Supplement of Saf. Nucl. Waste Disposal, 1, 185–186, 2021
https://doi.org/10.5194/sand-1-185-2021-supplement

Supplement of

Open Source Software Library for Thermo-Hydro-Mechanical Coupled Processes in Python

Carsten Rücker

Correspondence to: Carsten Rücker (carsten.ruecker@bfe.bund.de)

Open Source Software Library for THM coupled Processes in Python

Carsten Rücker

Federal Office for the Safety of Nuclear Waste Management (BASE), Berlin, Germany

safe ND

Federal Office
for the Safety of
Nuclear Waste Management

Motivation and Aims

Extensive knowledge of the scientific foundations of thermo-hydro-mechanical (THM) coupled processes, as well as the reliable handling of numerical methods for the simulation of such processes, are mandatory for the evaluation of preliminary safety investigations during the site selection process for the storage of high-level radioactive waste.

Motivations for the development of an Open Source Software Library

- Targeted development of expertise within BASE regarding numerical modeling.
- Diversify the testing capabilities regarding the preliminary safety investigations by means of an own, independent modeling software.
- Document basic THM scenarios for internal or, if necessary, public technical training.
- Ensure transparency and, in principle, might allow for providing the public appropriately quality assured and documented simulation tools.

Approach and Aims

- Building a new library based on pyGIMLi Pre- and Postprocessing framework (Rücker et al., 2017).
- Create Finite Element (FE) reference implementation in the Python scripting language for maximal transparency.
- Find an easy to use interface to solve the weak formulation for FE with expressions in a symbolic manner and allowing necessary flexibility.
- Define an interface to potential allow for the integration of alternative third party high performance libraries.
- Creating a library of jupyter notebooks of well documented test cases and benchmarks.

✓ free, open-source, platform independent

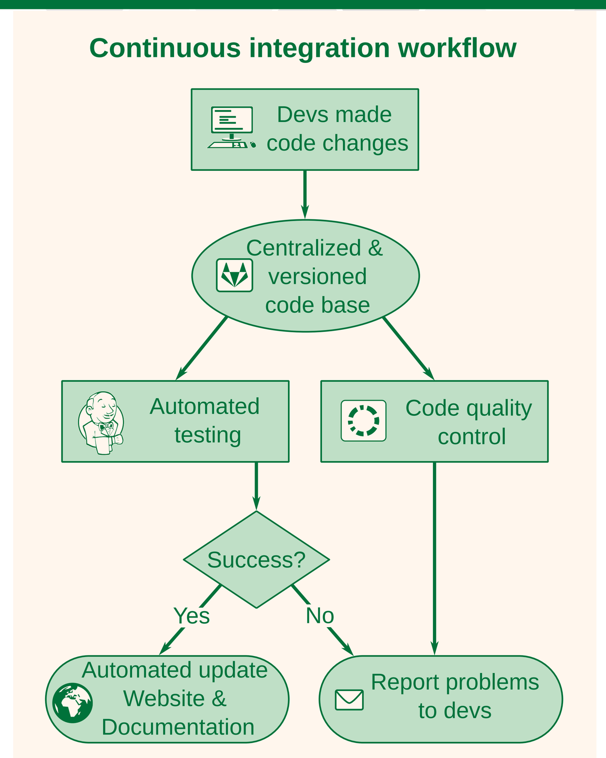
matplotlib · jupyter · python powered · NumPy · SciPy · pyGIMLi Geophysical Inversion & Modelling Library

Current state: Building technical proof of concept for the expression based interface. See Examples →

Quality assurance

It can be argued that good practice for codes should contain e.g., Transparency, Reliability, Maintainability, and Usability, which can be achieved with modern development tool chains and open-source software.

- Continuous integration
- Centralized version control with GIT
- Automated quality control after each code change
- Automated testing: Unittests, examples, benchmarks
- Automated generation of documentation



References

Elder, J.W., 1967. Transient convection in a porous medium, *J. Fluid Mech.*, 27, 609–623, doi:10.1017/s0022112067000576.

Popov, P., Qin, G., Bi, L., Efendiev, Y., Ewing, R.E., Li, J., 2009. Multiphysics and Multiscale Methods for Modeling Fluid Flow Through Naturally Fractured Carbonate Karst Reservoirs, *SPE Res Eval & Eng*, 12, 218–231, doi:10.2118/105378-pa.

Rücker, C., Günther, T., Wagner, F.M., 2017. pyGIMLi: An open-source library for modelling and inversion in geophysics, *Computers and Geosciences*, 109, 106–123, doi:10.1016/j.cageo.2017.07.011.

T Thermal conduction

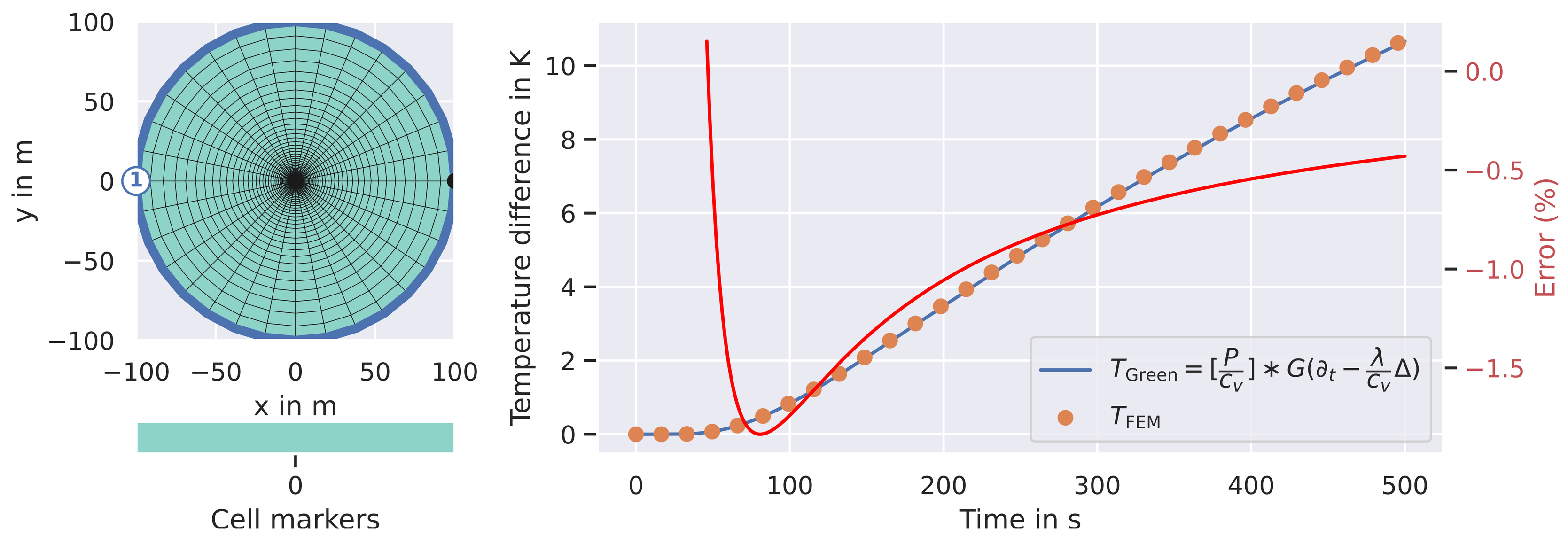
- Find the temperature field T for heat equation with thermal conductivity λ and volumetric heat capacity c_v and energy source P:

$$c_v\frac{\partial T}{\partial t} - \lambda\Delta T = P$$

- Semi-analytic 1D reference solution by convolving source term with Green's function for diffusion
- Finite Element problem solved in 2D on regular polar grid with logarithmic increased spacing
- Singular P at origin, fixed Temperature $T = 0$ on all boundaries (Dirichlet boundary condition)

Relevant part of the python script

```
# Sought scalar field for the mesh with linear basis functions
N = ScalarSpace(mesh, p=1)
# Assemble system matrix regarding weak formulation with Galerkin method
S = (grad(N) * lam * grad(N)).assemble()
# Assemble mass matrix needed for time integration
M = (N * cv * N).assemble()
# Create source term
rhs = np.zeros(N.dof)
# Set singular point source at origin
rhs[mesh.findNearestNode([0.0, 0.0])] = P
# Define Dirichlet boundary condition
dirichlet = DirichletManager({N:{'Dirichlet':{'*':0.0}}})
# Solve for Temperature at given times with Crank-Nicolson scheme
T = pg.solver.crankNicolson(times=t, S=S, I=M, f=rhs, dirichlet=dirichlet, theta=0.6)
# Interpolate probe values at a distance of 10 m
T_probe = T([10.0, .0])
```



H Stoke-Brinkman equation

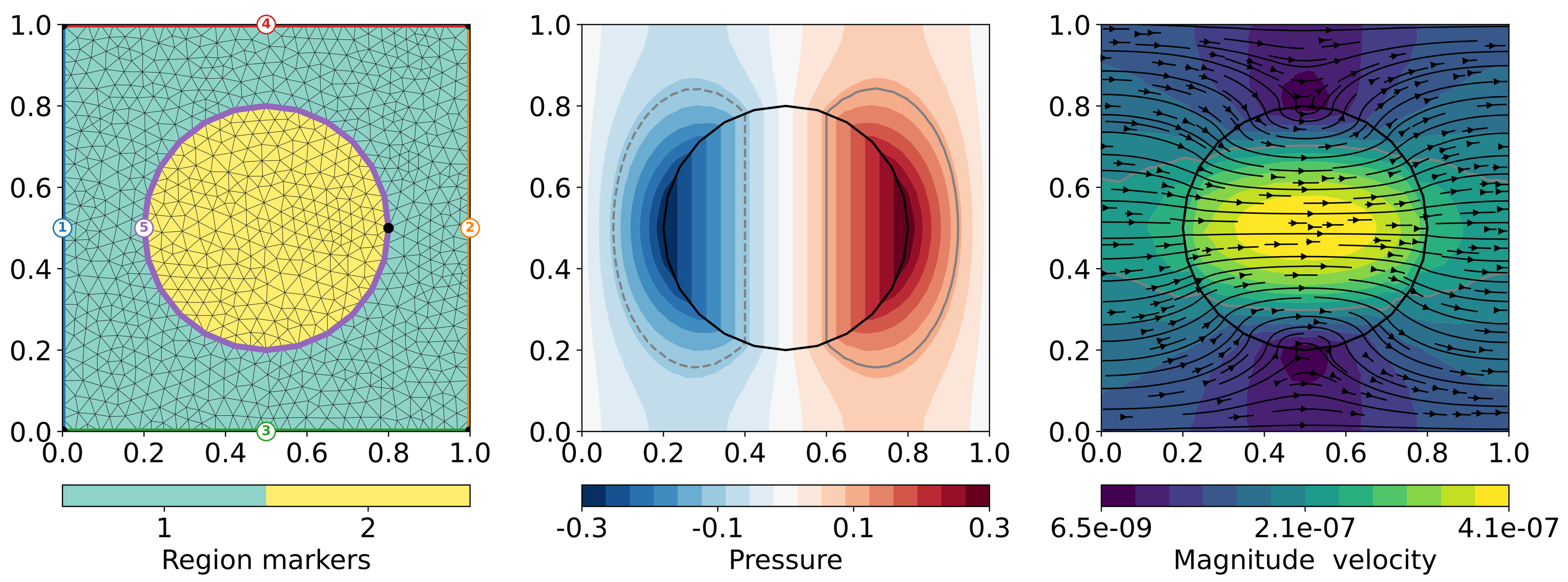
- Fluid flow in fractured media with seamless transition between porous media and free-flow region after (Popov et al., 2009)
- **K** – permeability tensor, μ – physical viscosity of the fluid, μ^* – effective viscosity

$$\mu\mathbf{K}^{-1}\mathbf{v} + \nabla p - \mu^*\Delta\mathbf{v} = \mathbf{f} \quad \text{and} \quad \nabla\cdot\mathbf{v} = 0$$

- Region 1: porous media, Region 2: free flow
- Source flow force $\mathbf{f} = [1.0, 0.0]$

Relevant part of the Python script

```
# Define permeability for regions 1 and 2
Ki = {1: 1./(100 * darcy), 2: 0})
mu = mus = 1e-3 # Water at 20° in Pa s
# Initialize TaylorHood Mixed base function
v, p = TaylorHood(mesh)
# Solve weak formulation
v, p = solve(v*mu*Ki*v + grad(v)*mus*grad(v) +
             grad(p)*v + v*grad(p) == v*[1., 0.],
             bc={p:{'Dirichlet':{1:0, 2:0}}})
```

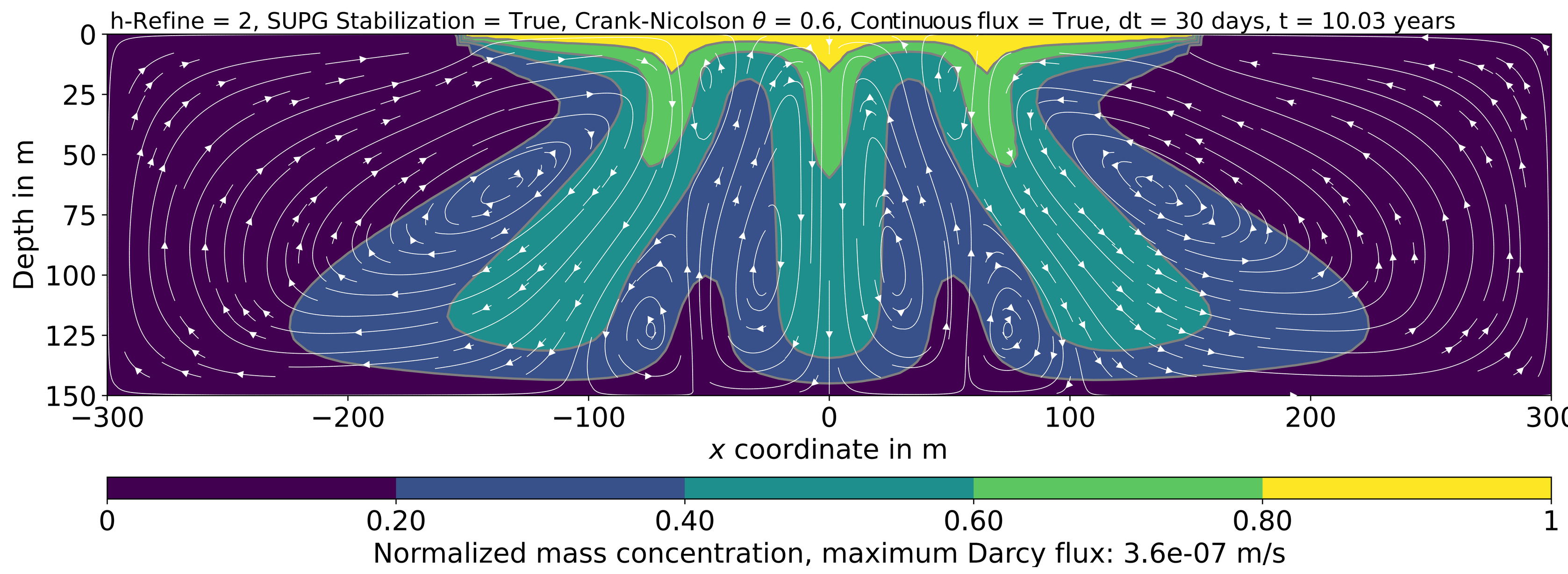


TH Thermo-Hydrogeological-Coupling

- Density (e.g., depends on temperature) driven solute flux **q** in porous media after Elder (1967)
- Connecting gravitational accelerated Darcy's law with mass transport due to advection and diffusion solved with higher level abstraction classes.

Relevant part of the Python script

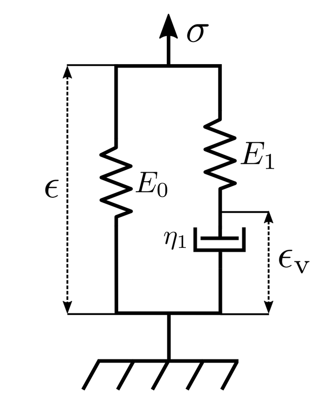
```
# Initialize generalized Darcy solver with Talor Hood Mixed elements
darcy = DarcySolver(mesh, K=K, bcP=bcH, bcV=bcQ, var=2, solver='scipy')
# Initialize generalized Advection-Diffusion solver using Streamline-Upwind
# Petrov-Galerkin stabilization and Crank-Nicolson time integration
advec = AdvectionDiffusionSolver(mesh, a=Dd*phi, c=phi,
                                 bc={'Dirichlet': {5:cMax, 3:cMin}},
                                 supg=True, theta=0.6, solver='scipy')
# Start main loop for all time steps and initialize starting values
q = None; c = 0.0
for i in range(maxYears//timeStepWidth):
    # Calculate concentration c for one timestep with flux q and for last concentration c
    c = advec.solve(vel=q, u0=c, times=[0, timeStepWidth])
    # Calculate flux and hydraulic head for density based force
    q, h = darcy.solve(fV=[0, -beta*C * c])
    # We mark the flux to be evaluated continuous instead of cell based
    q.continuous = continuous
```



M Linear viscoelastic modelling

- Simple viscoelastic 1D rheological model for uniaxial stress σ as sum of reversible and irreversible stress and time depending viscous strain ϵ_v

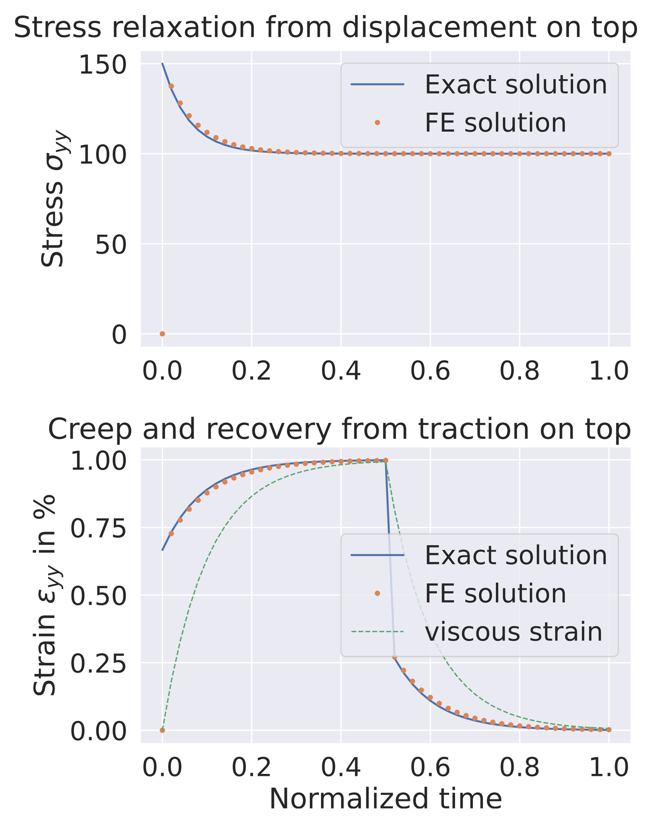
$$\sigma = E_0\epsilon + E_1(\epsilon - \epsilon_v) \quad \text{with} \quad \frac{\partial\epsilon_v}{\partial t} = \frac{E_1}{\eta_1}(\epsilon - \epsilon_v)$$



- Solve on 2D unit grid of 1 m × 1 m with 10 × 10 quadratic cells
- Two test scenarios applied with $E_0 = 10^4$, $E_1 = 5\cdot 10^3$, $\eta_1 = 3\cdot 10^2$.
  - Test 1: fixed displacement or 1 mm on top.
  - Test 2: traction on top $T = 100\,\forall\, t = [0..0.5]$, $T = 0\,\forall\, t = [0.5..1]$

Relevant part of the Python script

```
# Define strain
def eps(u):
    return sym(grad(u))
# Define viscos strain for times update
def epsv(u, ev):
    return 1/(1 + dt/tau) * (ev + dt/tau * eps(u))
# Define stress, C creates 2D Constitutive matrix
def sigma(u, ev):
    e = eps(u)
    return C(E0, mu)*e + C(E1, mu) * (e - epsv(u, ev))
# Initialize Vector valued space
v = VectorSpace(mesh, p=1)
# loop over time in 50 steps
for dt in np.linspace(0, 1, num=50):
    # Solve for displacement u. bc depends on test case.
    u = solve(grad(v)*sigma(v, ev) == 0,
              bc={'Dirichlet':{4:0.01}} # test 1
              bc={'Neumann':{4:traction(dt)}}) # test 2
    # Evaluate viscous strain needed for next time update
    ev = epsv(u, ev).eval()
```



✉ carsten.ruecker@bfe.base.de

www...org

gitlab.com/...