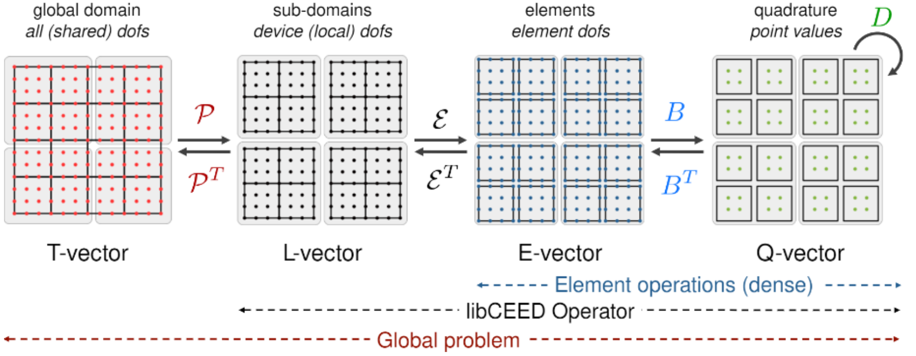


- Purely algebraic high-order FEM

Ratel: Extensible, performance-portable solid mechanics

Composition and Abstraction - libCEED: https://libceed.org/en/latest/

$$A = \mathcal{P}^T \mathcal{E}^T B^T D B \mathcal{E} \mathcal{P}$$

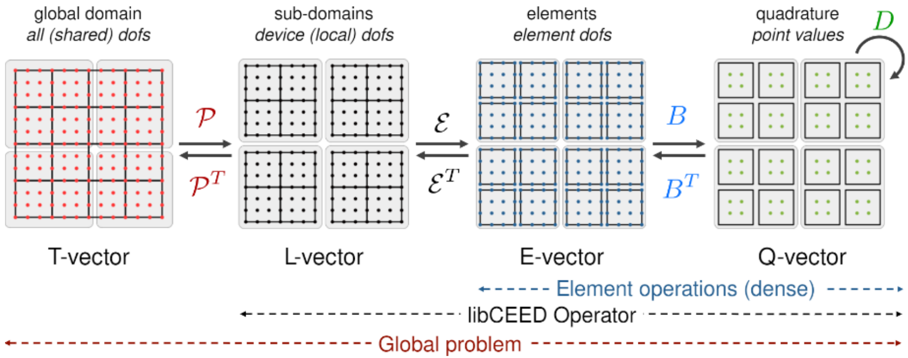


- Purely algebraic high-order FEM
- Single source Vanilla C for physics

RATEL: EXTENSIBLE, PERFORMANCE-PORTABLE SOLID MECHANICS

COMPOSITION AND ABSTRACTION - LIBCEED: HTTPS://LIBCEED.ORG/EN/LATEST/

$$A = \mathcal{P}^T \mathcal{E}^T B^T D B \mathcal{E} \mathcal{P}$$

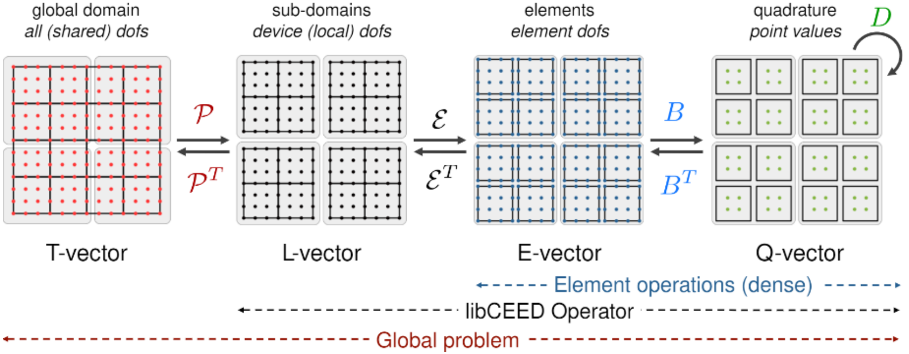


- Purely algebraic high-order FEM
- Single source Vanilla C for physics
- Various CPU and GPU backends

Ratel: Extensible, performance-portable solid mechanics

Composition and Abstraction - libCEED: https://libceed.org/en/latest/

$$A = \mathcal{P}^T \mathcal{E}^T B^T D B \mathcal{E} \mathcal{P}$$

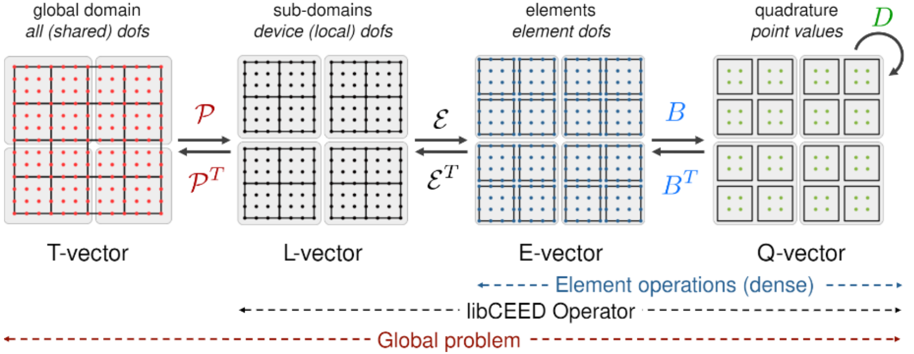


- Purely algebraic high-order FEM
- Single source Vanilla C for physics
- Various CPU and GPU backends
- Backend plugins with run-time selection `./bps -ceed /gpu/cuda`

Ratel: Extensible, performance-portable solid mechanics

Composition and Abstraction - libCEED: https://libceed.org/en/latest/

$$A = \mathcal{P}^T \mathcal{E}^T B^T D B \mathcal{E} \mathcal{P}$$

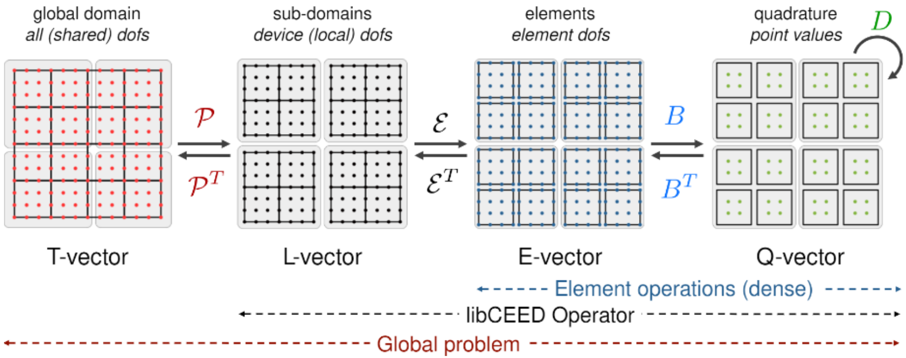


- Purely algebraic high-order FEM
- Single source Vanilla C for physics
- Various CPU and GPU backends
- Backend plugins with run-time selection `./bps -ceed /gpu/cuda`
- Support for Matrix-assembly and Matrix-free

RATEL: EXTENSIBLE, PERFORMANCE-PORTABLE SOLID MECHANICS

COMPOSITION AND ABSTRACTION - LIBCEED: HTTPS://LIBCEED.ORG/EN/LATEST/

$$A = \mathcal{P}^T \mathcal{E}^T B^T D B \mathcal{E} \mathcal{P}$$

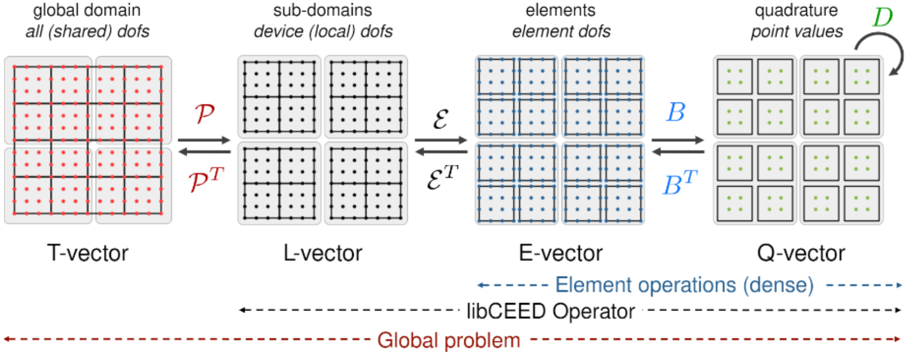


- Purely algebraic high-order FEM
- Single source Vanilla C for physics
- Various CPU and GPU backends
- Backend plugins with run-time selection `./bps -ceed /gpu/cuda`
- Support for Matrix-assembly and Matrix-free
- Operator abstraction

Ratel: Extensible, performance-portable solid mechanics

Composition and Abstraction - libCEED: https://libceed.org/en/latest/

$$A = \mathcal{P}^T \mathcal{E}^T B^T D B \mathcal{E} \mathcal{P}$$



- Purely algebraic high-order FEM
- Single source Vanilla C for physics
- Various CPU and GPU backends
- Backend plugins with run-time selection `./bps -ceed /gpu/cuda`
- Support for Matrix-assembly and Matrix-free
- Operator abstraction
- User choice of data storage at quadrature point

PETSc:

https://petsc.org/release/

- Parallel solution of PDEs

PETSc:

https://petsc.org/release/

- Parallel solution of PDEs
- CPUs (MPI)

PETSc:

https://petsc.org/release/

- Parallel solution of PDEs
- CPUs (MPI)
- GPUs (CUDA, HIP, and OpenCL)

PETSc:

https://petsc.org/release/

- Parallel solution of PDEs
- CPUs (MPI)
- GPUs (CUDA, HIP, and OpenCL)
- Hybrid MPI-GPU

PETSc:

https://petsc.org/release/

- Parallel solution of PDEs
- CPUs (MPI)
- GPUs (CUDA, HIP, and OpenCL)
- Hybrid MPI-GPU
- Optimization (PETSc/Tao)

Enzyme AD:

https://enzyme.mit.edu/

- High-Performance Automatic Differentiation

Enzyme AD:

https://enzyme.mit.edu/

- High-Performance Automatic Differentiation
- Work at the LLVM level

Enzyme AD:

https://enzyme.mit.edu/

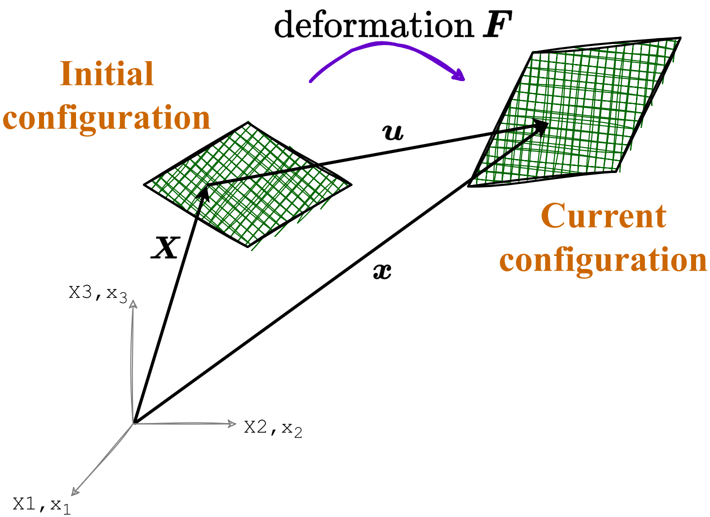
- High-Performance Automatic Differentiation
- Work at the LLVM level
- Support for variety of languages (C/C++, Julia, Rust, Fortran, etc)

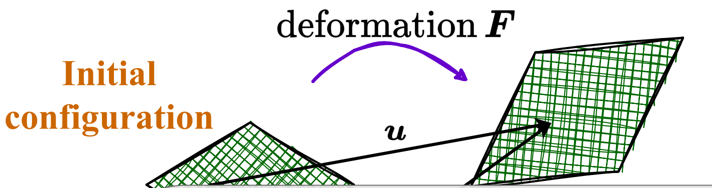
Enzyme AD:

https://enzyme.mit.edu/

- High-Performance Automatic Differentiation
- Work at the LLVM level
- Support for variety of languages (C/C++, Julia, Rust, Fortran, etc)
- *reverse* and *forward* mode AD

Initial vs Current configuration





$$\overbrace{\nabla_X \boldsymbol{v} : \boldsymbol{F}\boldsymbol{S}}^{\text{Initial Residual}} \xrightarrow{\text{push forward}} \overbrace{\nabla_x \boldsymbol{v} : \boldsymbol{\tau}}^{\text{Current Residual}}$$

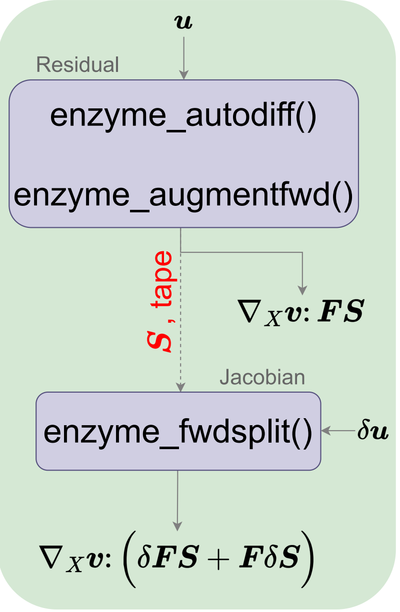
$$\Big\downarrow \text{linearize} \qquad\qquad\qquad\qquad \Big\downarrow \text{linearize}$$

$$\underbrace{\nabla_X \boldsymbol{v} : \left(\delta \boldsymbol{F}\boldsymbol{S} + \boldsymbol{F}\delta\boldsymbol{S}\right)}_{\text{Initial Jacobian}} \xrightarrow{\text{push forward}} \underbrace{\nabla_x \boldsymbol{v} : \left(\delta\boldsymbol{\tau} - \boldsymbol{\tau}(\nabla_x \delta\boldsymbol{u})^T\right)}_{\text{Current Jacobian}}$$

Ratel - Enzyme AD

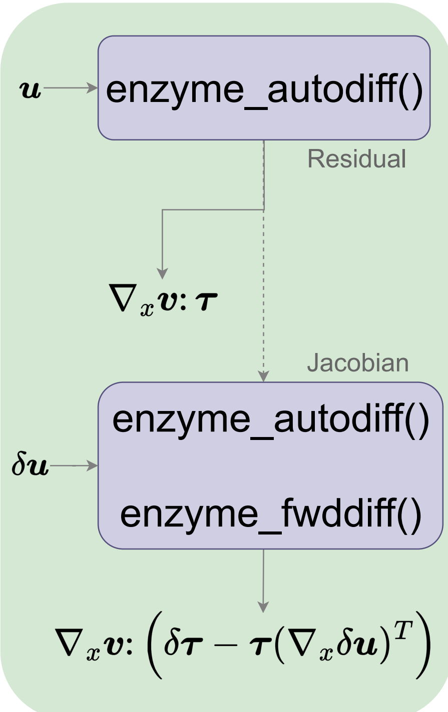
Initial configuration - Reverse and Forward Split

```
// S = d(\psi) / d(E) [Reverse mode]
void SecondPiolaKirchhoffStress_NeoHookean_AD(...) {
  __enzyme_autodiff((void *)StrainEnergy, ...);
}

// Call forward S and return tape
__enzyme_augmentfwd(
(void *)SecondPiolaKirchhoffStress_NeoHookean_AD,
enzyme_allocated, tape_bytes, enzyme_tape, tape,
enzyme_nofree, ...);

// Compute dS using the stored tape [Forward-split]
__enzyme_fwdsplit(
(void *)SecondPiolaKirchhoffStress_NeoHookean_AD,
enzyme_allocated, tape_bytes, enzyme_tape, tape, ...);
```



Ratel - Enzyme AD

Current configuration - Reverse and Forward




```
// Compute tau = (dPsi / de) * (2 e + I) [Reverse]
void Kirchhofftau_Voigt_NeoHookean_AD(...) {
  __enzyme_autodiff((void *)StrainEnergy, ...);
  ...
  for (int j = 0; j < 6; j++)
    b_Voigt[j] = 2 * e_Voigt[j] + (j < 3);
  ...
  RatelMatMatMult(1., dPsi, b, tau);
}

// Compute dtau [Forward]
CEED_QFUNCTION_HELPER void dtau_fwd(...) {
  __enzyme_fwddiff(
  (void *)Kirchhofftau_Voigt_NeoHookean_AD, ...);
}
```

Ratel - Enzyme AD

Performance for different Jacobian representations

Problem	Storage	Scalars	Time (s)
`current`	$W; \nabla_x \boldsymbol{\xi}, \boldsymbol{\tau}, J-1$	17	36.2
`initial`	$W, \nabla_X \boldsymbol{\xi}; \nabla_X \boldsymbol{u}$	19	48.4
`initial-AD`	$W, \nabla_X \boldsymbol{\xi}; \nabla_X \boldsymbol{u}, S,$ `tape`	31	53.9
`current-AD`	$W; \nabla_x \boldsymbol{\xi}, \boldsymbol{e}$	16	55.8

Current-Analytical

$\nabla_X \boldsymbol{u}$

$$\boldsymbol{e} = \boldsymbol{e}(\nabla_X \boldsymbol{u})$$

$$J = J(\nabla_X \boldsymbol{u})$$

$$\boldsymbol{\tau} = \frac{\lambda}{2}\left(J^2 - 1\right)\boldsymbol{I} + 2\mu e$$

Res

$\boldsymbol{\tau}, \nabla_x X, J - 1$

$\nabla_X \delta \boldsymbol{u}$

$$\delta \boldsymbol{\epsilon} = \delta \boldsymbol{\epsilon}(\nabla_x \delta \boldsymbol{u})$$

$$Jac = Jac(\nabla_x \delta \boldsymbol{u}, J, \boldsymbol{\tau}, \delta \boldsymbol{\epsilon})$$

Jac

Current-AD

$\nabla_X \boldsymbol{u}$

$$\boldsymbol{e} = \boldsymbol{e}(\nabla_X \boldsymbol{u})$$

$$\boldsymbol{\tau} = \frac{\partial \psi}{\partial \boldsymbol{e}}\,(2\boldsymbol{e} + \boldsymbol{I})$$

Res

$\boldsymbol{e}, \nabla_x X$

$\nabla_X \delta \boldsymbol{u}$

$$\boldsymbol{\tau} = \frac{\partial \psi}{\partial \boldsymbol{e}}\,(2\boldsymbol{e} + \boldsymbol{I})$$

$$\delta \boldsymbol{\tau} = \frac{\partial \boldsymbol{\tau}}{\partial \boldsymbol{e}}$$

Jac

Current-Analytical

$\nabla_X \boldsymbol{u}$

$$\boldsymbol{e} = \boldsymbol{e}(\nabla_X \boldsymbol{u})$$

$$J = J(\nabla_X \boldsymbol{u})$$

$$\boldsymbol{\tau} = \frac{\lambda}{2}\left(J^2 - 1\right)\boldsymbol{I} + 2\mu e$$

Res

$\boldsymbol{\tau}, \nabla_x X, J - 1$

$\nabla_X \delta\boldsymbol{u}$

$$\delta\boldsymbol{\epsilon} = \delta\boldsymbol{\epsilon}(\nabla_x \delta\boldsymbol{u},$$

No need for $\delta\boldsymbol{\tau}$

$$Jac = Jac(\nabla_x \delta\boldsymbol{u}, J, \boldsymbol{\tau}, \delta\boldsymbol{\epsilon})$$

Jac

Current-AD

$\nabla_X \boldsymbol{u}$

$$\boldsymbol{e} = \boldsymbol{e}(\nabla_X \boldsymbol{u})$$

$$\boldsymbol{\tau} = \frac{\partial \psi}{\partial \boldsymbol{e}}\left(2\boldsymbol{e} + \boldsymbol{I}\right)$$

Res

$\boldsymbol{e}, \nabla_x X$

$\nabla_X \delta\boldsymbol{u}$

$$\boldsymbol{\tau} = \frac{\partial \psi}{\partial \boldsymbol{e}}\left(2\boldsymbol{e} + \boldsymbol{I}\right)$$

$$\delta\boldsymbol{\tau} = \frac{\partial \boldsymbol{\tau}}{\partial \boldsymbol{e}}$$

Jac

Current-Analytical

$\nabla_X \boldsymbol{u}$

$$\boldsymbol{e} = \boldsymbol{e}(\nabla_X \boldsymbol{u})$$

$$J = J(\nabla_X \boldsymbol{u})$$

$$\boldsymbol{\tau} = \frac{\lambda}{2}\left(J^2 - 1\right)\boldsymbol{I} + 2\mu e$$

Res

$\boldsymbol{\tau}, \nabla_x X, J-1$

$\nabla_X \delta \boldsymbol{u}$

$$\delta\boldsymbol{\epsilon} = \delta\boldsymbol{\epsilon}(\nabla_x \delta \boldsymbol{u},$$

No need for

$\delta\boldsymbol{\tau}$

$$Jac = Jac(\nabla_x \delta \boldsymbol{u}, J, \boldsymbol{\tau}, \delta\boldsymbol{\epsilon})$$

Jac

Current-AD

$\nabla_X \boldsymbol{u}$

$$\boldsymbol{e} = \boldsymbol{e}(\nabla_X \boldsymbol{u})$$

$$\boldsymbol{\tau} = \frac{\partial \psi}{\partial \boldsymbol{e}}(2\boldsymbol{e} + \boldsymbol{I})$$

Res

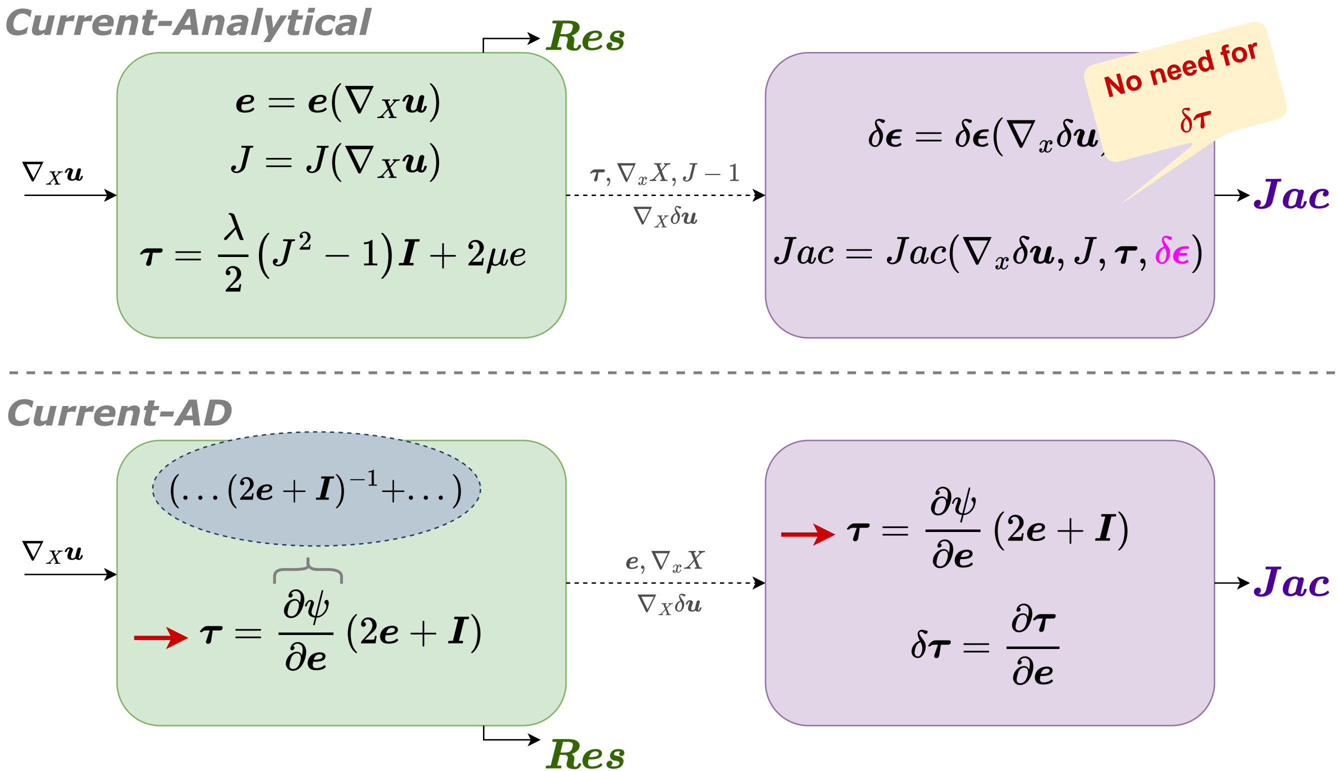
$\boldsymbol{e}, \nabla_x X$

$\nabla_X \delta \boldsymbol{u}$

$$\boldsymbol{\tau} = \frac{\partial \psi}{\partial \boldsymbol{e}}(2\boldsymbol{e} + \boldsymbol{I})$$

$$\delta\boldsymbol{\tau} = \frac{\partial \boldsymbol{\tau}}{\partial \boldsymbol{e}}$$

Jac


Current-Analytical
$\nabla_X \boldsymbol{u}$
$\boldsymbol{e} = \boldsymbol{e}(\nabla_X \boldsymbol{u})$
$J = J(\nabla_X \boldsymbol{u})$
$\boldsymbol{\tau} = \frac{\lambda}{2}\left(J^2 - 1\right)\boldsymbol{I} + 2\mu e$
Res
$\boldsymbol{\tau}, \nabla_x X, J-1$
$\nabla_X \delta\boldsymbol{u}$
$\delta\boldsymbol{\epsilon} = \delta\boldsymbol{\epsilon}(\nabla_x \delta\boldsymbol{u})$
No need for
$\delta\boldsymbol{\tau}$
$Jac = Jac(\nabla_x \delta\boldsymbol{u}, J, \boldsymbol{\tau}, \delta\boldsymbol{\epsilon})$
Jac
Current-AD
$\nabla_X \boldsymbol{u}$
$(\ldots(2\boldsymbol{e}+\boldsymbol{I})^{-1}+\ldots)$
$\boldsymbol{\tau} = \frac{\partial\psi}{\partial\boldsymbol{e}}(2\boldsymbol{e}+\boldsymbol{I})$
Res
$\boldsymbol{e}, \nabla_x X$
$\nabla_X \delta\boldsymbol{u}$
$\boldsymbol{\tau} = \frac{\partial\psi}{\partial\boldsymbol{e}}(2\boldsymbol{e}+\boldsymbol{I})$
$\delta\boldsymbol{\tau} = \frac{\partial\boldsymbol{\tau}}{\partial\boldsymbol{e}}$
Jac

Current-Analytical

$\nabla_X \boldsymbol{u} \rightarrow$

$$\boldsymbol{e} = \boldsymbol{e}(\nabla_X \boldsymbol{u})$$

$$J = J(\nabla_X \boldsymbol{u})$$

$$\boldsymbol{\tau} = \frac{\lambda}{2}\left(J^2 - 1\right)\boldsymbol{I} + 2\mu e$$

$\rightarrow$ ***Res***

$\xrightarrow[\nabla_X \delta\boldsymbol{u}]{\boldsymbol{\tau},\, \nabla_x X,\, J-1}$

$$\delta\boldsymbol{\epsilon} = \delta\boldsymbol{\epsilon}(\nabla_x \delta\boldsymbol{u},$$

No need for $\delta\boldsymbol{\tau}$

$$Jac = Jac(\nabla_x \delta\boldsymbol{u}, J, \boldsymbol{\tau}, \delta\boldsymbol{\epsilon})$$

$\rightarrow$ ***Jac***

Current-AD

$\nabla_X \boldsymbol{u} \rightarrow$

$$\boldsymbol{\tau} = \overbrace{\frac{\partial \psi}{\partial \boldsymbol{e}}}^{(\dots(2\boldsymbol{e}+\boldsymbol{I})^{-1}+\dots)}(2\boldsymbol{e}+\boldsymbol{I})$$

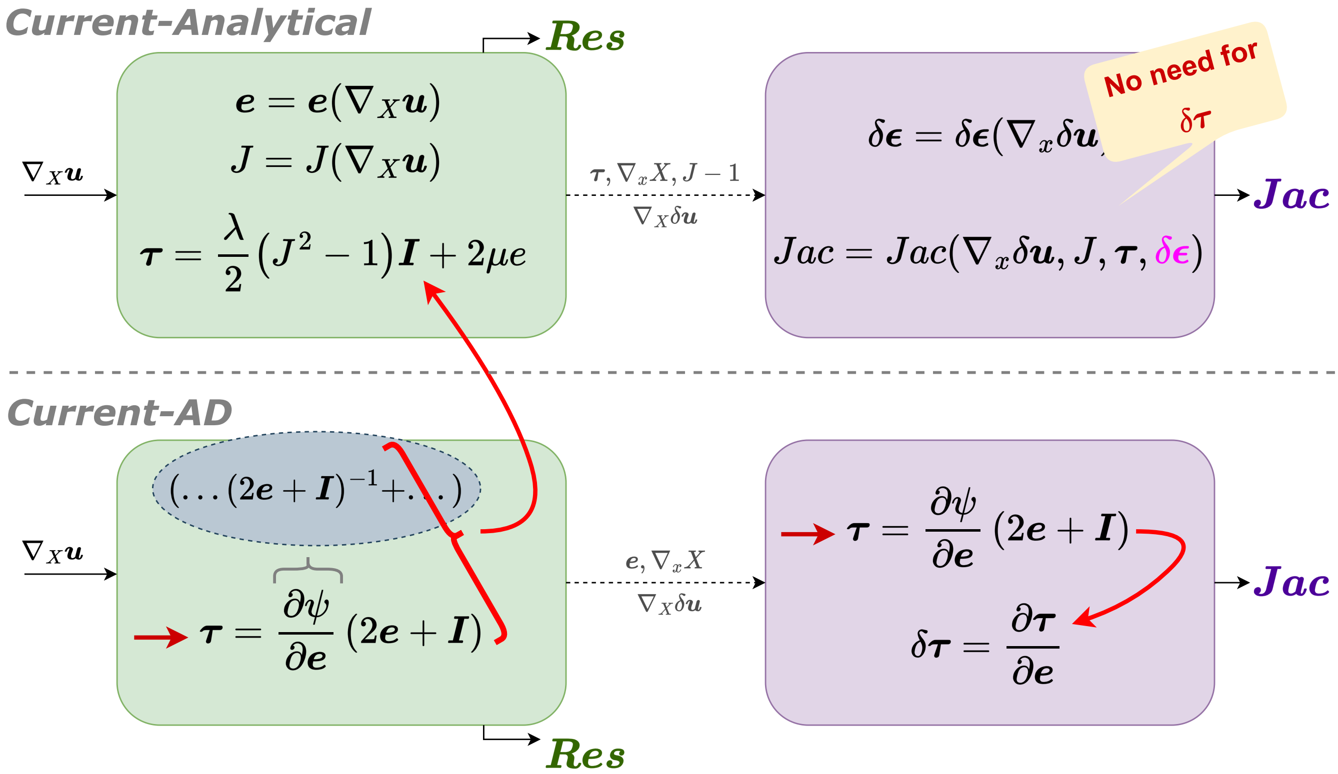
$\rightarrow$ ***Res***

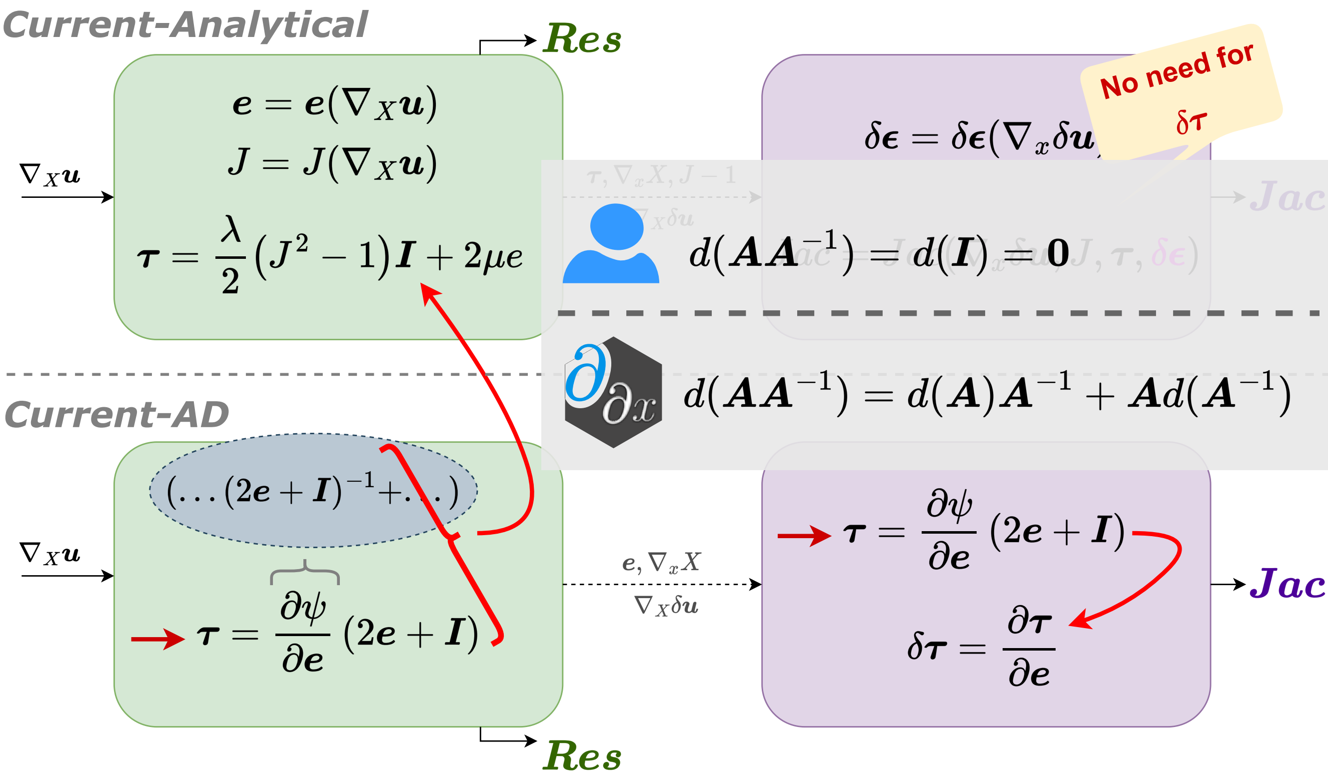
$\xrightarrow[\nabla_X \delta\boldsymbol{u}]{\boldsymbol{e},\, \nabla_x X}$

$$\boldsymbol{\tau} = \frac{\partial \psi}{\partial \boldsymbol{e}}(2\boldsymbol{e}+\boldsymbol{I})$$

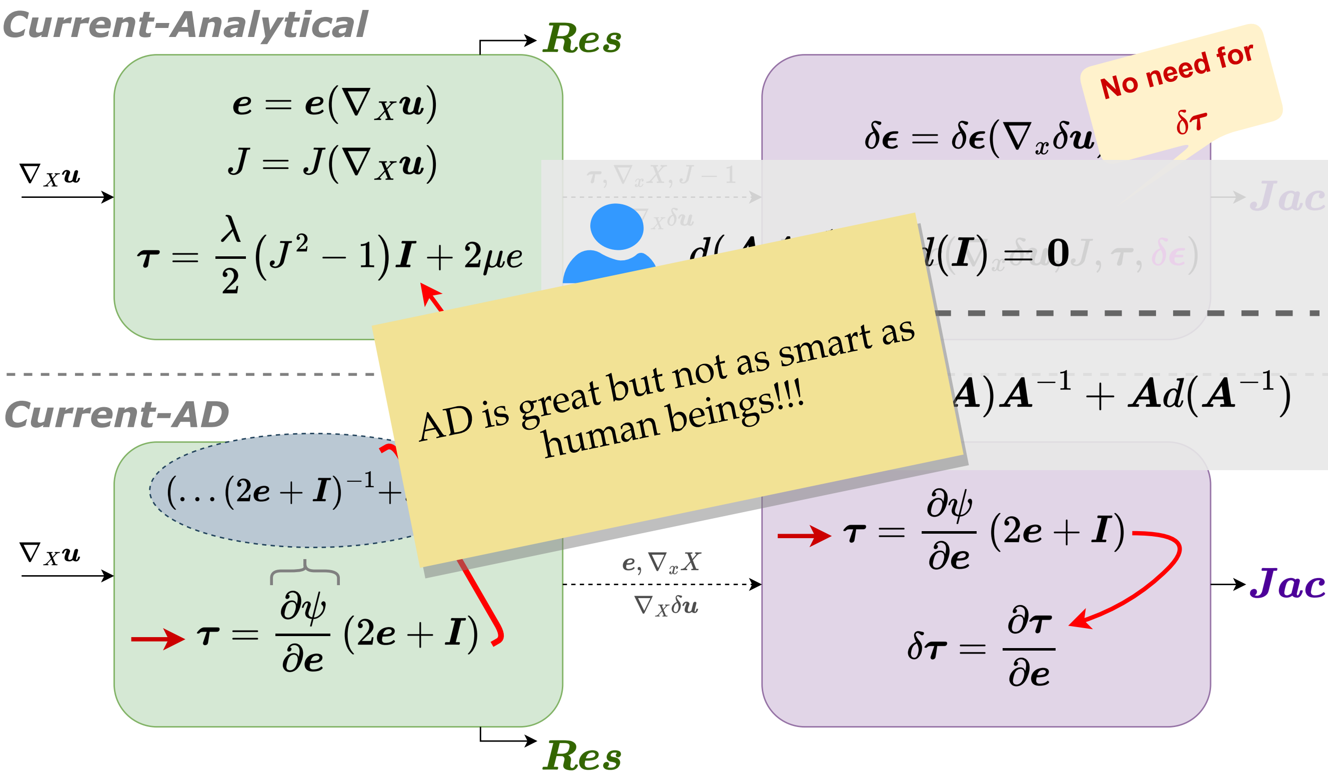
$$\delta\boldsymbol{\tau} = \frac{\partial \boldsymbol{\tau}}{\partial \boldsymbol{e}}$$

$\rightarrow$ ***Jac***


Current-Analytical
Res
$\boldsymbol{e} = \boldsymbol{e}(\nabla_X \boldsymbol{u})$
$J = J(\nabla_X \boldsymbol{u})$
$\boldsymbol{\tau} = \frac{\lambda}{2}\big(J^2 - 1\big)\boldsymbol{I} + 2\mu e$
$\nabla_X \boldsymbol{u}$
$\boldsymbol{\tau}, \nabla_x X, J-1$
$\nabla_X \delta \boldsymbol{u}$
$\delta\boldsymbol{\epsilon} = \delta\boldsymbol{\epsilon}(\nabla_x \delta \boldsymbol{u})$
No need for $\delta\boldsymbol{\tau}$
$Jac = Jac(\nabla_x \delta \boldsymbol{u}, J, \boldsymbol{\tau}, \delta\boldsymbol{\epsilon})$
Jac
Current-AD
$(\ldots(2\boldsymbol{e}+\boldsymbol{I})^{-1}+\ldots)$
$\nabla_X \boldsymbol{u}$
$\boldsymbol{\tau} = \frac{\partial\psi}{\partial \boldsymbol{e}}(2\boldsymbol{e}+\boldsymbol{I})$
$\boldsymbol{e}, \nabla_x X$
$\nabla_X \delta \boldsymbol{u}$
$\boldsymbol{\tau} = \frac{\partial\psi}{\partial \boldsymbol{e}}(2\boldsymbol{e}+\boldsymbol{I})$
$\delta\boldsymbol{\tau} = \frac{\partial \boldsymbol{\tau}}{\partial \boldsymbol{e}}$
Jac
Res



Current-Analytical
Res
$e = e(\nabla_X u)$
$J = J(\nabla_X u)$
$\tau = \frac{\lambda}{2}(J^2-1)I + 2\mu e$
$\nabla_X u$
No need for $\delta\tau$
$\delta\epsilon = \delta\epsilon(\nabla_x \delta u)$
Jac
$d(AA^{-1}) = d(I) = 0$
$d(AA^{-1}) = d(A)A^{-1} + Ad(A^{-1})$
Current-AD
$(\dots(2e+I)^{-1}+\dots)$
$\tau = \frac{\partial\psi}{\partial e}(2e+I)$
$e, \nabla_x X$
$\nabla_X \delta u$
$\tau = \frac{\partial\psi}{\partial e}(2e+I)$
$\delta\tau = \frac{\partial\tau}{\partial e}$
Jac
Res



Current-Analytical
Res
$\boldsymbol{e} = \boldsymbol{e}(\nabla_X \boldsymbol{u})$
$J = J(\nabla_X \boldsymbol{u})$
$\boldsymbol{\tau} = \frac{\lambda}{2}(J^2 - 1)\boldsymbol{I} + 2\mu e$
$\nabla_X \boldsymbol{u}$
$\delta\boldsymbol{\epsilon} = \delta\boldsymbol{\epsilon}(\nabla_x \delta\boldsymbol{u}$
No need for $\delta\boldsymbol{\tau}$
$\boldsymbol{\tau}, \nabla_x X, J-1$
$\nabla_X \delta u$
Jac
$d(\boldsymbol{I}) = \boldsymbol{0}$
$\boldsymbol{A})\boldsymbol{A}^{-1} + \boldsymbol{A}d(\boldsymbol{A}^{-1})$
AD is great but not as smart as human beings!!!
Current-AD
$(\ldots(2\boldsymbol{e}+\boldsymbol{I})^{-1}+$
$\nabla_X \boldsymbol{u}$
$\boldsymbol{\tau} = \frac{\partial\psi}{\partial\boldsymbol{e}}(2\boldsymbol{e}+\boldsymbol{I})$
$\boldsymbol{e}, \nabla_x X$
$\nabla_X \delta\boldsymbol{u}$
$\boldsymbol{\tau} = \frac{\partial\psi}{\partial\boldsymbol{e}}(2\boldsymbol{e}+\boldsymbol{I})$
$\delta\boldsymbol{\tau} = \frac{\partial\boldsymbol{\tau}}{\partial\boldsymbol{e}}$
Jac
Res

OUTLOOK

TOWARDS PLASTICITY

$$\text{Input Scalar Functions} = \begin{cases} \psi(\boldsymbol{E}; \mathbb{I}) & \leftarrow \text{ free energy} \\ \phi(\boldsymbol{S}; \mathbb{I}) & \leftarrow \text{ dissipation potential} \\ f(\boldsymbol{S}; \mathbb{I}) & \leftarrow \text{ yield surface} \end{cases}$$

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research under Award Number DE-SC0016140.

Registration and travel support for this presentation was provided by the Society for Industrial and Applied Mathematics.