GeMA
Geo Modelling Analysis

30 ANOS
TECGRAF
PUC-RIO

ERAS

Training Shell: Example 02 Hydraulic Fracturing Propagation in Permeable Porous Media

Multiphysics Modeling and Simulation Group

Tecgraf Institute / PUC-Rio

Training Meeting

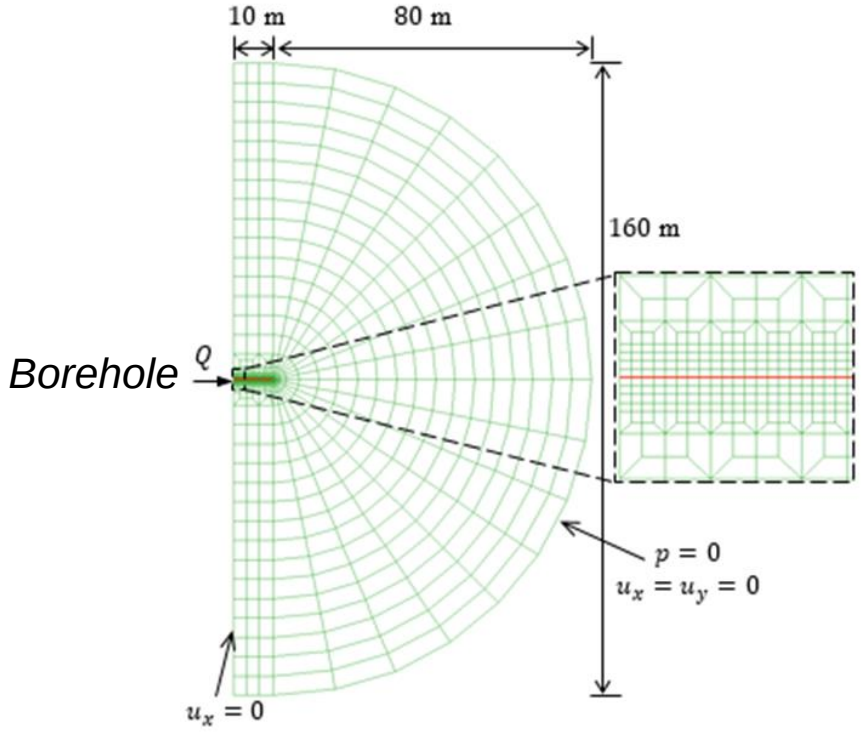
September 2020

Key Example Points

- **<u>This example shows:</u>**
  - How to simulate hydraulic fracturing using interface elements;
  - How to use a three-noded coupled HM interface element and cohesive constitutive model;
  - How to define initially open fractures;
  - How to setup specific options for interface elements;
  - Orchestration script for hydraulic fracturing in impermeable porous media.

Example Overview

- The KGD problem consists of a half-circle plate with one layer of coupled HM interface elements inserted in the middle. The geometry, boundary conditions and finite element mesh of the problem are detailed below.



- The borehole is hydraulically pressurized until a fracture is initiated and propagated through the rock formation.

Simulation file: kVertexGema.lua

GeMA Input Files

*.lua

Main file that stores the analyzes description and loads the remaining model files

*_model.lua

File that gathers the model data (nodes, elements, boundary conditions, etc.).

*_solution.lua

File that gathers information regarding the execution of the analyzes (solver settings, script, orchestration details, etc)

Running File: *.lua

Model description and file information to run the numerical simulation.

```
Simulation
{
  name        = 'kVertexGema',
  description = 'kVertex hydraulic fracturing - Gema'
}

dofile('$SIMULATIONDIR/$SIMULATIONNAME_model.lua')
dofile('$SIMULATIONDIR/$SIMULATIONNAME_solution.lua')
--dofile('$SIMULATIONDIR/$SIMULATIONNAME_result.lua')
```

Model file: State Variables

```
-------------------------------------------------------------
--  State variables
-------------------------------------------------------------
StateVar{id = 'u', dim = 2, description = 'Displacements in
the X and Y directions', unit = 'm', format =
'8.4f',groupName = 'mechanic'}
StateVar{id = 'P', description = 'Pore pressure degree-of-
freedom', unit = 'kPa', format = '8.4f', groupName =
'hydraulic'}
```

➢ Variable definition for the problem:

- For a coupled hydromechanical analysis, displacement and pore pressure state variables are required.
- ***id*** refers to the names of each state variable (u = displacement; P = pore pressure);
- ***dim*** = 2 refers to a two-dimensional (2D) problem;
- The ***description*** field is freely available to the user and serves to give a brief information about the variable mentioned in ***id.***
- ***unit*** provides the unit of the state variable, while format indicates the format (precision) in which this variable will be printed;
- ***groupName*** classifies the physics associated with the model you are working on (‘mechanic’, ‘hydraulic’, etc).
- The HM coupled physics for continuum and interface elements require the same state variables.

Model File: Hydromechanical Properties

```
------------------------------------------------------------
--  Cell properties
------------------------------------------------------------
-- hydro-mechanical properties
PropertySet
{
  id          = 'HMProp',
  typeName    = 'GemaPropertySet',
  description = 'Material properties',
  properties  = {
    {id = 'E',   description = 'Elasticity modulus',  unit = 'kPa'},
    {id = 'nu',  description = 'Poisson ratio',  unit = ''},
    {id = 'K',  description = 'Hydraulic permeability',  unit = 'm/s'},
    {id = 'Kww',  description = 'Bulk modulus of water',  unit = 'kPa'},
    {id = 'Kss',  description = 'Bulk modulus of solid',  unit = 'kPa'},
    {id = 'gw',  description = 'Specific weight of water', unit = 'kN/m3'},
    {id = 'Pht',  description = 'Porosity', unit = ''},
    {id = 'Kn',   description = 'Normal elastic stiffness',  unit = 'kPa/m'},
    {id = 'Ks',  description = 'Shear elastic stiffness',  unit = 'kPa/m'},
              {id = 'Sn',  description = 'Normal traction',  unit = 'kPa'},
              {id = 'Ts',  description = 'Shear traction',  unit = 'kPa'},
              {id = 'ed',  description = 'Equivalent separation',  unit = 'm'},
              {id = 'Gap', description = 'Initial gap opening',  unit = 'm'},
              {id = 'Ufw', description = 'Dynamic fluid viscosity',  unit = 'kPa*s'},
              {id = 'Lkt', description = 'Leakoff at top',  unit = 'm/(kPa*s)'},
              {id = 'Lkb', description = 'Leakoff at bottom',  unit = 'm/(kPa*s)'},
               {id = 'damIni',  description = 'Damage initiation criterion', constMap =
constants.MechanicalFemPhysics.damageInitiation},
              {id = 'material',  description = 'Mechanical material type', constMap =
constants.CoupledHMFemPhysics.materialModels}
  },
  values = {
           {E = 1.7e+07, nu = 0.2,  K = 1e-07,Kww = 1e+16, Kss =3.778e+07, gw = 10.0,
Pht = 0.2,  material= 'poroElastic'},
           {Kn = 1.7e+11, Ks = 1.7e+11, Sn = 1250.0, Ts = 1250.0, ed = 1.92e-4,
damIni='maxS', Gap = 1.0e-5, Ufw = 1.0e-7,  Lkt =1.0, Lkb = 1.0, material=
'cohesiveLinearSoftening'},
  }
}
```

Hydromechanical properties for continuum elements

Hydromechanical properties for interface elements

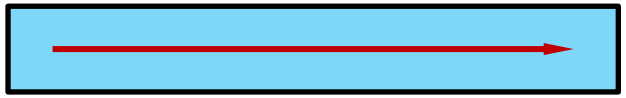
- The ***material*** string defines the cohesive model: cohesive linear softening or cohesive exponential softening. The material must be in agreement with the defined parameters.

Model File: Section Properties

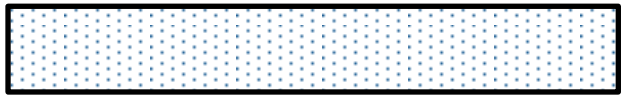
```
-- Section properties
PropertySet
{
  id          = 'SecProp',
  typeName    = 'GemaPropertySet',
  description = 'Section properties',
  properties  = {
    {id = 'h',  description = 'Element thickness', unit
= 'm'},
    {id = 'Iopen', description = 'Intially open
element', unit = ''},
  },
  values = {
    {h =  1.0, Iopen = 0}, -- Iopen = 0 close fracture
        {h =  1.0, Iopen = 1}, -- Iopen = 1 open
fracture
  }
}
```

- Section Property Definition:

- For this problem, ***h*** refers to the element thickness, while ***Iopen*** defines the initial fracture condition (open or closed).



Iopen=1: Initially open fracture allows longitudinal fluid



Iopen=0: No flow inside closed fracture

Model File: Nodes

```
-----------------------------------------
Mesh definition
-----------------------------------------
-- Nodes
local mesh_nodes = {
--  X,      Y
{           0.,           30.},
{0.0250000004,           30.},
{0.0502953157,           30.},
{0.0758894384,           30.},
{ 0.101785891,           30.},
{ 0.127988249,           30.},
{ 0.154500142,           30.},
{ 0.181325197,           30.},
{ 0.208467126,           30.},
{ 0.235929683,           30.},
{ 0.263716638,           30.},
{ 0.291831821,           30.},
{ 0.320279151,           30.},
{ 0.349062502,           30.},
{ 0.378185838,           30.},
{ 0.407653242,           30.},
{ 0.437468708,           30.},
{ 0.467636377,           30.},
{ 0.498160392,           30.},
                 ....
```

- Definition of Mesh Information:

local ***mesh_nodes*** = { } is the LUA table that gathers the {X, Y} coordinates of the model nodes. The table definition syntax in the must be followed:

```
mesh_nodes location = {
        {X1, Y1},
        {X2, Y2},
        {X3, Y3},
        {X4, Y4},
  }
```

- Note carefully that nodal coordinates must be defined between commas, including when moving from one line to another.

Model File: Elements

```
-- coupled HM continuum element
local mesh_quad4_elements = {
{ 3720, 3719, 3551, 3552 },
{ 3889, 3891, 3768, 3766 },
{ 3903, 3902, 3480, 3481 },
{ 1502, 1505, 3429, 3430 },
{ 1501, 1506, 3681, 1504 },
{ 3402, 3429, 1505, 3428 },
{ 3426, 1506, 1501, 3425 },
{  864, 1501, 1504,  752 },
        .....
```

Table with mesh incidence for Quad4 element

```
-- Border edge group
local mesh_edges = {
{id = 'topEdge',   cellList ={
   {  30,1},
   {  31,1},
   {1570,1},
   {1571,1},
   {1576,1},
   {1667,1},
   {1675,1},
      ...
```

Top elements from the model where load boundary conditions will be further applied.

```
id = 'topEdge', cellList = {
      {element, face},
                }
```

```
-- element definition
local mesh_elements = {
  {cellType = 'quad4',    cellList =
mesh_quad4_elements, HMProp=1, SecProp =
1},
  {cellType = 'int2dl6',  cellList =
mesh_cze2d4p_close, HMProp=2, SecProp =
1},
  {cellType = 'int2dl6',  cellList =
mesh_cze2d4p_open, HMProp=2, SecProp = 2},
}
```

Definition of element type, where 'int2dl6' refers to the interface elements.

Model File: Mesh

```
-- mesh definition
Mesh
{
  -- General mesh attributes
  id          = 'mesh',
  typeName    = 'GemaMesh.elem',
  description = 'Plate mesh discretization',

  -- Mesh dimensions
  coordinateDim  = 2,

  -- State vars stored in this mesh (per node)
  stateVars = {'u', 'P'},

  -- Mesh node coordinates
  nodeData = mesh_nodes,

  -- Element data
  cellProperties = {'HMProp', 'SecProp'},
  cellData = mesh_elements,

  -- Integration Rules
  elementRules = {
    {quad4 = 2, int2dl6 = 'N2' },
  },

  -- Boundary data
  boundaryEdgeData = mesh_edges,
}
```

➢ Loading Mesh Information:

- ***id*** is the identifier containing the mesh name. ***typeName*** must have the syntax respected by the user. The ***description*** field is free to be filled in by the user;
- ***coordinateDim*** indicates the size of the finite element mesh. It must accompany the same dimension previously defined for the model;
- ***stateVars*** stores the displacement and pore pressure variables per node in the defined mesh. ***nodeData*** loads the information from the model nodes;
- ***cellProperties*** and ***cellData*** load the information with respect to the previously defined solid elements.
- ***elementRules*** = { } is the LUA table that specifies the type of numerical integration to be performed during finite element analysis. It is necessary to assign the number of integration points used by the element.

Model File: Boundary Conditions

```
-----------------------------------------------------
--  Boundary conditions
-----------------------------------------------------
-- pressure BC condition
BoundaryCondition {
  id   = 'bpd',
  type = 'pressure load',
  mesh = 'mesh',
  properties  = {
    {id = 'pl',  description = 'distribuited pressure load aplied
at the border elements', unit = 'kPa'},
  },
  edgeValues = {
    {'topEdge',  3700.0},   -- Pressure on the border 'edge_1'
 }
}
```

Distributed pressure load applied at the border elements, according to 'topEdge' element list defined previously;

Model File: Boundary Conditions (cont.)

```
-- Fixed displacement BC
BoundaryCondition {
  id   = 'bc1',
  type = 'node displacements',
  mesh = 'mesh',

  properties  = {
    {id = 'ux',  description = 'Fixed node displacement in
the X direction', unit = 'm', defVal = -9999},
    {id = 'uy',  description = 'Fixed node displacement in
the Y direction', unit = 'm', defVal = -9999},
  },

  nodeValues = {
    -- border, ux,  uy
  { 3402, 0.0, nil },
  { 3403, 0.0, nil },
  { 3404, nil, 0.0 },
  { 3407, 0.0, nil },
  { 3429, 0.0, nil },
  { 3430, 0.0, nil },
        ....
```

'0' refers to a fixed displacement at the corresponding direction, while 'nil' refers to free displacement at the corresponding direction.

Model File: Boundary Conditions (cont.)

```
-- Fluid flow BC
 BoundaryCondition {
   id   = 'bc2',
   type = 'node pore flow',
   mesh = 'mesh',

   properties  = {
   {id = 'qw',  description = 'Nodal pore flow', unit =
'm3/s', defVal = -9999, functions = true},
   },

   nodeValues = {
  -- node,    -qw (-) for injection and (+) for depletion
    {4209,    'flow_f'},
}
 }
```

'0' refers to a fixed displacement at the corresponding direction, while '`nil`' refers to free displacement at the corresponding direction.

Flow function

```
NodeFunction { id = 'flow_f',
      parameters = { {src = 'time', unit = 's'} },
      method = function(t)
      t1 = t - 1
      local flow = -5E-4
         if t1 < 0.1 then
           flow = flow*(t1/0.1)
      end
--io.print(tr('Flow = %1 s') :num(flow))
   return flow
end
   }
```

Model File: Boundary Conditions (cont.)

```
-- fixed node pore pressure
 BoundaryCondition {
   id   = 'bc3',
   type = 'node pore pressure',
   mesh = 'mesh',

  properties  = {
    {id = 'P',  description = 'Fixed node pore pressure', unit =
'kPa', defVal = -9999},
  },

   nodeValues = {
    -- node,   P
         {  1,   0.0}, -- nodes with fixed pore pressure
   }
 }
```

'0' refers to a node with fixed pore pressure.

Solution File: Numerical Solvers

```
-----------------------------------------------------
--  Numerical Solvers
-----------------------------------------------------

NumericalSolver {
  id          = 'solver',
  typeName    = 'ArmadilloSolver',
  description = 'Simple matrix solver',
  sparse = true,
}
```

- Solver Information:

- ***NumericalSolver*** = { } is the LUA structure that defines which linear solver will be used to solve the [K] {u} = {f} system. The string typeName = '*ArmadilloSolver*' defines that the Armadillo direct solver plugin will be used.

Solution File: Physical Methods

```
------------------------------------------------------------
--  Physical Methods
------------------------------------------------------------
PhysicalMethod {
  id       = 'Rock',
  typeName = 'CoupledHMFemPhysics.PlaneStrain',
  type     = 'fem',
  mesh = 'mesh',
  boundaryConditions = {'bpd', 'bc1'},
  ruleSet  = 1,
  materials = {'poroElastic'},
  isoParametric = true,
}

PhysicalMethod {
  id       = 'HMCohesiveGeo',
  typeName = 'CoupledHMFemPhysics.Interface',
  type     = 'fem',
  mesh = 'mesh',
  permeabilityUpdate = true,
  materials = {'cohesiveLinearSoftening'},
  isoParametric = false,
  ruleSet = 1,
}

PhysicalMethod {
  id       = 'HMCohesive',
  typeName = 'CoupledHMFemPhysics.Interface',
  type     = 'fem',
  mesh = 'mesh',
  permeabilityUpdate = true,
  materials = {'cohesiveLinearSoftening'},
  isoParametric = false,
  ruleSet = 1,
  boundaryConditions = {'bc2'},
}
```

➢ Definition of Physics Information:

- ***Physical Method*** = { } is the LUA structure that defines the basic information regarding the physics of the problem to be carried out numerically. Again it contains an ID that can be freely defined by the user.

- The **typeName** variable accesses the physical type (“CoupledHMFemPhysics.PlaneStrain” or “CoupledHMFemPhysics.Interface”) of GeMA. There are countless physics that can be accessed, see detailed framework manual to properly access the desired physics. The type variable must also be kept as “**fem**”.

- ***mesh, materials, boundaryConditions*** and ***ruleSet*** must follow the information detailed above.

Solution File: Process execution script

```
-------------------------------------------------------------
--  Process execution script
-------------------------------------------------------------
local solverOptionsGeo = {
  type                = 'transient nonlinear',
  timeMax             = 1.0,
  timeInitIncrement   = 1.0,
  timeMinIncrement    = 0.01,
  timeMaxIncrement    = 1.0,
  iterationsMax       = 15,
  integrationScheme   = 'implicit',
  tolerance           = {mechanic =1.000E-05,hydraulic =1e-5,},
}

local solverOptions = {
  type                    = 'transient nonlinear',
  timeMax             = 10,
  iterationsMax       = 100,
  timeInitIncrement   = 0.001,
  timeMinIncrement    = 0.000001,
  timeMaxIncrement    = 0.1,
  attemptMax          = 10,
  iterationsMax       = 20,
  tolerance           ={mechanic =1.0E-05,hydraulic =1.0e-5,},
}
```

Solver configurations for geostatic step

Solver configurations for the following simulation steps

Solution File: Initial Conditions

```
----------------------------------------
-- Initial conditions
----------------------------------------
function initialCondition()
  -- Get the needed accessors.
  local mesh = modelData:mesh('mesh')
  -- Accessor for the old stress state
  local accS = mesh:gaussAttributeAccessor('S',1, true)
  local accAf = mesh:gaussAttributeAccessor('Afo',0, true)
  -- Node coordinate accessor
  local coordAc = mesh:nodeCoordAccessor()

  -- Integration rules
  local quadIr = mesh:elementIntegrationRule('quad4', 1)
  local intIr = mesh:elementIntegrationRule('int2dl6', 1)
  local pS1 = modelData:propertySet('HMProp')
  assert(type(pS1) == 'propertySet')

  knAc = pS1:propertyAccessor('Kn', 'kPa/m')
  assert(type(knAc) == 'valueAccessor')

  local Kn=knAc:value(2)
  -- Initial stress for each element
  for i=1, mesh:numCells() do
    local e = mesh:cell(i)  -- element
-- solid element
    if (e:type() == 'quad4') then
      -- for each integration point
      for j=1, quadIr:numPoints() do
        -- fills the insitu stress
        --local v = {0.0, 0, 0, 0.0}
        local v = {-3700, -3700, -3700, 0.0}
        accS:setValue(e, j, v)
      end
    end
    -- interface element
    if (e:type() == 'int2dl6') then
      -- elemental node coordinates
      local Xnode = e:nodeMatrix(coordAc, true, 'vertices')
      assert(Xnode)
      -- get fault angle in radians
      -- for each integration point
      for j=1, intIr:numPoints() do
        -- fill stress acording
        local Sv = -3700.0
        local Sh = 0
        -- stresses on the fracture plane
        local Sn  = Sv
        local Tau = Sh
        -- fills the insitu stress vector
        local v = {0, Sn, 0, Tau}
        accS:setValue(e, j, v)
        -- fills the initial aperture
        --local Ai = Sn/Kn
        --accAf:setValue(e, j, Ai)
      end
    end
  end
end
```

- The process execution script for initial conditions follows a similar template from previous examples. Please refer to GeMA manual.

Solution File: Process Script

Set the corresponding accessors

```
function ProcessScript()
   -- Gets mesh accessor
  local mesh = modelData:mesh('mesh')
  local accU = assert(mesh:nodeValueAccessor('u', 'mm'))   -- Get displacement values in mm
  local accP = assert(mesh:nodeValueAccessor('P', 'MPa'))  -- Get Pressure values in MPa
-------------------------------------------------------------------------------------------
-- Initial condition: Geostatic
-------------------------------------------------------------------------------------------
print(' Initializing stress state.....\n')
initialCondition()
```

Call the initialCondition() function defined previously;

Solution File: Process Script (cont.)

```
--================================
 -- Initialization
 --================================
  local solver = fem.init({'Rock','HMCohesiveGeo'}, 'solver',
solverOptionsGeo)
   local resFile =
io.open(translatePath('$SIMULATIONDIR/out/$SIMULATIONNAME.sim')
, "w+")


  -- Saves initial state after the Geostatic step
  local dt = solverOptionsGeo.timeInitIncrement
  dt=fem.step(solver, dt)

--save Displacement and pressure
  resFile:write("Time", " ","P(MPa)", " ","Opening(mm)"\n")
  local Pore = accP:value(4209)
  resFile:write(0, " ", Pore, " ", 0.0, "\n")

  print('End Geostatic Step')
  print('-----------------------------------------')
```

Create *.sim file for results monitoring

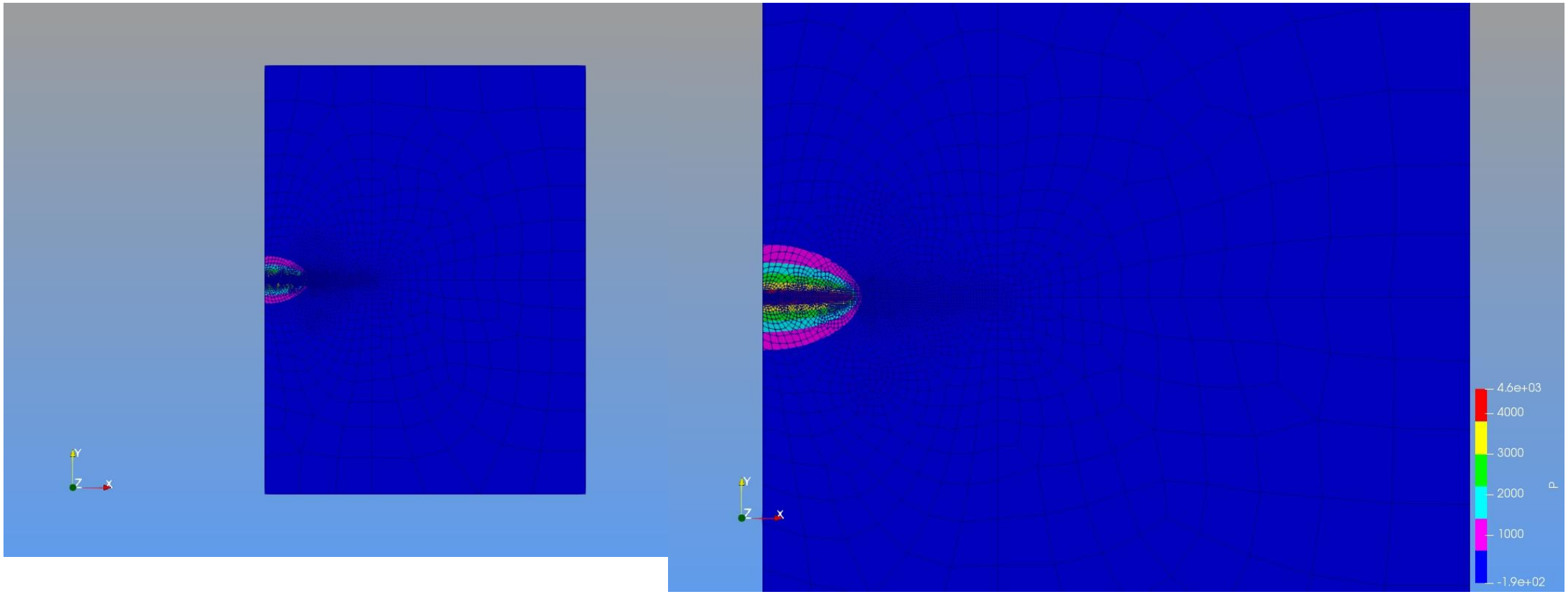
```
 --================================
  -- Hydraulic fracturing
  --================================
  local solver = fem.init({'Rock','HMCohesive'},
'solver', solverOptions)
  local dt = solverOptions.timeInitIncrement
  local FinalTime = solverOptions.timeMax
  local Time      = dt
  local LastStep  = false
  local countP = 0
  --PostRest (countP, 0.0, wfileRes)
  while (Time <= FinalTime) do
          io.print(tr('Time Step = %1 s') :num(Time))
          dt = fem.step(solver, dt)
          countP = countP + 1
          -- save results
          io.addResultToMeshFile(file, Time)
     local Pore = accP:value(4209)/1000.0
     local Ut = accU:value(3403)*1000.0
     local Ub = accU:value(4071)*1000.0
     resFile:write(Time, " ", Pore, " ", Ut(2)-Ub(2),
"\n")
     resFile:flush()
```

Space

resFile:flush () updates the *.sim file with the corresponding results

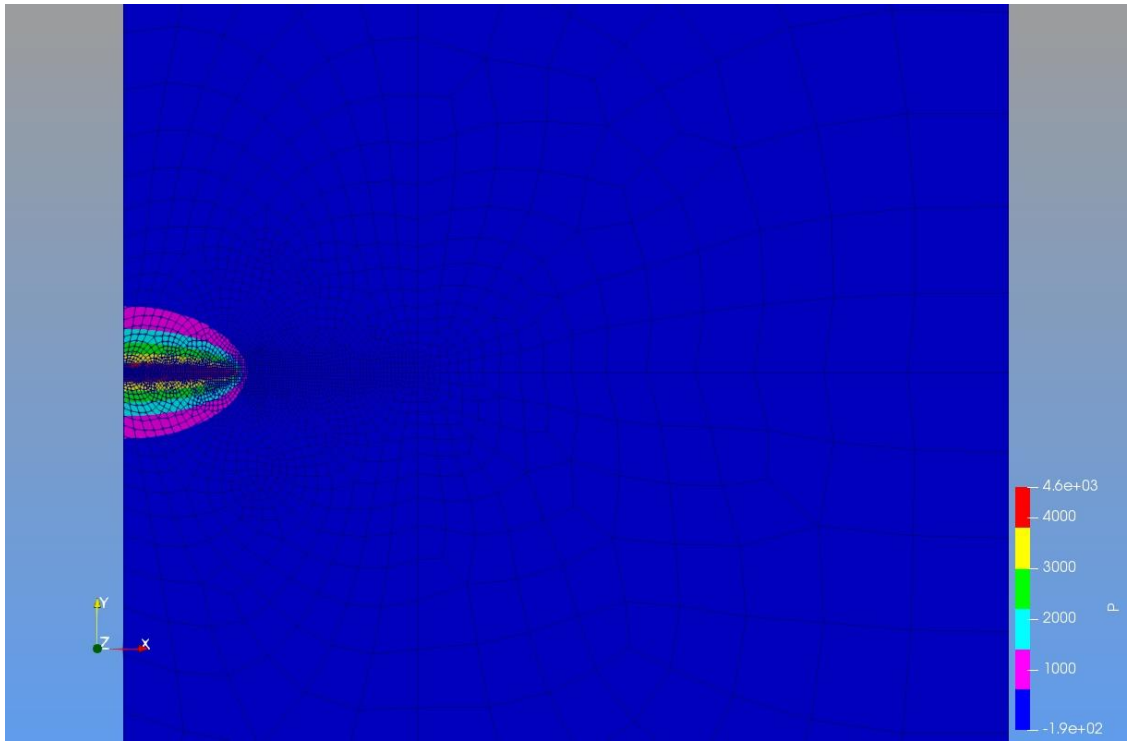
- Two physics objects are needed, one for the continuum and another for interface elements

Results – Pore Pressure



Pore Pressure (kPa)

Results



Pore Pressure (kPa)