

O-D code repository

To execute the julia script contained in this directory :

- first **install julia** ≥ 1.10 (best using [juliaup](#))
- open a terminal in the "OD" folder
- **start a julia REPL and activate the project environment** by entering `julia --project .` . If the julia command is not found, run `. ~/.bashrc` (or `. ~/.zshrc` depending on your shell environment) to reload the configuration file.
- enter the package manager mode by typing a right square bracket `]`
- **install our three packages** that are not listed in the general registry :

```
add https://github.com/Leooop/SCAM.jl.git
add https://github.com/Leooop/DataFormatter.jl.git
add https://github.com/Leooop/ParametersEstimator.jl.git
```

- run the `instantiate` command to **install other project dependencies**
- finally, run `include("SCAM_OD_forward.jl")` or `include("SCAM_OD_parameters_inversion.jl")` to **run one of the available scripts**.

Scripts content

- `SCAM_OD_forward.jl` simulates both constant strain rate and constant stress experiments. Details can be found in the example section of the this README.
- `SCAM_OD_parameters_inversion.jl` shows the inversion procedure with live visualization of the model behavior as parameters get optimized. The Prior parameters are an intermediate result of parameter inversion against constant strain rate experimental data, whereas what is performed in this script is the full inversion against both constant strain rate and constant stress data. The last part of the script computes the posterior covariance matrix of the inverted parameters and a plot of the probability distribution of each parameter.

Introduction

This rheological model considers the growth of tensile cracks in a compressive stress state, based on the wing-crack model of [Ashby and Sammis \(1990\)](#) coupled with a sub-critical crack growth law (Charles, 1958), where the shear modulus is altered by the increase of the damage state of the material.

It uses an empirical linear dependency of the shear modulus on damage $D = \frac{4}{3}\pi N_v(l + \alpha a)^3$, where N_v is the number of cracks per unit volume, l is the length of each tensile crack growing from the tips of closed penny-shaped cracks of radius a and oriented at an angle $\psi = \cos^{-1} \alpha$.

The penny-shaped cracks normals are assumed to be contained in the σ_1 - σ_3 plane, such that [Ashby and Sammis \(1990\)](#) derived an expression for the stress intensity factor K_I at the tips of wing cracks, using linear elastic fracture mechanics, that can be plugged into the Charles Law to describe the slow extension of cracks under sustained applied stresses, and its effect on elastic modulus.

Long term behavior, post crack coalescence (at $D \sim 1$), can be represented by 2-dimensional Mohr-Coulomb plasticity in the same plane.

This model specifically describes the deformation of compact rocks under various conditions of confining pressures, strain rates or constant stress, in the brittle regime. Temperature dependence of brittle deformation is not taken into account.

Coupled with the unregistered packages [DataFormatter.jl](#) and [ParametersEstimator.jl](#), This model can be used to perform bayesian parameters inversion against triaxial experimental data under constant strain rate or brittle creep conditions.

Get started

To simulate the mechanical behavior of a damaged material, you first need to build a `Model` Instance.

The `Model` constructor takes two arguments :

- A `ConstitutiveModel` instance, representing the rheology of the material
- A `NumericalSetup` instance, including the geometry of the problem and setting control parameters for the simulation.

The class diagram of the next section helps understanding types relationships within this `Model` type.

Once a `model::Model` instance is created, you just need to call the `simulate` function to integrate your model's ODEs in time using [DifferentialEquations.jl](#):

```

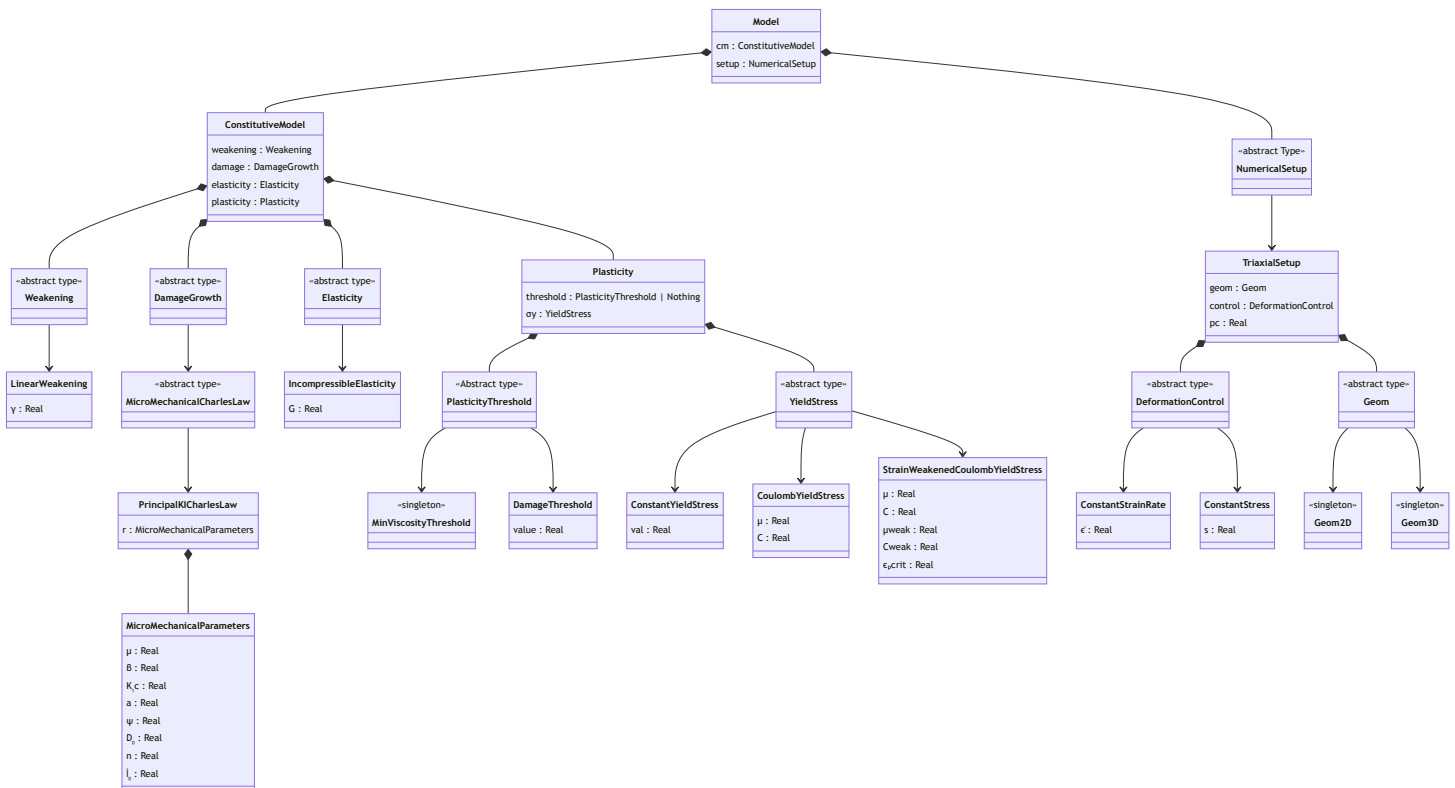
tspan = (0.0, 530)
sol = simulate(model, tspan;
    solver = Tsit5(), # ODE solver
    saveat = range(0, tspan[2]; length=500), # array of times saved in sol
    abstol = 1e-6, # absolute tolerance of the solver
    reltol = 1e-4, # relative tolerance of the solver
    maxiters = 1e5, # maximum number of solver iterations
    D1=nothing, # initial damage (if not specified or nothing:  $D(t_0) = D_0$ )
    Dmax=0.95, # maximum damage value
    stop_at_peak = false, # whether the solver should stop when peak stress is achieved
    cb=nothing # DiffEq Callbacks. If nothing, uses default callbacks appropriate to the problem
)

```

The result of `simulate` is a solution Object (more about it [here](#)), where the solution vector at each timestep holds the deviatoric stress or strain (depending on the deformation being performed under constant strain rate or constant stress conditions), damage, and accumulated plastic strain (if the chosen plastic yield stress is `StrainWeakenedCoulombYieldStress`).

Model architecture

The top level `Model` type displays the following type hierarchy :



The `ConstitutiveModel` type contains the parameterization for all bricks of the O-D rheology (equations 4, 5, 8, 19 and 23 of the manuscript):

- `weakening=LinearWeakening(γ)` - Linear weakening of shear modulus G as a function of damage D :

$$f(D) = \frac{\gamma - 1}{1 - D_0} D + \frac{1 - \gamma D_0}{1 - D_0}.$$

- `damage=PrincipalKICCharlesLaw(r)` - Micromechanical damage as defined by [Ashby and Sammis \(1990\)](#) coupled with Charles' Law:

$$\dot{D} = \frac{3D^{\frac{2}{3}} D_0^{\frac{1}{3}} \dot{l}_0}{\alpha a} \left(\frac{K_I}{K_{IC}} \right)^n,$$

with the expression of the stress intensity factor derived by [Bhat et al. \(2011\)](#):

$$K_I = \sqrt{\pi a} [(\sigma_3 A_3 - \sigma_1 A_1) (c_1 + c_2) + \sigma_3 c_3].$$

- `elasticity:IncompressibleElasticity(G)` - Incompressible elasticity gives rise to a damaged-elastic Maxwell rheology upon time differentiation of the elastic constitutive law including a damage dependent altered elastic modulus:

$$\dot{e}_{ij} = \frac{\dot{s}_{ij}}{2G_0 f(D)} + \frac{s_{ij}}{2\eta_D},$$

where

$$\eta_D = \frac{f^2(D) G_0}{|f'(D)| \dot{D}}.$$

Constructed ODE

Depending on `NumericalSetup.control` , the ODE that will be constructed will be:

- equation 27 if `NumericalSetup.control=ConstantStrainRate($\dot{\epsilon}$)` :

$$\dot{s}_{ax} = 2G_0 f(D) \left(\dot{e}_{ax} - \frac{s_{ax}}{2\eta_D} \right),$$

- equation 28 if `NumericalSetup.control=ConstantStress(σ)` :

$$\dot{e}_{ax} = \frac{s_{ax}}{2\eta_D},$$

Examples

The code in this section can also be found in the [example](#) folder.

Constant axial strain rate axisymmetric simulation

Let's load SCAM , an ODE solver library and a plotting package

```
using SCAM
using OrdinaryDiffEq
using Plots
```

Let's assume that we want to model the mechanical behavior of a rock under axisymmetric loading (i.e. $\sigma_2 = \sigma_3 = 50$ MPa) and constant axial strain rate of -10^{-5} s^{-1} .

We need a `NumericalSetup` type corresponding to the above conditions :

```
ε̇ = -1e-5
setup = TriaxialSetup(
    geom = Geom3D(),
    control = ConstantStrainRate(ε̇),
    pc = 50e6
)
```

Note that if we used `Geom2D()` instead of `Geom3D()` , a plane-strain setup would be generated.

We seek to model the damaged-elastic part of the deformation with Incompressible elasticity and a shear modulus of 30 GPa :

```
elast = IncompressibleElasticity(G = 30e9)
```

Assume penny-shaped cracks of radius $a = 0.5$ mm and oriented at an angle $\psi = 45^\circ$ (actually the only possible value) are initially present in the material and are characterized by damage $D_0 = 0.2$ (function of their number per unit volume).

```

mmp = MicroMechanicalParameters(
    μ=0.7,
    ψ=45,
    a=0.5e-3,
    D₀=0.2,
    n=10,
    KIC=2e6,
    i₀=1e-2
)

```

where μ is the coefficient of friction of the material. K_{IC} is the fracture toughness of the material, n is the Charles' exponent and \dot{l}_0 the reference tensile crack speed. The Charles (1958) subcritical crack growth law reads :

$$\dot{l} = \dot{l}_0 \left(\frac{K_I}{K_{IC}} \right)^n .$$

The evaluation of the stress intensity factor K_I can be performed using the principal stresses (see Ashby and Sammis, 1991) which we indicate by wrapping the micromechanical parameters in the following type :

```

damage_growth = PrincipalKICCharlesLaw(mmp)

```

Now we need a connection between damage and the mechanical behavior. This is done through a damage-induced weakening of the shear modulus. In this package you can choose between a linear and an asymptotic weakening. Let's use the form with $\gamma = 0.5$. This parameter corresponds to the residual value of shear modulus when the material is broken (i.e., $D = 0$) :

```

weak = LinearWeakening(0.5)

```

We can now assemble the `ConstitutiveModel` type

```

cm = ConstitutiveModel(
    weakening = weak,
    damage = damage_growth,
    elasticity = elast,
    plasticity = nothing
)

```

and finally the full model, also using the setup informations :

```
model = Model(cm,setup)
```

The coupled integration of axial stress and damage is then performed up to 530 s by invoquing

```
tspan = (0.0, 530)
sol = simulate(model, tspan;
    solver = Tsit5(), # ODE solver
    saveat = range(0, tspan[2]; length=500),
    abstol = 1e-6,
    reltol = 1e-4,
    maxiters = 1e5,
    Di=nothing, # if nothing D(t0) = D0
    Dmax=0.95,
    stop_at_peak = false,
    cb=nothing # whatever DiffEq Callback. If nothing, uses the appropriate callbac
)
```

If you want to use you own ODE solver, the package also provides the lower level function

```
update_derivatives!(du::Vector,u::Vector,p::NamedTuple,t::Any,model::Model)
```

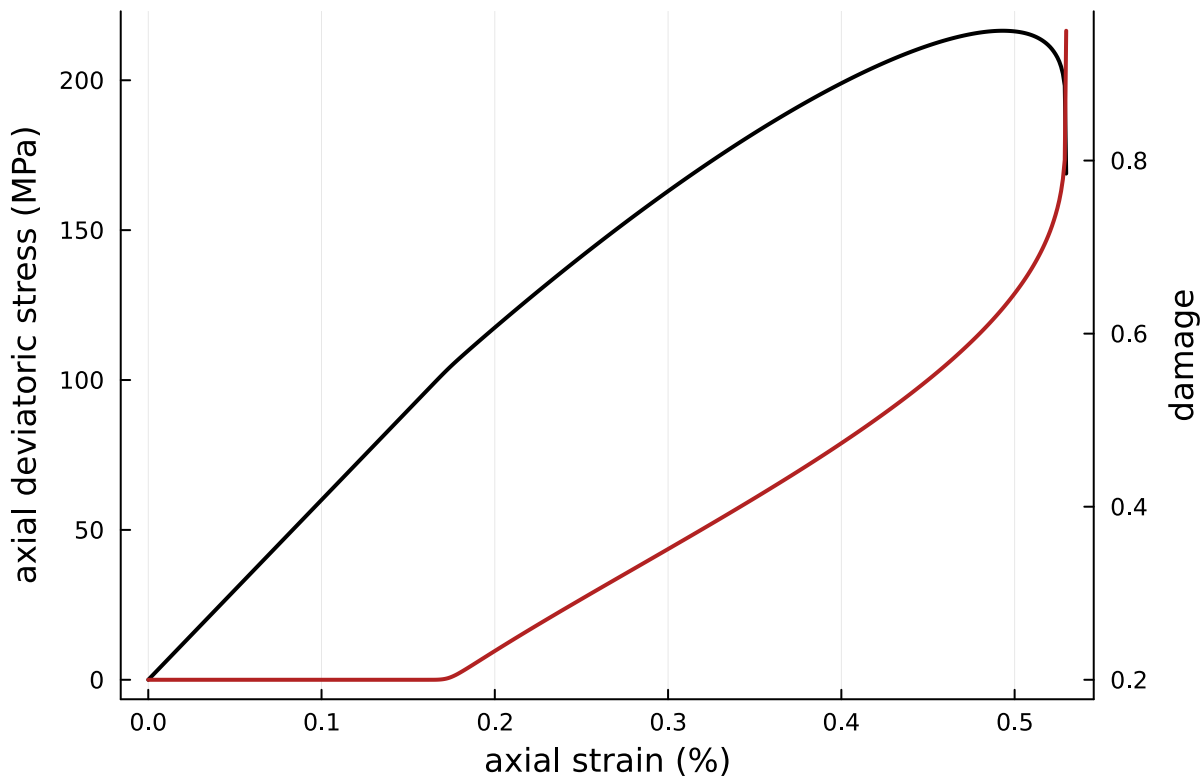
returning the relevant vector of time derivatives `du` from state `u` and model `model`. `p` must be a `NamedTuple` containing the field `Dmax`. Overall for Constant strain rate models the state vector contains the axial deviatoric stress and damage, whereas for constant stress setup, the deviatoric stress is replaced by the creep strain rate. `t` can be input anything, it is not used, and is there to satisfy `OrdinaryDiffEq.jl` interface.

The `sol` return variable is the `Solution` type ouput by `OrdinaryDiffEq`. We can straightforwardly plot it :

```

es = -sol.t.*ė.*100 # in %
os = sol[1,:]
Ds = sol[2,:]
plot(es, -os./1e6,
      c=:black,
      lw=2,
      label="",
      xlabel = "axial strain (%)",
      ylabel = "axial deviatoric stress (MPa)"
)
plot!(twinx(), es, Ds,
      c = :firebrick,
      lw=2,
      label = "",
      ylabel= "damage"
)

```



Constant stress (brittle creep) axisymmetric simulation

In order to model the brittle creep (also referred to as static fatigue) behavior of the same material under the same confining pressure, we need to get the rock to the desired stress and then keep the axial stress fixed. Let's evaluate brittle creep behavior at $\sigma_c = 200$ MPa. We can reuse previous results to obtain the damage state at this stress level :


```

creep_stress = -200e6
os_to_peak = os[1:findfirst(diff(os).>= 0)]
id = argmin(abs.(os_to_peak .- creep_stress))

oc = os[id]
Dic = Ds[id]

```

We then create a new constant stress setup and reinstantiate a model,

```

setup_creep = TriaxialSetup(
    geom = Geom3D(),
    control = ConstantStress(oc),
    pc = 50e6
)

model_creep = Model(cm, setup_creep)

```

integrate again (notice the initial damage initialized at D_{ic})

```

tspan = (0.0, 1500)
sol_creep = simulate(model_creep, tspan;
    solver = Tsit5(), # ODE solver
    saveat = range(0, tspan[2]; length=500),
    abstol = 1e-6,
    reltol = 1e-4,
    maxiters = 1e5,
    Di = Dic, # if nothing D(t0) = D0
    Dmax=0.95,
    stop_at_peak = false,
    cb=nothing # whatever DiffEq Callback. If nothing, uses the appropriate callbac
)

```

and plot !

```

ts_creep = sol_creep.t
es_creep = -sol_creep[1,:].*100 # in %
Ds_creep = sol_creep[2,:]

plot(ts_creep, es_creep,
      c=:black,
      lw=2,
      label="strain",
      legend=:topleft,
      xlabel = "time (s)",
      ylabel = "axial creep strain (%)"
)
plot!(twinx(), ts_creep, Ds_creep,
      c = :firebrick,
      lw=2,
      label = "",
      ylabel= "damage"
)

```

