Mechanical behaviours in MFront

MFront training course

[1] CEA, DES, IRESNE, DEC, SESC, LSC, Cadarache, France

Thomas Helfer[1]

Outline

Mechanical behaviours

An overview of `MFront`

Conclusions and perspectives

Sommaire

Mechanical behaviours

An overview of MFront

Conclusions and perspectives

Role of the mechanical behaviours

- Mechanical equilibrium: find$\Delta\vec{\mathbb{U}}$ such as:

$$\vec{\mathbb{R}}\left(\Delta\vec{\mathbb{U}}\right)=\vec{\mathbb{O}}\quad\text{avec}\quad\vec{\mathbb{R}}\left(\Delta\vec{\mathbb{U}}\right)=\vec{\mathbb{F}}_i\left(\Delta\vec{\mathbb{U}}\right)-\vec{\mathbb{F}}_e$$

Role of the mechanical behaviours

- Mechanical equilibrium: find$\Delta\vec{\mathbb{U}}$ such as:

$$\vec{\mathbb{R}}\left(\Delta\vec{\mathbb{U}}\right)=\vec{\mathbb{O}}\quad\text{avec}\quad\vec{\mathbb{R}}\left(\Delta\vec{\mathbb{U}}\right)=\vec{\mathbb{F}}_{i}\left(\Delta\vec{\mathbb{U}}\right)-\vec{\mathbb{F}}_{e}$$

- element contribution to inner forces:

$$\begin{aligned}\vec{\mathbb{F}}^{e}_{i}&=\int_{V^{e}}\underline{\sigma}_{t+\Delta t}\left(\Delta\,\underline{\epsilon}^{to},\Delta\,t\right)\,:\,\underline{\mathbf{B}}\,\mathrm{d}V\\&=\sum_{i=1}^{N^{G}}\left(\underline{\sigma}_{t+\Delta\,t}\left(\Delta\underline{\epsilon}^{to}\left(\vec{\eta}_{i}\right),\Delta\,t\right)\,:\,\underline{\underline{\mathbf{B}}}\left(\vec{\eta}_{i}\right)\right)w_{i}\end{aligned}$$

where $\underline{\mathbf{B}}$ gives the relationship between $\Delta\,\underline{\epsilon}^{to}$ and $\Delta\vec{\mathbb{U}}$

Role of the mechanical behaviours

- Mechanical equilibrium: find$\Delta\vec{\mathbb{U}}$ such as:

$$\vec{\mathbb{R}}\left(\Delta\vec{\mathbb{U}}\right)=\vec{\mathbb{O}}\quad\text{avec}\quad\vec{\mathbb{R}}\left(\Delta\vec{\mathbb{U}}\right)=\vec{\mathbb{F}}_{i}\left(\Delta\vec{\mathbb{U}}\right)-\vec{\mathbb{F}}_{e}$$

- element contribution to inner forces:

$$\vec{\mathbb{F}}_{i}^{e}=\sum_{i=1}^{N^{G}}\left(\underline{\sigma}_{t+\Delta\,t}\left(\Delta\underline{\epsilon}^{to}\left(\vec{\eta}_{i}\right),\Delta\,t\right):\underline{\underline{\mathbf{B}}}\left(\vec{\eta}_{i}\right)\right)w_{i}$$

- Resolution using the Newton-Raphson algorithm:

$$\Delta\vec{\mathbb{U}}^{n+1}=\Delta\vec{\mathbb{U}}^{n}-\left(\left.\frac{\partial\vec{\mathbb{R}}}{\partial\Delta\vec{\mathbb{U}}}\right|_{\Delta\vec{\mathbb{U}}^{n}}\right)^{-1}.\vec{\mathbb{R}}\left(\Delta\vec{\mathbb{U}}^{n}\right)=\Delta\vec{\mathbb{U}}^{n}-\underline{\underline{\mathbb{K}}}^{-1}.\vec{\mathbb{R}}\left(\Delta\vec{\mathbb{U}}^{n}\right)$$

Role of the mechanical behaviours

- Mechanical equilibrium: find$\Delta\vec{\mathbb{U}}$ such as:

$$\vec{\mathbb{R}}\left(\Delta\vec{\mathbb{U}}\right)=\vec{\mathbb{O}}\quad\text{avec}\quad\vec{\mathbb{R}}\left(\Delta\vec{\mathbb{U}}\right)=\vec{\mathbb{F}}_{i}\left(\Delta\vec{\mathbb{U}}\right)-\vec{\mathbb{F}}_{e}$$

- element contribution to inner forces:

$$\vec{\mathbb{F}}^{e}_{i}=\sum_{i=1}^{N^{G}}\left(\underline{\sigma}_{t+\Delta\,t}\left(\Delta\underline{\epsilon}^{to}\left(\vec{\eta}_{i}\right),\Delta\,t\right)\,:\,\underline{\underline{\mathbf{B}}}\left(\vec{\eta}_{i}\right)\right)w_{i}$$

- Resolution using the Newton-Raphson algorithm:

$$\Delta\vec{\mathbb{U}}^{n+1}=\Delta\vec{\mathbb{U}}^{n}-\underline{\underline{\mathbb{K}}}^{-1}.\vec{\mathbb{R}}\left(\Delta\vec{\mathbb{U}}^{n}\right)$$

- element contribution to the stiffness:

$$\underline{\underline{\mathbb{K}}}^{e}=\sum_{i=1}^{N^{G}}{}^{t}\underline{\underline{\mathbf{B}}}\left(\vec{\eta}_{i}\right)\,:\,\frac{\partial\Delta\underline{\sigma}}{\partial\Delta\underline{\epsilon}^{to}}\left(\vec{\eta}_{i}\right)\,:\,\underline{\underline{\mathbf{B}}}\left(\vec{\eta}_{i}\right)w_{i}$$

$\dfrac{\partial\Delta\underline{\sigma}}{\partial\Delta\underline{\epsilon}^{to}}$ is the **consistent tangent operator**

Main functions of the mechanical behaviour

$$\left(\left.\underline{\epsilon}^{to}\right|_{t}, \left.\vec{Y}\right|_{t}, \Delta\,\underline{\epsilon}^{to}, \Delta\,t\right) \underbrace{\Longrightarrow}_{behaviour} \left(\left.\underline{\sigma}\right|_{t+\Delta\,t}, \left.\vec{Y}\right|_{t+\Delta\,t}, \dfrac{\partial\Delta\,\underline{\sigma}}{\partial\Delta\,\underline{\epsilon}^{to}}\right)$$

- Given a strain increment $\Delta\,\underline{\epsilon}^{to}$ over a time step $\Delta\,t$, the mechanical behaviour must compute:
  - The value of the stress $\left.\underline{\sigma}\right|_{t+\Delta\,t}$ at the end of the time step.
  - The value of internal state variables, noted $\left.\vec{Y}\right|_{t+\Delta\,t}$ at the end of the time step.
  - The consistent tangent operator: $\dfrac{\partial\Delta\,\underline{\sigma}}{\partial\Delta\,\underline{\epsilon}^{to}}$
- For specific cases, the mechanical behaviour shall also provide:
  - a prediction operator
  - the elastic operator (Abaqus-Explicit, Europlexus)
  - estimation of the stored and dissipated energies (Abaqus-Explicit)

Other functions of the mechanical behaviour

- Provide a estimation of the next time step for time step automatic adaptation
- Check bounds:
  - Physical bounds
  - Standard bounds
- Clear error messages
- Parameters
  - It is all about Quality Assurance!
  - Parametric studies, identification, etc. . .
- Generate ‘MTest‘ files on integration failures
- Generated example of usage:
  - Generation of MODELISER/MATERIAU instructions (Cast3M)
  - Input file for Abaqus, Ansys
- Provide information for dynamic resolution of inputs (MTest/Aster/Europlexus):
  - Numbers Types (scalar, tensors, symmetric tensors)
  - Entry names /Glossary names. . .

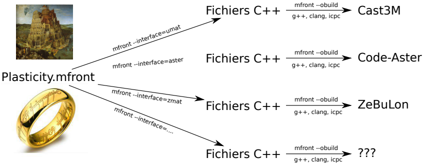
Sommaire

Mechanical behaviours

An overview of `MFront`

Conclusions and perspectives

MFront goals



- `MFront` is a code generation tool dedicated to material knowledge (material properties, mechanical behaviours, point-wise models):
  - Support for small and finite strain behaviours, cohesive zone models, **generalised behaviours** (non local and or multiphysics).
- Main goals:
  - Numerical efficiency (see various benchmarks on the website).
  - Portability (`Cast3M`, `Cyrano`, `code_aster`, `Europlexus`, `TMFTT`, `AMITEX_FFTP`, `Abaqus`, `CalculiX`, `MTest`).
  - **Ease of use**: *Longum iter est per praecepta, breve et efficax per exempla* (It's a long way by the rules, but short and efficient with examples).

An example of the `StandardElasticityVicoPlastici`

```
@DSL Implicit;
@Behaviour MohrCoulomAbboSloan3;

@Epsilon 1.e-14;
@Theta 1;

@Brick StandardElastoViscoPlasticity{
  stress_potential : "Hooke" {
    young_modulus : 150.e3,
    poisson_ratio : 0.3
  },
  inelastic_flow : "Plastic" {
    criterion : "MohrCoulomb" {
      c : 3.e1,        // cohesion
      phi : 0.523598775598299,   // friction angle or dilatancy angle
      lodeT : 0.506145483078356, // transition angle as defined by Abbo and Sloan
      a : 1e1          // tension cuff-off parameter
    },
    flow_criterion : "MohrCoulomb" {
      c : 3.e1,        // cohesion
      phi : 0.174532925199433,   // friction angle or dilatancy angle
      lodeT : 0.506145483078356, // transition angle as defined by Abbo and Sloan
      a : 3e1          // tension cuff-off parameter
    },
    isotropic_hardening : "Linear" {R0 : "0"}
  }
};
```

- The `StandardElasticityVicoPlasticity` brick allows to implement complex visco-plastic behaviours with a declarative syntax using predefined components.

An simple example with the Implicit DSL and the Stan

```
@DSL Implicit;
@Behaviour Norton;
@Brick StandardElasticity;

@MaterialProperty stress E;
E.setGlossaryName("YoungModulus");
@MaterialProperty real ν, A, nn;
ν.setGlossaryName("PoissonRatio");
A.setEntryName("NortonCoefficient");
nn.setEntryName("NortonExponent");

@StateVariable real    p;
p.setGlossaryName("EquivalentViscoplasticStrain");

@Integrator{
  constexpr const auto Mᵉ = Stensor4::M();
  const auto μ = computeMu(E, ν);
  const auto σᵉ = sigmaeq(σ);
  const auto iσᵉ = 1 / (max(σᵉ, real(1.e-12) ⋅ E));
  const auto vᵖ = A ⋅ pow(σᵉ, nn);
  const auto ∂vᵖ/∂σᵉ = nn ⋅ vᵖ ⋅ iσᵉ;
  const auto n = 3 ⋅ deviator(σ) ⋅ (iσᵉ / 2);
  // Implicit system
  fεᵉˡ += Δp ⋅ n;
  fp -= vᵖ ⋅ Δt;
  // jacobian
  ∂fεᵉˡ/∂Δεᵉˡ += 2 ⋅ μ ⋅ θ ⋅ dp ⋅ iσᵉ ⋅ (Mᵉ - (n ⊗ n));
  ∂fεᵉˡ/∂Δp = n;
  ∂fp/∂Δεᵉˡ = -2 ⋅ μ ⋅ θ ⋅ ∂vᵖ/∂σᵉ ⋅ Δt ⋅ n;
} // end of @Integrator
```

- Implicit integration.
- Implicit system:

$$\begin{cases} f_{\underline{\epsilon}^{el}} = \Delta\,\underline{\epsilon}^{el} - \Delta\,\underline{\epsilon}^{to} + \Delta\,p\,\underline{\mathbf{n}} \\ f_p = \Delta\,p - A\,\sigma_{eq}^{n} \end{cases}$$

- Jacobian:

$$\begin{cases} \dfrac{\partial f_{\underline{\epsilon}^{el}}}{\partial \Delta\,\underline{\epsilon}^{el}} = \underline{\underline{\mathbf{I}}} + \dfrac{2\,\mu\,\theta\,\Delta\,p}{\sigma_{eq}}\left(\underline{\underline{\mathbf{M}}} - \underline{\mathbf{n}} \otimes \underline{\mathbf{n}}\right) \\ \dfrac{\partial f_{\underline{\epsilon}^{el}}}{\partial \Delta\,p} = \underline{\mathbf{n}} \\ \dfrac{\partial f_p}{\partial \Delta\,\underline{\epsilon}^{el}} = -2\,\mu\,\theta\,A\,n\,\sigma_{eq}^{n-1}\,\Delta\,t\,\underline{\mathbf{n}} \end{cases}$$

- **All programming and numerical details are hidden (by default).**

Support for quantities in MFront

```
@Parameter strainrate A = 8.e−67;
@Parameter real E = 8.2;
@Parameter stress K = 1;
@Parameter stress R0 = 20e6;
@Parameter stress Rinf = 40e6;
@Parameter real bvp = 10;

@Integrator {
  ...
  const auto seq = sigmaeq(sig);                        // seq has the unit of a stress
  const auto iseq = 1 / max(seps, seq);                 // iseq has the unit of the inverse of a stress
  const auto n = 3 * deviator(sig) * (iseq / 2);        // normal has no unit
  const auto exp_bvp = exp(−bvp * (p + theta * dp));  // exp_bvp has no unit
  Rvp = R0 + (Rinf − R0) * (1 − exp_bvp);               // Rvp has the unit of a stress
  if (seq > Rvp) {
    const auto vp = A * pow((seq − Rvp) / K, E);        // vp has the unit of a strainrate
    fp −= vp * dt;                                      // fp has the unit a a strain
    // fp −= pow((seq − Rvp) / K, E) * dt;              // This would not compile !
    // fp −= A * pow(seq − Rvp, E) * dt;                // This would not compile !
  }
  feel += dp * n;
}
```

- The `UseQt` keyword actives the use of quantities.
- Dedicated documentation of the declaration of variables in MFront
- Quantities are supported in DSLs associated with material properties and behaviours.

An example of the `StandardElasticityVicoPlastici`

```
@DSL Implicit;
@Behaviour MohrCoulomAbboSloan3;

@Epsilon 1.e-14;
@Theta 1;

@Brick StandardElastoViscoPlasticity{
  stress_potential : "Hooke" {
    young_modulus : 150.e3,
    poisson_ratio : 0.3
  },
  inelastic_flow : "Plastic" {
    criterion : "MohrCoulomb" {
      c : 3.e1,          // cohesion
      phi : 0.523598775598299,   // friction angle or dilatancy angle
      lodeT : 0.506145483078356, // transition angle as defined by Abbo and Sloan
      a : 1e1           // tension cuff-off parameter
    },
    flow_criterion : "MohrCoulomb" {
      c : 3.e1,          // cohesion
      phi : 0.174532925199433,   // friction angle or dilatancy angle
      lodeT : 0.506145483078356, // transition angle as defined by Abbo and Sloan
      a : 3e1           // tension cuff-off parameter
    },
    isotropic_hardening : "Linear" {R0 : "0"}
  }
};
```

- The `StandardElasticityVicoPlasticity` brick allows a declarative syntax using predefined components.

An simple example with the Implicit DSL and the Stan

```
@DSL Implicit;
@Behaviour Norton;
@Brick StandardElasticity;

@MaterialProperty stress E;
E.setGlossaryName("YoungModulus");
@MaterialProperty real ν, A, nn;
ν.setGlossaryName("PoissonRatio");
A.setEntryName("NortonCoefficient");
nn.setEntryName("NortonExponent");

@StateVariable real    p;
p.setGlossaryName("EquivalentViscoplasticStrain");

@Integrator{
  constexpr const auto Mᵉ = Stensor4::M();
  const auto μ = computeMu(E, ν);
  const auto σᵉ = sigmaeq(σ);
  const auto iσᵉ = 1 / (max(σᵉ, real(1.e-12) ⋅ E));
  const auto vᵖ = A ⋅ pow(σᵉ, nn);
  const auto ∂vᵖ/∂σᵉ = nn ⋅ vᵖ ⋅ iσᵉ;
  const auto n = 3 ⋅ deviator(σ) ⋅ (iσᵉ / 2);
  // Implicit system
  fεᵉˡ += Δp ⋅ n;
  fp -= vᵖ ⋅ Δt;
  // jacobian
  ∂fεᵉˡ/∂Δεᵉˡ += 2 ⋅ μ ⋅ θ ⋅ dp ⋅ iσᵉ ⋅ (Mᵉ - (n ⊗ n));
  ∂fεᵉˡ/∂Δp = n;
  ∂fp/∂Δεᵉˡ = -2 ⋅ μ ⋅ θ ⋅ ∂vᵖ/∂σᵉ ⋅ Δt ⋅ n;
} // end of @Integrator
```

- Implicit integration.
- Implicit system:

$$\begin{cases} f_{\underline{\epsilon}^{el}} = \Delta\,\underline{\epsilon}^{el} - \Delta\,\underline{\epsilon}^{to} + \Delta\,p\,\underline{\mathbf{n}} \\ f_p = \Delta\,p - A\,\sigma_{eq}^{n} \end{cases}$$

- Jacobian:

$$\begin{cases} \dfrac{\partial f_{\underline{\epsilon}^{el}}}{\partial \Delta\,\underline{\epsilon}^{el}} = \underline{\underline{\mathbf{I}}} + \dfrac{2\,\mu\,\theta\,\Delta\,p}{\sigma_{eq}}\left(\underline{\underline{\mathbf{M}}} - \underline{\mathbf{n}} \otimes \underline{\mathbf{n}}\right) \\ \dfrac{\partial f_{\underline{\epsilon}^{el}}}{\partial \Delta\,p} = \underline{\mathbf{n}} \\ \dfrac{\partial f_p}{\partial \Delta\,\underline{\epsilon}^{el}} = -2\,\mu\,\theta\,A\,n\,\sigma_{eq}^{n-1}\,\Delta\,t\,\underline{\mathbf{n}} \end{cases}$$

- **All programming and numerical details are hidden (by default).**

Generic behaviours

- Standard small strain mechanical behaviours:

$$\Delta\varepsilon, \boldsymbol{\sigma}_n, \boldsymbol{Y}_n \rightarrow \boxed{\text{MFront}} \rightarrow \boldsymbol{\sigma}_{n+1}, \boldsymbol{Y}_{n+1}, \frac{\partial\boldsymbol{\sigma}}{\partial\Delta\varepsilon}$$

- Generalized behaviours:

$$(\Delta\boldsymbol{g}^1, \ldots \Delta\boldsymbol{g}^p)_n, (\boldsymbol{\sigma}^1, \ldots, \boldsymbol{\sigma}^p)_n, \boldsymbol{Y}_n \rightarrow \boxed{\text{MFront}} \rightarrow (\boldsymbol{\sigma}^1, \ldots, \boldsymbol{\sigma}^p)_{n+1}, \boldsymbol{Y}_{n+1}, \frac{\partial\boldsymbol{\sigma}^j}{\partial\Delta\boldsymbol{g}^k}$$

- $\boldsymbol{g}^j$ are **gradients** (temp. gradient, strain, etc.) depending on the FE unknowns $\boldsymbol{u}$
- $\boldsymbol{\sigma}^j$ are associated **fluxes** or **thermodynamic forces** (heat flux, stress, etc.)
- $\dfrac{\partial\boldsymbol{\sigma}^j}{\partial\Delta\boldsymbol{g}^k}$ are so-called **tangent blocks**
- Work of internal forces density: $\delta w_{\text{int}} = \displaystyle\sum_{i=1}^{p} \boldsymbol{\sigma}^i \cdot \delta\boldsymbol{g}^i$