Supplement of `PorousLab`: FEM framework for fractured porous media

Danilo Cavalcanti*[1,2,3], Rafael L. Rangel[4], Carlos A. Mendes[3], Sebastia Olivella[1,2], Victor Vilarrasa†[5], Luiz F. Martha[3], Deane Roehl[3], Ignasi de-Pouplana[1,2], and Guillermo Casas[1]

[1]Centre Internacional de Mètodes Numèrics a l'Enginyeria (CIMNE), Barcelona, Spain
[2]Universitat Politècnica de Catalunya - BarcelonaTech (UPC), Barcelona, Spain
[3]Pontifical Catholic University of Rio de Janeiro (PUC-Rio), Rio de Janeiro, Brazil
[4]University of Twente, Enschede, Netherlands
[5]Global Change Research Group (GCRG), IMEDEA, CSIC-UIB, Esporles, Spain

S1 Simulation Workflow

This section complements the workflow described in the main manuscript by providing additional details on the pre-processing, post-processing, and graphical-interface utilities available in `PorousLab`. A simulation is defined through two main objects: the model object, `mdl`, and the analysis object, `anl`. The model object stores the finite-element mesh, material data, boundary and initial conditions, and physics-specific state variables. The analysis object controls the solution procedure, including the type of analysis, nonlinear scheme, time-stepping parameters, and convergence tolerances. This separation is used consistently across the different physics types implemented in the framework.

S1.1 Pre-processing

The pre-processing stage defines the simulation's geometric and physical parameters. This includes mesh generation, assignment of boundary and initial conditions, and material configuration. All these tasks are encapsulated within the `Model` class and its derived physics-specific classes.

S1.1.1 Mesh generation

The `Model` object defines the finite-element mesh using two key attributes: `NODE` and `ELEM`. The `NODE` attribute is a matrix where each row stores the Cartesian coordinates of a mesh node. The `ELEM` attribute is a cell array in which each cell contains the connectivity of one element, i.e., the indices of the nodes that define the element geometry. This cell-based implementation for `ELEM` increases flexibility, allowing a mesh to be composed of different element types.

The mesh can be generated internally using helper functions such as `regularMesh`, which create structured quadrilateral or triangular grids, or externally using tools like Gmsh [1]. For the latter, we provide an auxiliary Python function to export meshes generated with the Gmsh API into a format compatible with PorousLab[1]. Once created, the mesh is assigned to the model using the `setMesh` method, which also initializes basic internal properties such as the number of nodes, number of elements, and degrees of freedom.

S1.1.2 Boundary conditions

The framework supports both Dirichlet and Neumann boundary conditions, which can be applied in three distinct ways:

- At a **specific node**, using methods like `setDirichletBCAtNode` and `setNeumannBCAtNode`.
- At a **specific coordinate point**, using `setDirichletBCAtPoint` and `setNeumannBCAtPoint`, which automatically identify the closest node.

*Corresponding author. Email: dborges@cimne.upc.edu
†Corresponding author. Email: victor.vilarrasa@csic.es
[1]`github.com/dbcavalcanti/porousLab/wiki/import-gmsh`

- Along a **boundary of a rectangular domain** (e.g., “left”, “right”, “top”, or “bottom”), via `setDirichletBCAtBorder` or `setNeumannBCAtBorder`, which internally identify all nodes along the specified edge within a given range.

Physics-specific classes provide wrappers with meaningful names for setting problem-relevant conditions. For instance, in mechanical problems, the method `setDisplacementDirichletBCAtBorder` is available, while hydraulic models use `setPressureDirichletBCAtNode`, among others.

Neumann boundary conditions on borders are implemented using equivalent nodal forces computed based on the boundary segment length and element geometry.

S1.2 Post-processing

Post-processing tools in the framework are designed to offer clear visual feedback on simulation results and to facilitate convergence and performance analysis.

The `FEMPlot` class is dedicated to graphical outputs, offering:

- **Field visualization**: Using `plotField`, users can display results like displacements, pressures, or stresses across the domain. The method internally gathers element-wise vertex data and constructs unified visual patches over the mesh.
- **Field evaluation along segments**: The method `plotFieldAlongSegment` evaluates and plots a user-defined field along a linear segment defined by two endpoints. This is useful for inspecting gradients and internal profiles.
- **Deformed configuration**: Mechanical models support visualization of the deformed mesh with amplification, through `plotDeformedMesh`, enabling qualitative assessment of displacements.

In addition, simulation results can be printed to the console in tabular format using the `printResults` method. This includes nodal values of primary variables and can be customized via the `printResultsHeader` method defined in each physics-specific model.

S1.3 Graphical User Interface

A graphical user interface developed with MATLAB App Designer is included as an auxiliary tool for setting up and running mechanical analyses. The interface follows the same sequence adopted in script-based simulations: model creation, mesh generation or import, material definition, boundary-condition prescription, analysis setup, solution, and post-processing.

The GUI process begins with the model creation, where the user selects the type of physics to be modeled. After defining the desired physics, the user is prompted to generate or import the mesh. For rectangular domains, we offer an auxiliary process to generate a structured mesh by setting the domain’s dimensions and number of subdivisions. Figure S1 illustrates these steps.

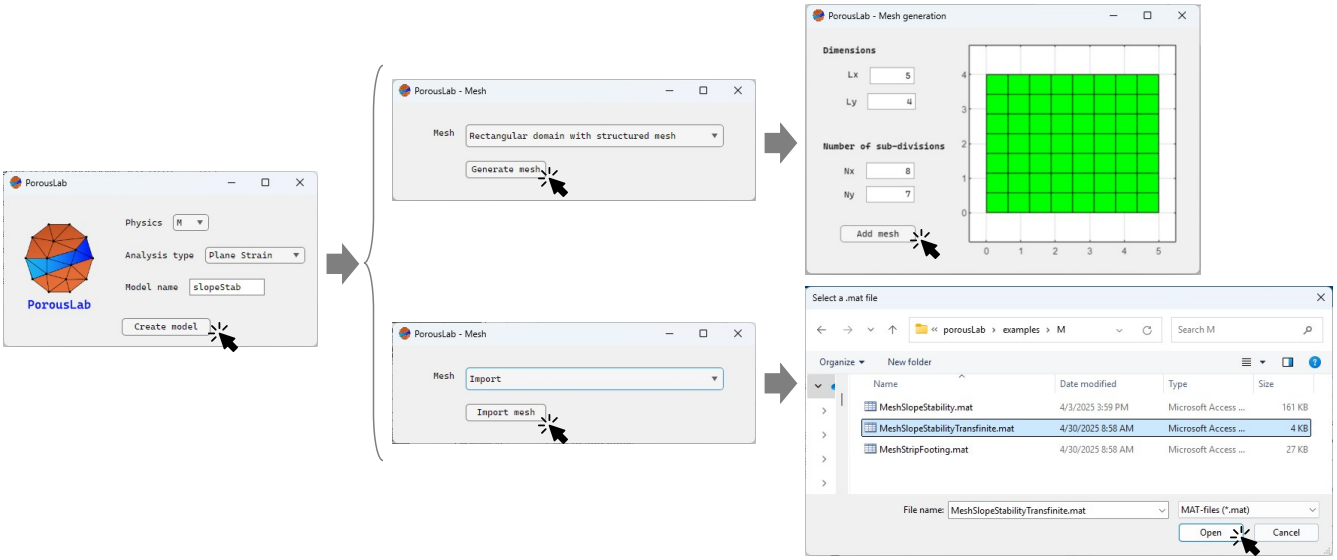

Figure S1: GUI: Physics definition and mesh generation.

After generating or importing the mesh, the physics window appears, providing a clear step-by-step interface for applying the necessary simulation settings. Each tab corresponds to a task that must be completed in sequence, reflecting the standard workflow followed when writing code manually.

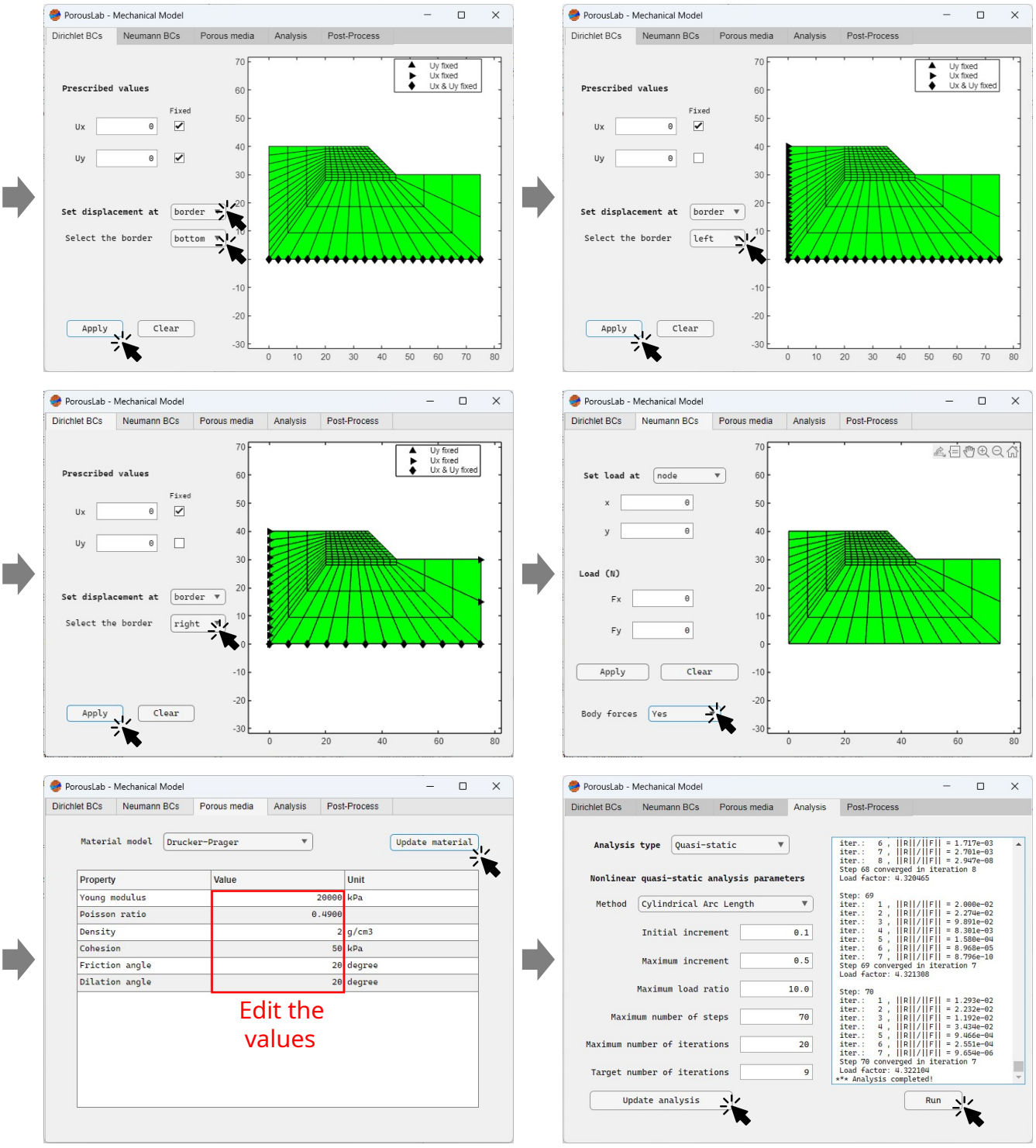

Figure S2: GUI: Setting the model and analysis.

Once the mesh and physics are defined, the user proceeds to set the boundary conditions for the model. The GUI provides

an easy way to apply prescribed displacements (Dirichlet boundary conditions) and loads (Neumann boundary conditions). Users can specify the location and type of boundary conditions and apply them directly to the model. Additionally, the material properties can be modified in the same window, where users can select a material model (e.g., Linear Elastic) and input relevant properties such as Young's modulus, Poisson's ratio, and density (Fig. S2).

With the boundary conditions and material properties set, users can move to other settings, such as additional solver options or preprocessing tasks (Fig. S3), before running the simulation. The clear layout of the interface ensures that users can easily navigate through these stages, from model creation to post-processing, making the simulation process more accessible.

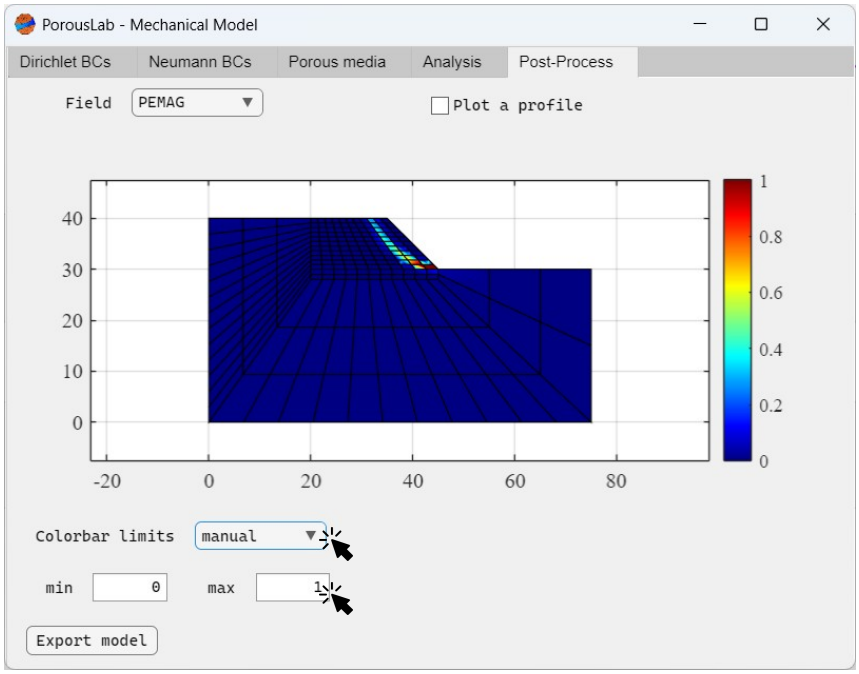

Figure S3: GUI: Post-process.

S2 Example scripts

This section complements the verification examples presented in Sect. 6 of the main manuscript by documenting the corresponding `PorousLab` input scripts. The physical definition, reference solutions, and numerical results are discussed in the main text. Therefore, the focus here is restricted to the practical organization of the scripts and to the main framework commands required to reproduce each case. For clarity, the figures showing the geometry and boundary conditions are reproduced in this supplement, so that the script listings can be interpreted without repeatedly referring back to the main manuscript.

In the following examples, it should be noted that PorousLab does not enforce a unit system. Consequently, the numerical values reported in each script are meaningful only when a consistent set of units is adopted.

S2.1 Strip footing bearing capacity

This example corresponds to the strip-footing verification case discussed in Sect. 6.1 of the main manuscript. Here, the emphasis is on the script structure used to define and run the mechanical elastoplastic analysis.

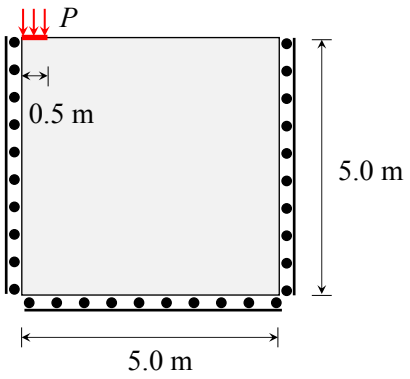

Figure S4: Strip footing: geometry and boundary conditions.

We generate a 5×5 m rectangular mesh with 30×30 linear quadrilaterals using `regularMesh` (Listing S1). The last four parameters control a smooth spacing warp to make the mesh denser near the load application. The linear mesh is converted to quadratic with `convertToQuadraticMesh` and node IDs can be reordered with `resequenceNodes` to reduce matrix bandwidth before assembly.

```
%% MODEL
% Create model
mdl = Model_M();
% Set model options
mdl.intOrder = 2
%% MESH
% Generate mesh
[node, elem] = regularMesh(5.0, 5.0, 30, 30, [], [], 'ISOQ4', 0.1, 0.92, 1.0, 0.7);
[node, elem] = convertToQuadraticMesh(node, elem);
% Set mesh to model
mdl.setMesh(node,elem);
mdl.resequenceNodes();
```

Listing S1: Model and mesh

Listing S2 defines the porous medium properties using a Drucker–Prager plasticity model. The rock parameters, including density, stiffness, and strength properties, are then assigned to the model. To use the Mohr-Coulomb plasticity model, the user should use `rock.mechanical = 'mohrCoulomb'`. For these two elastoplastic models, alternative return mapping algorithms are available, and can be used setting: `rock.stressIntAlgorithm = 'alternative'`. They correspond to the implementations presented in Souza Neto et al. [2].

```
%% MATERIALS
% Create porous media
rock = PorousMedia('rock');
rock.mechanical    = 'druckerPrager';  % Mechanical constitutive law
rock.Young         = 1.0e+7;           % Young modulus (kPa)
rock.nu            = 0.48;             % Poisson ratio
rock.cohesion      = 490.0;            % Cohesion (kPa)
rock.frictionAngle = 20.0*pi/180;      % Friction angle (rad)
rock.dilationAngle = 20.0*pi/180;      % Dilation angle (rad)
rock.MCmatch       = "planestrain";    % How the surfaces match the Mohr-Coulomb ones
rock.stressIntAlgorithm = 'alternative'; %
% Set materials to model
mdl.setMaterial(rock);
```

Listing S2: Material

Listing S3 applies the boundary conditions. The horizontal displacement is fixed on the left and right boundaries of the domain, and the vertical displacement is fixed on the bottom boundary. A pressure load is applied in the region of the footing, with a value corresponding to Prandtl's analytical solution.

```
%% DIRICHLET BOUNDARY CONDITIONS
mdl.setDisplacementDirichletBCAtBorder('left',   [0.0, NaN]);
mdl.setDisplacementDirichletBCAtBorder('right',  [0.0, NaN]);
mdl.setDisplacementDirichletBCAtBorder('bottom', [NaN, 0.0]);
%% NEUMANN BOUNDARY CONDITIONS
% Prandtl's limit pressure
Nq = exp(pi*tan(rock.frictionAngle)) * tan(pi/4 + rock.frictionAngle/2)^2;
Nc = (Nq - 1.0) * cot(rock.frictionAngle);
Plim = rock.cohesion * Nc;
% Apply pressure load at the footing (0 < x < 0.5)
mdl.addLoadAtBorder('top', 2, -Plim, [0.0 , 0.5]);
```

Listing S3: Boundary and initial conditions

Finally, Listing S4 configures and executes a quasi-static nonlinear analysis using the Generalized Displacement control. The vertical displacement of the point on the top left is monitored to track the load-displacement response. Post-processing includes visualizing the load curve and the plastic strain magnitude.

```
%% PROCESS
% Configure analysis
anl = Anl_NonlinearQuasiStatic('GeneralizedDisplacement');
anl.adjustStep     = true;
anl.increment      = 0.1;
anl.max_increment  = 1.0;
anl.max_lratio     = 2.0;
anl.max_step       = 100;
anl.max_iter       = 20;
anl.trg_iter       = 4;
anl.tol            = 1.0e-4;
% Node and DOF used to plot Load Factor vs Displacement
ndId = mdl.closestNodeToPoint([0.0, 5.0]);
anl.setPlotDof(ndId, 2);
% Run analysis
anl.run(mdl);
%% POST-PROCESS
anl.plotCurves();                  % Load vs. Displacement curve
mdl.plotField('PEMAG',[0.0,0.01]); % Plastic strain magnitude
```

Listing S4: Process and post-processing

S2.2 Stresses changes along a fault in a pressurized reservoir

This example corresponds to the pressurized-reservoir verification case discussed in Sect. 6.2 of the main manuscript. Here, the focus is on the script commands used to prescribe an external pore-pressure field in a mechanical analysis and to recover stress quantities along an embedded fault. The geometry and boundary conditions are shown again in Fig. S5 to identify the coordinates and regions used in the input script.

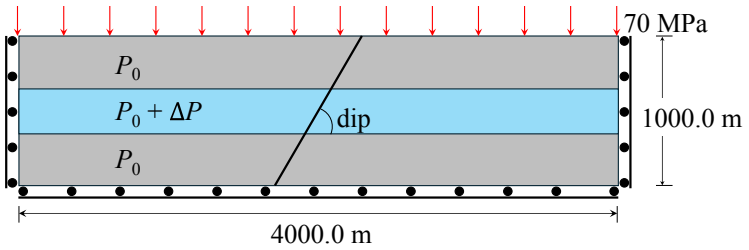

Figure S5: Boundary conditions and geometry adopted for the pressurized reservoir problem.

In Listing S5, the mechanical model attribute `addPorePressure` is a flag that indicates if the model has a pore-pressure field that will contribute to the external force vector.

```
%% MODEL
mdl = Model_M();
% Set model options
mdl.addPorePressure = true;
%% MESH
[node, elem] = regularMesh(4000.0, 1000.0, 60, 60, [], [], 'ISOQ4', 0.5, 0.7, 0.5, 0.8);
% Set mesh to model
mdl.setMesh(node, elem);
```

Listing S5: Model and mesh

Listing S6 defines the material of the continuum, which in this problem is linear elastic.

```
%% MATERIALS
% Create porous media
rock = PorousMedia('rock');
rock.Young = 11868.0e+6;        % Young modulus (Pa)
rock.nu    = 0.29;              % Poisson ratio (-)
% Set materials to model
mdl.setMaterial(rock);
```

Listing S6: Material

Listing S7 sets the boundary conditions. The normal displacement is fixed on the lateral edges ($u_x = 0$) and on the bottom edge ($u_y = 0$). A vertical compressive load of 70 MPa is applied on the top edge the model. The pore-pressure field is then added as a nodal vector. Here it is initialized uniformly at 35 MPa, but a hydrostatic profile could be easily prescribed instead.

```
%% BOUNDARY CONDITIONS
% Displacements
mdl.setDisplacementDirichletBCAtBorder('left',   [0.0, NaN]);
mdl.setDisplacementDirichletBCAtBorder('right',  [0.0, NaN]);
mdl.setDisplacementDirichletBCAtBorder('bottom', [NaN, 0.0]);
% Pressure load (N/m)
mdl.addLoadAtBorder('top', 2, -70.0e6);
% Set external pore-pressure (Pa)
P0 = 35.0e6;
P = P0 * ones(mdl.nnodes,1);
mdl.setPorePressureField(P);
```

Listing S7: Boundary and initial conditions

The fault is modeled with the embedded approach, by first creating a `Discontinuity` from the fault endpoints, then setting the cohesive law and stiffnesses and configuring the EFEM options: non-symmetric formulation (`symmetricSDAEFEM=false`) with optional condensation of enrichment DOFs (`condenseEnrDofs`) (Listing S8). The fault is finally added with `addPreExistingDiscontinuities`, which intersects the line with the mesh and builds the discontinuity segments for assembly.

```
%% DISCONTINUITIES
% Fault coordinates
Xd = [ 1711.325 , 0.0;
       2288.675 , 1000.0 ];
fault = Discontinuity(Xd, true);
% Set fracture material properties
fault.cohesiveLaw     = 'elastic';
fault.normalStiffness = 1.0e15;                                                    % (Pa/m)
fault.shearStiffness  = fault.normalStiffness * (1 - 2*rock.nu)/(2*(1-rock.nu));   % (Pa/m)
% Set embedded formulation options
mdl.condenseEnrDofs  = false;
mdl.symmetricSDAEFEM = false;
% Add fault to the model
mdl.addPreExistingDiscontinuities(fault);
```

Listing S8: Discontinuity

Before imposing the reservoir overpressure, the model is equilibrated with a linear analysis to recover the in-situ stress state. Displacements and strains are then reset to zero, so that the subsequent step isolates the effect of the pressure perturbation. The nodal pore-pressure vector is updated by increasing `P` only at nodes within the reservoir layer, identified with `isInsideRectangle`, and the system is solved again, as in Listing S9

```
%% PROCESS
% Run a linear analysis to initialize the stresses
anl = Anl_Linear();
anl.run(mdl);
% Reset displacement and strains
mdl.resetDisplacements();
% Identify nodes in the reservoir region
tol = 1.0e-5;
reservoir = isInsideRectangle(mdl.NODE, [0.0-tol,350.0-tol], [4000.0+tol,650.0+tol]);
% Update pressure at the reservoir
P(reservoir == 1) = P0 + 20.0e6;     % (Pa)
% Update the pore-pressure and solve it
mdl.setPorePressureField(P);
anl.run(mdl);
```

Listing S9: Process

To analyze the results, we plot vertical profiles at $x = 2000$ m for the continuum normal stresses and extract cohesive tractions along the fault (Listing S10).

```
%% POST-PROCESS
% Vertical profiles at x = Lx/2 = 2000 m
mdl.plotFieldAlongSegment('Sy',[2000.0,0.0],[2000.0,1000.0],100,'y');
mdl.plotFieldAlongSegment('Sx',[2000.0,0.0],[2000.0,1000.0],100,'y');
% Fault cohesive tractions
mdl.plotFieldAlongDiscontinuiy('St',1,'y');
mdl.plotFieldAlongDiscontinuiy('Sn',1,'y');
```

Listing S10: Post-processing

S2.3 Fluid flow through the foundation of a dam

This example corresponds to the single-phase seepage verification case discussed in Sect. 6.3 of the main manuscript. Here, the focus is on the script commands used to define a hydraulic model, prescribe pressure boundary conditions over selected boundary segments, introduce embedded hydraulic discontinuities, and extract pore-pressure profiles. The geometry and boundary conditions are shown again in Fig. S6 to identify the boundary intervals and fracture configurations used in the input script.

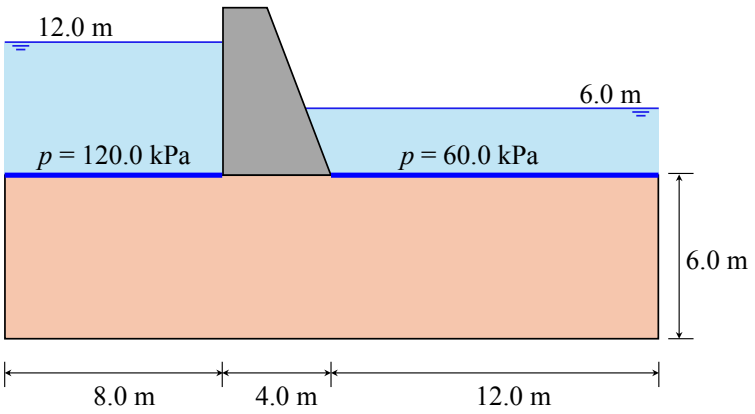

Figure S6: Boundary conditions and geometry adopted for the fluid flow through the dam foundation.

The mesh is structured with 48×12 linear quadrilaterals (Listing S11).

```
%% MODEL
mdl = Model_H();
%% MESH
[node, elem] = regularMesh(24.0, 6.0, 48, 12);
mdl.setMesh(node,elem);
```

Listing S11: Model and mesh

In the H physics, the material requires a porous medium and a fluid object (Listing S12).

```
%% MATERIALS
% Create fluids
water = Fluid('water');
% Create porous media
rock = PorousMedia('rock');
rock.K   = 1.0194e-14;  % Intrinsic permeability (m2)
rock.phi = 0.3;         % Porosity (-)
% Set materials to model
mdl.setMaterial(rock, water);
```

Listing S12: Material

The pore pressure corresponding to the water table on each side of the dam is prescribed in the top border of the domain. The top boundary has two Dirichlet segments ($p = 120$ kPa for $0 \leq x \leq 8$ m and $p = 60$ kPa for $12 \leq x \leq 24$ m) (Listing S13). All other boundaries are no-flow.

```
%% BOUNDARY CONDITIONS
mdl.setPressureDirichletBCAtBorder('top', 120.0e3, [0.0, 8.0]);      % (Pa)
mdl.setPressureDirichletBCAtBorder('top', 60.0e3 , [12.0, 24.0]);    % (Pa)
```

Listing S13: Boundary and initial conditions

We provide two auxiliary functions to create a set of discontinuities inside a rectangular domain (Listing S14). To generate a set of parallel fractures that cross the entire domain with a given spacing and angle: `generateParallelFractures`. To generate a n random fracture: `generateRandomFractures`.

```
%% DISCONTINUITIES
rng(123,'twister');  % Reproducible apertures/sets
% Parallel set across the domain (spacing=1 m, angle=-pi/4)
FractureDataDamFoundation = generateParallelFractures(24.0, 6.0, -pi/4.0, 1.0);
% Or: random set of 20 fractures with min length 1 m
% FractureDataDamFoundation = generateRandomFractures(24.0, 6.0, 20, 1.0, 'Seed', 12345);
% Create discontinuities
nd = length(FractureDataDamFoundation);   % Number of discontinuities
fractures(1,nd) = Discontinuity();
for i = 1:nd
    fractures(i) = Discontinuity(FractureDataDamFoundation{i}, true);
end
% Set fracture material properties
for i = 1:nd
    fractures(i).liquidFluid = water;
    fractures(i).initialAperture = 1e-4;  % (m)
end
% Add fractures to model
mdl.addPreExistingDiscontinuities(fractures);
```

Listing S14: Discontinuities

A linear analysis is performed to obtain the steady-state response (Listing S15). Post-processing reports the pore-pressure field and a horizontal profile at mid-height ($y = 3$ m).

```
%% PROCESS
anl = Anl_Linear();
anl.run(mdl);
%% POST-PROCESS
mdl.plotField('Pressure');
mdl.plotFieldAlongSegment('Pressure', [0.0, 3.0], [24.0, 3.0], 500, 'x');
```

Listing S15: Process and post-processing

S2.4 Fluid injection into a single fracture

This example corresponds to the hydro-mechanical fracture-injection verification case discussed in Sect. 6.4 of the main manuscript. Here, the focus is on the script structure used to define a coupled hydromechanical analysis with an embedded discontinuity. The example illustrates the creation of the porous medium and fluid objects, the definition of an elastic cohesive fracture, the two-stage initialization and injection procedure, and the activation of aperture updates during the transient analysis. The geometry and boundary conditions are shown again in Fig. S7 to clarify the coordinates used to prescribe the fracture and the pressure boundary conditions.

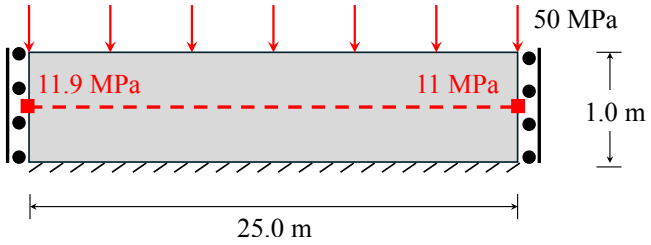

Figure S7: Boundary conditions and geometry adopted for the Wijesinghe problem

Listing S16 presents the hydromechanical model initialization and the mesh generation.

```
%% MODEL
% Create model
mdl = Model_HM();
%% MESH
[node, elem] = regularMesh(25.0, 1.0, 50, 21, [], [], 'ISOQ4', 0.0, 0.8, 0.5, 0.95);
mdl.setMesh(node, elem);
```

Listing S16: Model and mesh

In the `HM` physics, define both a porous medium and a fluid (Listing S17). The values of the material parameters are defined following Watanabe et al. [3].

```
%% MATERIALS
% Create fluids
water = Fluid('water');
% Create porous media
rock = PorousMedia('rock');
rock.K     = 1.0e-21;        % Intrinsic permeability (m2)
rock.phi   = 0.001;          % Porosity (-)
rock.Young = 60.0e+9;        % Young modulus (Pa)
rock.nu    = 0.0;            % Poisson ratio (-)
% Set materials to model
mdl.setMaterial(rock, water);
```

Listing S17: Material

Listing S18 presents the boundary conditions. The lateral edges are constrained in the normal direction, the base is fixed, and a vertical compressive traction acts on the top boundary. A uniform pore pressure is prescribed for the initialization step.

```
%% BOUNDARY AND INITIAL CONDITIONS
% Displacements
mdl.setDisplacementDirichletBCAtBorder('bottom', [0.0, 0.0]);
mdl.setDisplacementDirichletBCAtBorder('left',   [0.0, NaN]);
mdl.setDisplacementDirichletBCAtBorder('right',  [0.0, NaN]);
% Loads (N/m)
mdl.addLoadAtBorder('top', 2, -50.0e6);
% Pressure (Pa)
mdl.setPressureDirichletBCAtDomain(11.0e6);
```

Listing S18: Boundary conditions

A pre-existing horizontal fracture at mid-height is embedded with elastic cohesive behavior, as in Listing S19.

```
%% DISCONTINUITY
% Polyline that defines the discontinuity
Xd = [ 0.0  , 0.5;
       25.0 , 0.5];
fracture = Discontinuity(Xd, true);
% Set fracture material properties
fracture.cohesiveLaw     = 'elastic';
fracture.shearStiffness  = 100.0e9;    % (Pa/m)
fracture.normalStiffness = 100.0e9;    % (Pa/m)
fracture.initialAperture = 1.0e-5;     % (m)
fracture.fluid           = water;
% Add fracture to the model
mdl.addPreExistingDiscontinuities(fracture);
```

Listing S19: Discontinuity

The analysis proceeds in two stages (Listing S20). Stage 1 initializes in-situ stresses under uniform pressure. Stage 2 imposes injection at the left fracture mouth and a reference pressure at the right boundary, enables aperture updates, and advances in time.

```
%% PROCESS
% Run analysis to initialize the model
anl = Anl_Transient("Newton");
% Analysis parameters:   ti,  dt,  tf
anl.setUpTransientSolver(0.0, 1.0, 1.0);
anl.run(mdl);
% Update the pore-pressure boundary condition (Pa)
mdl.resetPressureDirichletBC();
for i = 1:mdl.nnodes
    if (mdl.NODE(i,1)<1.0e-12 && (abs(mdl.NODE(i,2)-0.5))<0.03)
        mdl.setPressureDirichletBCAtNode(i,11.9e6);
    end
end
mdl.setPressureDirichletBCAtBorder('right', 11.0e6);
mdl.updateDirichletBC();
% Allow the discontinuities to update the aperture with the mechanical deformations
mdl.setUpdateAperture(true);
% Reset analysis parameters and run it
% Analysis parameters: ti, dt, tf, dtmax, dtmin, adaptStep
anl.setUpTransientSolver(0.0, 0.01, 500.0, 1, 0.0001, true);
anl.run(mdl);
```

Listing S20: Process

Post-processing extracts fracture aperture and pore pressure along the discontinuity (Listing S21).

```
%% POST-PROCESS
mdl.plotFieldAlongDiscontinuiy('Aperture',1,'x');
mdl.plotFieldAlongDiscontinuiy('Pressure',1,'x');
```

Listing S21: Post-processing

S2.5 McWhorter problem

This example corresponds to the two-phase flow verification case discussed in Sect. 6.5 of the main manuscript. Here, the focus is on the script commands used to define a two-phase flow analysis, assign liquid and gas fluid objects, prescribe capillary-pressure and relative-permeability laws, and convert saturation-based initial and boundary conditions into the pressure variables used by the formulation. The geometry and boundary conditions are shown again in Fig. S8 to identify the saturation and gas-pressure data imposed in the script.

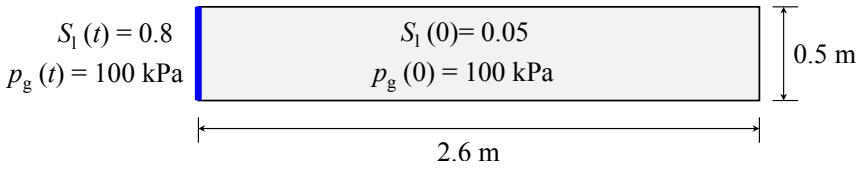

Figure S8: Boundary conditions and geometry adopted for the McWhorter and Sunada problem.

The mesh is structured with 100×1 linear quadrilaterals (Listing S22).

```
%% MODEL
% Create model
mdl = Model_H2();
%% MESH
[node, elem] = regularMesh(2.6, 0.5, 100, 1);
mdl.setMesh(node, elem);
```

Listing S22: Model and mesh

Listing S23 defines water, gas, and the porous medium with Brooks–Corey capillary pressure and relative permeabilities.

```
%% MATERIALS
% Create fluids
water = Fluid('water');
gas   = Fluid('gas');  gas.mu = 0.005;   % Gas viscosity (Pa*s)
% Create porous media
rock = PorousMedia('rock');
rock.K                  = 1.0e-10;        % Intrinsic permeability (m2)
rock.phi                = 0.15;           % Porosity (-)
rock.Slr                = 0.02;           % Residual liquid saturation (-)
rock.Sgr                = 0.001;          % Residual gas saturation (-)
rock.Pb                 = 5.0e+3;         % Gas-entry pressure (Pa)
rock.lambda             = 3.0;            % Curve-fitting parameter (-)
rock.liqRelPermeability = 'BrooksCorey';  % Liquid relative permeability (-)
rock.gasRelPermeability = 'BrooksCorey';  % Gas relative permeability (-)
rock.capillaryPressure  = 'BrooksCorey';  % Saturation degree function (-)
% Set materials to model
mdl.setMaterial(rock, water, gas);
```

Listing S23: Material

In Fig. S8, the initial and boundary data are specified in terms of gas pressure p_g and liquid saturation S_l. However, the formulation implemented uses the pore pressures p_l and p_g as primary variables. Then, to impose a target saturation $\bar{S}_l$, we use the capillary pressure definition, and the liquid saturation law, by setting $p_l = p_g - p_c(\bar{S}_l)$.

```
%% BOUNDARY AND INITIAL CONDITIONS
% Brooks and Corey capillary pressure
pc = @(Sl) (rock.Pb * ((Sl-rock.Slr)/(1.0-rock.Slr-rock.Sgr))^(-1/rock.lambda));
% Set Dirichlet boundary conditions
mdl.setPressureDirichletBCAtBorder('left', 1.0e5 - pc(0.8));
mdl.setGasPressureDirichletBCAtBorder('left', 1.0e5);
% Set initial conditions
mdl.setInitialPressureAtDomain(1.0e5 - pc(0.05));
mdl.setInitialGasPressureAtDomain(1.0e5);
```

Listing S24: Boundary and initial conditions

A transient analysis uses adaptive stepping from 10^{-3} s to 10^{3} s with Newton iterations (Listing S25). Post-processing includes plotting the liquid saturation degree along the bottom border.

```
%% PROCESS
anl = Anl_Transient("Newton");
% Analysis parameters: ti, dt, tf, dtmax, dtmin, adaptStep
anl.setUpTransientSolver(1.0e-3, 1.0e-3, 1.0e3, 5.0, 1.0e-7, true);
% Run analysis
anl.run(mdl);
%% POST-PROCESS
mdl.plotField('CapillaryPressure');
mdl.plotFieldAlongSegment('LiquidSaturation', [0.0, 0.0], [2.6, 0.0], 500, 'x');
```

Listing S25: Process and post-processing

S2.6 Liakopoulos test

This example corresponds to the coupled hydromechanical with two-phase flow verification case discussed in Sect. 6.6 of the main manuscript. Here, the focus is on the script organization required to activate gravity, initialize the stress state, and configure the nonlinear transient solution procedure. The geometry and boundary conditions are shown again in Fig. S9.

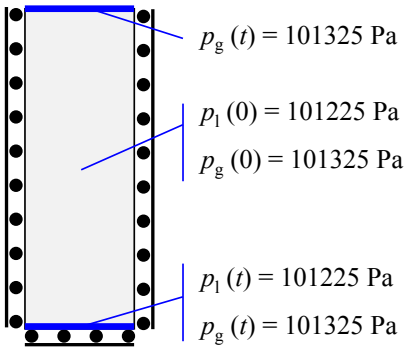

Figure S9: Boundary conditions and geometry of the Liakopoulos test.

The domain is discretized with a structured mesh of 1×100 linear quadrilateral elements (Listing S26). The gravity forces are activated.

```
%% MODEL
mdl = Model_H2M();
mdl.gravityOn = true;
%% MESH
[node, elem] = regularMesh(0.1, 1.0, 1, 100);
mdl.setMesh(node, elem);
```

Listing S26: Liakopoulos problem: model and mesh definition

Listing S27 defines water as a nearly incompressible fluid and gas as an ideal gas. The porous medium utilizes Liakopoulos' empirical capillary-pressure curve and liquid relative permeability; the gas relative permeability follows the Brooks–Corey model.

```
%% MATERIALS
% Create fluids
water = Fluid('water');
water.K = 1.0e+25;      % Compressibility/Bulk modulus (1/Pa)
gas = IdealGas('gas');
gas.mu = 1.8e-5;        % Viscosity (Pa*s)
gas.M  = 0.028949;      % Molar mass (kg/mol)
gas.T  = 300;           % Temperature (K)
% Create the porous media
rock = PorousMedia('rock');
rock.K                  = 4.5e-13;        % Intrinsic permeability (m2)
rock.phi                = 0.2975;         % Porosity
rock.Ks                 = 1.0e+25;        % Solid bulk modulus (Pa)
rock.Slr                = 0.2;            % Residual liquid saturation
rock.lambda             = 3.0;            % Curve-fitting parameter
rock.Young              = 1.3e+6;         % Young modulus (Pa)
rock.nu                 = 0.4;            % Poisson ratio
rock.rho                = 2000.0;         % Density (kg/m3)
rock.liqRelPermeability = 'Liakopoulos';  % Liquid relative permeability
rock.gasRelPermeability = 'BrooksCorey';  % Gas relative permeability
rock.capillaryPressure  = 'Liakopoulos';  % Saturation degree function
rock.setMinGasRelPermeability(1.0e-4);    % Minimum relative permeability
% Set materials to model
mdl.setMaterial(rock, water, gas);
```

Listing S27: Liakopoulos problem: material definition

Stress initialization requires calling `preComputations()` before assigning initial stresses (Listing S28). This function is responsible for initializing several attributes of the model, including the element objects and their integration points.

```
%% BOUNDARY AND INITIAL CONDITIONS
% Displacements
mdl.setDisplacementDirichletBCAtBorder('bottom', [NaN, 0.0]);
mdl.setDisplacementDirichletBCAtBorder('left',   [0.0, NaN]);
mdl.setDisplacementDirichletBCAtBorder('right',  [0.0, NaN]);
% Liquid pressure
mdl.setPressureDirichletBCAtBorder('bottom', 101225.0);     % (Pa)
mdl.setInitialPressureAtDomain(101225.0);                   % (Pa)
% Gas pressure
mdl.setGasPressureDirichletBCAtBorder('top',    101325.0);  % (Pa)
mdl.setGasPressureDirichletBCAtBorder('bottom', 101325.0);  % (Pa)
mdl.setInitialGasPressureAtDomain(101325.0);                % (Pa)
% Initialize the stresses
mdl.preComputations();
for el = 1:mdl.nelem
    elem = mdl.element(el).type;
    for i = 1:elem.nIntPoints
        elem.intPoint(i).stressOld = [101325; 101325; 0.0; 0.0]; % (Pa)
    end
end
```

Listing S28: Liakopoulos problem: boundary and initial conditions

A transient analysis is performed using the Newton-Raphson iterative scheme. Post-processing includes plotting profiles of different fields (Listing S29).

```
%% PROCESS
% Run analysis
anl = Anl_Transient("Newton");
anl.setUpTransientSolver(1.0e-2, 1.0e-2, 3.0e2, 60.0, 1.0e-4, true);
anl.setScaleLinearSystem(true);
anl.maxIter = 15;
anl.run(mdl);
%% POST-PROCESS
Xi = [0.05, 0.0]; Xf = [0.05, 1.0];
mdl.plotFieldAlongSegment('LiquidPressure', Xi, Xf, 500, 'x');
mdl.plotFieldAlongSegment('CapillaryPressure', Xi, Xf, 500, 'x');
mdl.plotFieldAlongSegment('GasPressure', Xi, Xf, 500, 'x');
mdl.plotFieldAlongSegment('LiquidSaturation', Xi, Xf, 500, 'x');
mdl.plotFieldAlongSegment('Uy', Xi, Xf, 500, 'x');
```

Listing S29: Liakopoulos problem: analysis and post-processing

References

[1] Christophe Geuzaine and Jean-François Remacle. Gmsh: A 3d finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79(11):1309–1331, 2009. doi:10.1002/nme.2579.

[2] Eduardo A de Souza Neto, Djordje Peric, and David RJ Owen. *Computational Methods for Plasticity: Theory and Applications*. John Wiley & Sons, 2011.

[3] Nori Watanabe, W Wang, J Taron, UJ Görke, and O Kolditz. Lower-dimensional interface elements with local enrichment: application to coupled hydro-mechanical problems in discretely fractured porous media. *International Journal for Numerical Methods in Engineering*, 90(8):1010–1034, 2012.