

Modelling non-linear constitutive material laws in FEniCS with MFront

Jeremy Bleier

*Laboratoire Navier, UMR 8205
Ecole des Ponts ParisTech-IFSTTAR-CNRS*



IFSTTAR



MFront User Day, October, 14th 2019

Outline

1 FEniCS overview

2 MFront integration

<http://fenicsproject.org/>

collection of free, open source, software components for automated solution of differential equations



- initiated by Chicago and Chalmers University in 2003
- contributions from people from Simula Research Lab, Cambridge, KTH...
- large community and active development

Features

- automated solution of variational formulation (same spirit as FreeFem++, deal.ii, etc.)
- extensive library of finite elements
- designed for parallel computation (high-performance linear algebra through PETSc backends)
- simple Python interface and concise high-level language

A collection of interoperable components

- **FIAT** (*Finite element Automatic Tabulator*) generates finite elements of arbitrary order on lines, triangles and tetrahedra.
- **UFL** (*Unified Form Language*), for specifying FE discretizations of PDE in terms of FE variational forms
- **UFC** (*Unified Form Compiler*), a C++ interface consisting of low-level functions for evaluating and assembling FE variational forms
- **FFC** (*FEniCS Form Compiler*), a JIT compiler for UFL \rightarrow UFC
- **DOLFIN**, a C++/Python library providing data structures and algorithms for FE meshes, automated FE assembly, and numerical linear algebra
- communication with PETSc, Eigen, ParMETIS, SCOTCH, MPI, OpenMP and XDMF output

Cantilever beam example

```
from dolfin import *

# Elasticity parameters
lmbda = Constant(90.)
mu = Constant(100.)

# Define epsilon and sigma operator
def eps(v):
    return sym(grad(v))
def sigma(v):
    dim = v.geometric_dimension()
    return 2.0*mu*eps(v) + lmbda*tr(eps(v))*Identity(dim)

# Create mesh [0;10]x[0;1]
mesh = RectangleMesh(Point(0.,0.),Point(10.,1.),100, 10)

# Loading due to self weight
f = Constant((0.,-1.))
```

Cantilever beam example

We now define the variational problem for elasticity : Find $\underline{u} \in V_{Trial}$ s.t.

$$a(\underline{u}, \underline{v}) = \int_{\Omega} \underline{\underline{\sigma}}(\underline{u}) : \underline{\underline{\epsilon}}(\underline{v}) d\Omega = \int_{\Omega} \underline{f} \cdot \underline{v} d\Omega = L(\underline{v}) \quad \forall \underline{v} \in V_{Test}$$

here $V_{Test} = V_{Trial} = V$ (Galerkin method)

```
# Define function space
V = VectorFunctionSpace(mesh, 'Lagrange', degree=1)

# Define variational problem
u = TrialFunction(V)
v = TestFunction(V)
a = inner(sigma(u), eps(v))*dx
L = dot(f, v)*dx
```

Cantilever beam example

Defining boundary conditions :

```
def left(x, on_boundary):  
    return near(x[0],0.) and on_boundary  
  
bc = DirichletBC(V, Constant((0.,0.)), left)
```

Compute the solution :

```
u = Function(V)  
solve(a == L, u, bc)
```

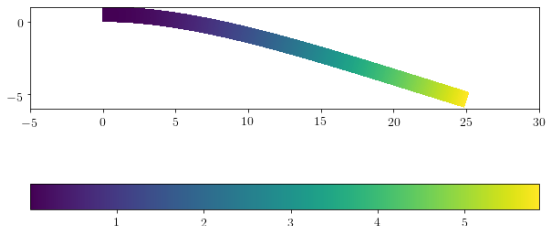
We can specify the **linear solver type**:

- direct (LU, MUMPS, ...)
- iterative (CG, GMRES, ...)
- with preconditioners (ILU, Algebraic Multigrid, ...)

Cantilever beam example

Plotting routines

```
plot(u, mode='displacement')
```



But also output to Paraview

```
output_file = XDMFFile("output.xdmf")  
output_file.write(u, 0.)
```


Multi-field variational problem

One of the **big advantages of FEniCS**

E.g. Timoshenko beam variational formulation : Find $(w, \Phi) \in V_{Trial}$:

$$\int_{\Omega} EI(\nabla \Phi) \cdot (\nabla \hat{\Phi}) + \kappa \mu A (\nabla w - \Phi) \cdot (\nabla \hat{w} - \hat{\Phi}) d\Omega = \int_{\Omega} f w d\Omega \quad \forall (\hat{w}, \hat{\Phi}) \in V_{Test}$$

```
U = FiniteElement("CG", mesh.ufl_cell(), 2)
T = FiniteElement("CG", mesh.ufl_cell(), 1)
V = FunctionSpace(mesh, U*T)

u = TrialFunction(V)
v = TestFunction(V)

(w, phi) = split(u)
(w_, phi_) = split(v)
```

with $V_{Test} = V_{Trial} = \mathcal{P}^2 \times \mathcal{P}^1$

Multi-field variational problem

```

# Generalized strain measures
def chi(Phi):
    return grad(Phi)
def gamma(w,Phi):
    return grad(w)-Phi

# Constitutive laws for bending and shear
def M(Phi):
    return E*I*chi(Phi)
def Q(w,Phi):
    return kappa*mu*A*gamma(w,Phi)

# Final variational forms
a = ( inner(M(phi),chi(phi_)) +
      inner(Q(w,phi),gamma(w_,phi_)) ) * dx
L = f*w_*dx

# Compute the solution
u = Function(V)
solve(a == L, u, bc)

```

Other example: Stokes equation

Mixed $u - p$ formulation with **Taylor-Hood element** ($\mathcal{P}^2/\mathcal{P}^1$)

```
# Define function space
P2 = VectorElement('P', tetrahedron, 2)
P1 = FiniteElement('P', tetrahedron, 1)
TH = P2 * P1
W = FunctionSpace(mesh, TH)

# Define variational problem
(u, p) = TrialFunctions(W)
(v, q) = TestFunctions(W)
a = inner(grad(u), grad(v))*dx - p*div(v)*dx + div(u)*q*dx
L = dot(f, v)*dx
```

MINI element ($\mathcal{P}^1 + \mathcal{B}$)/ \mathcal{P}^1

```
# Define function space
Pv1 = VectorElement('P', tetrahedron, 2)
B = VectorElement('Bubble', tetrahedron, 3)
W = FunctionSpace(mesh, (Pv1+B)*P1)
```

Nonlinear problems

Hyperelastic model : elastic potential + automatic differentiation

```
F = Identity(dim) + grad(u)    # Deformation gradient
C = F.T*F
# Invariants of deformation tensors
Ic, J = tr(C), det(F)
# potential (compressible neo-Hookean model)
psi = (mu/2)*(Ic - 3) - mu*ln(J) + (lmbda/2)*(ln(J))**2

# Total potential energy
Pi = psi*dx - dot(f, u)*dx

# Compute first variation of Pi (directional derivative)
F = derivative(Pi, u, v)

# Compute Jacobian of F
J = derivative(F, u, du)
```

Nonlinear problems

```
# Solve variational problem  
solve(F == 0, u, bc, J=J)
```

implicitly use a non linear solver, but again we can specify more precisely which solver and parameters to use

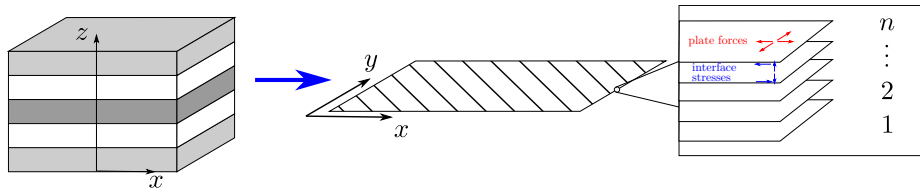
- Newton method
- PETSc SNES solver : line search, trust region
- PETSC TAO solver : bound-constrained minimization

we can choose which linear solver and preconditioner to use for the iterative process

also possible to **formulate yourself** a Newton method at the PDE level

Multilayered plate models

Multilayered plate models for accurate representation of edge effects, interface stress, delamination, etc...



3D laminate is represented by a family of n "plates" in interaction: generalized forces, plate kinematics and associated constitutive behaviors depend on the construction of the model

Many variants have been developed: [Ehrlacher, 1993; Caron, 1997; Chabot, 1997; Diaz Diaz, 2001; Baroud, 2016]

Recent developments

A **statically compatible** stress-based model [Baroud et al., 2016]

- offers a **purely statical** construction from 3D stresses (**no hypothesis** of the kinematics along z)
- enables to satisfy exactly stress-free boundary conditions
- **convergence results** wrt 3D model when refining the number of layers

Basic assumptions

- in-plane $\sigma_{\alpha\beta}$ stresses are **linear** $\longrightarrow N_{\alpha\beta}^i, M_{\alpha\beta}^i$
- out-of-plane shear stresses are **quadratic** $\longrightarrow \underline{Q}^i, \underline{\tau}^{i-1,i}, \underline{\tau}^{i,i+1}$
- out-of-plane normal stress σ_{33} is **cubic** $\longrightarrow \nu^{i-1,i}, \nu^{i,i+1}, \pi^{i-1,i}, \pi^{i,i+1}$

with continuity of

- interlaminar shear stress $\tau_{\alpha}^{i,i+1} = \sigma_{\alpha 3}(x, y, h_i^+) = \sigma_{\alpha 3}(x, y, h_{i+1}^-)$
- interlaminar normal stress $\nu^{i,i+1} = \sigma_{33}(x, y, h_i^+) = \sigma_{33}(x, y, h_{i+1}^-)$

Recent developments

Equilibrium equations:

$$\operatorname{div} \underline{\underline{N}}^i + \underline{\tau}^{i,i+1} - \underline{\tau}^{i-1,i} = 0$$

$$\operatorname{div} \underline{Q}^i + \underline{\nu}^{i,i+1} - \underline{\nu}^{i-1,i} = 0$$

$$\operatorname{div} \underline{\underline{M}}^i - \underline{Q}^i + \frac{e^i}{2} (\underline{\tau}^{i,i+1} + \underline{\tau}^{i-1,i}) = 0$$

$$\operatorname{div} \underline{\tau}^{i,i+1} - \pi^{i,i+1} = 0$$

Kinematics:

$$2n + n + 2n + n - 1 = 6n - 1 \text{ dofs/node}$$

- in-plane displacement per layer \underline{U}^i
- out-of-plane displacement per layer W^i
- rotations $\underline{\Phi}^i$ per layer
- "interface relative displacement" $\underline{V}^{i,i+1}$

Recent developments

Generalized strain measures:

$$\underline{\underline{N}}^i \longleftrightarrow \underline{\underline{\varepsilon}}^i = \underline{\underline{\nabla^s U}}^i$$

$$\underline{\underline{M}}^i \longleftrightarrow \underline{\underline{\chi}}^i = \underline{\underline{\nabla^s \Phi}}^i$$

$$\underline{Q}^i \longleftrightarrow \underline{\gamma}^i = \underline{\Phi}^i + \underline{\nabla W}^i$$

$$\underline{\tau}^{i,i+1} \longleftrightarrow \underline{D}^{i,i+1} = \underline{U}^{i+1} - \underline{U}^i - \frac{1}{2}(e^i \underline{\Phi}^i + e^{i+1} \underline{\Phi}^{i+1}) + \underline{\nabla V}^{i,i+1}$$

$$\nu^{i,i+1} \longleftrightarrow D_\nu^{i,i+1} = U_3^{i+1} - U_3^i$$

$$\pi^{i,i+1} \longleftrightarrow \lambda^{i,i+1} = V^{i,i+1}$$

Elastic constitutive behaviour

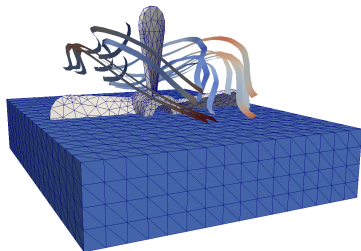
$$\begin{Bmatrix} \mathbf{E}_1 \\ \mathbf{E}_2 \end{Bmatrix} = \begin{bmatrix} \mathbf{S}_1 & 0 \\ 0 & \mathbf{S}_2 \end{bmatrix} \begin{Bmatrix} \boldsymbol{\Sigma}_1 \\ \boldsymbol{\Sigma}_2 \end{Bmatrix}$$

where $\boldsymbol{\Sigma}_1 = \{ \underline{\underline{N}}^1 \quad \underline{\underline{M}}^1 \quad \nu^{1,2} \quad \pi^{1,2} \quad \underline{\underline{N}}^2 \quad \dots \quad \pi^{n-1,n} \}^T$
 $\boldsymbol{\Sigma}_2 = \{ \underline{Q}^1 \quad \underline{\tau}^{1,2} \quad \underline{Q}^2 \quad \dots \quad \underline{\tau}^{n-1,n} \}^T$

Other features

Available features :

- adaptive mesh refinement
- Discontinuous Galerkin method
- multimesh



Partly missing features

- cohesive elements in 2D/3D solids: possible with DG methods but need deeper look
- partial support for quadrilaterals/hexahedra, current development of isoparametric (curved) elements
- partial Lagrange multipliers, coupling different models (possible with multiphenics library)

What FEniCS has not been designed for

Complex material constitutive laws:

- FEniCS has been written by applied mathematics/computational fluid mechanics researchers
- not with the mindset of complex material behaviour, do not have direct access at what happens at the Gauss point level but at the continuous form level
- **until now:** a small attempt in the FEniCS Solid Mechanics project

Contact: simple contact penalty formulations or Augmented Lagrangian approaches are possible to implement but no such thing as contact detection/surface to surface or mortar approaches, etc...

Low-level FE tweaks like B-bar method, assumed strain approaches, etc... Possible but needs to go deep in the code (possible amelioration quite soon with `dolfin-x` version)

Shells: Solid shell elements not implemented, non-manifold surfaces need proper testing, some developments in the `fenics-shells` library

von Mises plasticity with linear hardening

Define a Newton-Raphson variational form:

```
K_Newton = inner(eps(v), dot(Ct, eps(u_)))*dx
residual = -inner(eps(u_), sig)*dx + F_ext(u_)
```

where C_t is the tangent stiffness

sig is a FEniCS Function living in a **Quadrature** FunctionSpace

similarly for \mathbb{C}^{tan}

Explicit return mapping as UFL expressions

sig is the prediction of $\underline{\underline{\sigma}}_n$, depends on strain increment $\Delta \underline{\underline{\epsilon}}$ and previous stress $\underline{\underline{\sigma}}_{n-1}$ and cumulated plastic strain p_{n-1}

Analytical expression in UFL:

```
sig_elas = sig_old + sigma(deps) # elastic predictor
s = dev(sig_elas)                # deviatoric part predictor
sig_eq = sqrt(3/2.*inner(s, s))  # equivalent von Mises norm
f_elas = sig_eq - sig0 - H*p_old # plasticity criterion
dp = ppos(f_elas)/(3*mu+H)       # plastic strain increment
sig = sig_elas - 3*mu*dp/sig_eq*s # corrected stress state
```

von Mises plasticity with linear hardening

Analytical expression also for \mathbb{C}^{tan}

Expressions are non-linear expressions of u

They are evaluated at Gauss points by projecting on the **Quadrature** function space

Limitations

the return mapping is **analytical** in this simple case

it can therefore be expressed explicitly with UFL expressions

Unfortunately, FEniCS does not provide a mechanism for inner non-linear procedures at the Gauss point level

Outline

1 FEniCS overview

2 MFront integration

Coupling with MFront through MGIS

we keep the same Newton-Raphson method with arrays of current stresses, tangent matrix and internal state variables at each Gauss point

MGIS Python interface for defining the behaviour

```
# Defining the modelling hypothesis
h = mgis_bv.Hypothesis.PlaneStrain
# Loading the behaviour
b = mgis_bv.load('src/libBehaviour.so', '
    IsotropicLinearHardeningPlasticity', h)
# Setting the material data manager
m = mgis_bv.MaterialDataManager(b, ngauss)
for s in [m.s0, m.s1]:
    mgis_bv.setMaterialProperty(s, "YoungModulus", 70e3)
    mgis_bv.setMaterialProperty(s, "PoissonRatio", 0.3)
    mgis_bv.setMaterialProperty(s, "HardeningSlope", 707.1)
    mgis_bv.setMaterialProperty(s, "YieldStrength", 250.)
    mgis_bv.setExternalStateVariable(s, "Temperature",
    293.15)
```

Coupling with MFront

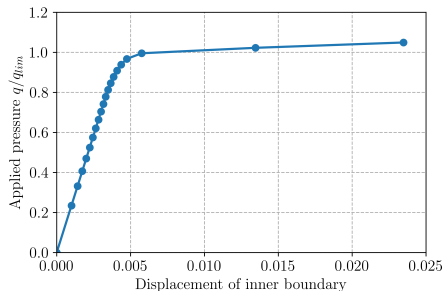
Inside the NR loop:

- project (evaluate) ϵ at Gauss points
- feed Gauss point values to MGIS
- integrate behaviour
- get stress and tangent operator back and update sig and Ct with new values

```
# copy the strain values to `MGIS`
m.s1.gradients[:, :] = Eps.vector().get_local().reshape((m.n
    , s_dim))
# integrate the behaviour
it = mgis_bv.IntegrationType.
    IntegrationWithConsistentTangentOperator
mgis_bv.integrate(m, it, 0, 0, m.n);
# getting the stress and consistent tangent operator back to
    the FEniCS world.
sig.vector().set_local(m.s1.thermodynamic_forces.flatten())
Ct.vector().set_local(m.K.flatten())
```


Results

Cylinder expansion subject to internal pressure



Same residuals, computing time:

6.8s for pure FEniCS implementation

5.9s for FEniCS-MFront implementation

Limitations

Still some issues right now:

- **multi-materials** : internal variables should be defined on their corresponding subdomain, currently they must live on the whole domain
- **memory and efficiency** : we need to store the values of stresses and tangent operator components at all Gauss points, then use these values when performing the assembly
usually, behaviour integration is performed during the assembly (FEniCS offers no user intervention during assembly)
- not implemented for **parallel computations**

In the future

- FEniCS currently undergoing consequent redesign
<https://github.com/FEniCS/dolfinx>
- should allow definition of functions on subdomains
- should allow user-defined coefficients in forms

Extension to large strains

Quite simple using for instance $\underline{\underline{F}}$ and $\underline{\underline{P}}$ (PK1 stress):

Residual is:

$$R(\underline{u}) = W_{\text{ext}}(\underline{u}) - \int_{\Omega} \underline{\underline{P}} : \delta \underline{\underline{F}}(\underline{u}) dx = W_{\text{ext}}(\underline{u}) - \int_{\Omega} \underline{\underline{P}} : \nabla(\delta \underline{u}) dx$$

```
residual = Wext-inner(pk1, grad(u_))*dx
```

Consistent tangent bilinear form is:

$$a(\underline{u}, \underline{v}) = \int_{\Omega} \nabla \underline{u} : \frac{\partial \underline{\underline{P}}}{\partial \underline{\underline{F}}} : \nabla \underline{v} dx$$

```
a_Newton = inner(grad(u_), dot(Ct, grad(v)))*dx
```

where Ct is provided by MGIS in the right "DPK1_DF" format using

```
mgis_bv.convertFiniteStrainTangentOperator(Ct, m, mgis_bv.  
FiniteStrainStress.DPK1_DF)
```

Example

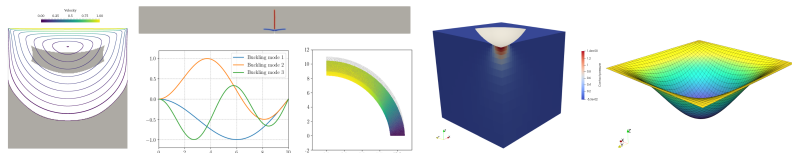
3D traction of elastic Signorini law

Example

Axisymmetric traction of a notched plate: large strain isotropic plasticity using logarithmic deformation framework

A collection of computational mechanics examples

Available at: <https://comet-fenics.readthedocs.io/en/latest/>



- isotropic/orthotropic elasticity
- axisymmetric structures
- uncoupled and coupled thermoelasticity
- transient elastodynamics
- periodic homogenization
- viscoelasticity
- elastoplasticity
- beam buckling
- plates with reduced integration/Discontinuous Galerkin formulation