 Ana Ortega, Luis Pérez Tato

Tutorial 3: Properties of a reinforced concrete section obtained from a fiber-section model

This tutorial deals with a simple RC section $0.40 \times 0.60\ m^2$ subjected to an uniaxial bending $M_y = 300\ Nm$. The section is discretized in 20 cells (concrete fibers). Two layers of reinforcing bars are defined, 6Φ24 at the bottom face and 4Φ12 at the top.

The aim is to obtain some geometrical and mechanical properties of the RC section as well as its deformation under load.

Import modules. Firstly, we execute some `import` statements, so that the code in our script gains access to the code in the imported modules.

```
1 import math
2 import xc_base
3 import geom
4 import xc
5 from solution import predefined_solutions
6 from model import predefined_spaces
7 from materials.ec2 import EC2_materials
8 from materials.sections.fiber_section import fiber_sets
9 from materials.sections.fiber_section import
      plot_fiber_section
```

Listing 1: Imported modules.

The `math` module (line 15) provides access to several mathematical functions (`floor, exp, sqrt, cos, sin` ...).

The following three lines correspond with three main modules of XC :

- `xc_base`: includes the basic functions for the Python interface: assign and retrieve properties stored in the C++ classes (see example `test_evalPy.py`) and execute Python scripts (see example `test_execPy.py`).
- `geom`: handles entities related to geometry, like points, lines, polylines, planes, polygons, circles, coordinate systems, grids, vectors, matrices, rotations, translations, ...
- `xc`: this module provides access to the finite element classes and functions: mesh generation, element type and material definition, analysis, ...

Import statements in lines 5-6 have to do with the following modules:

- `predefined_solutions`: provides access to several solution procedures for which a set of properties have been predefined (solution algorithm, integrator, DOF-numberer, ...).
- `predefined_spaces`: this module is intended to set the dimension of the space and the number of nodal DOF, as well as to introduce constraints to them.

Finally, the following modules are imported in lines 7 to 9:

- `EC2_materials` is the module where concrete (C12-15 to C90-105) and steel (S400 to S600) material properties are defined according to Eurocode-2 criteria.
- Module `fiber_sets` provides access to the classes that handle fiber-sections of reinforced concrete.
- `plot_fiber_section` module is imported to plot the results obtained.

Definition of parameters. XC allows full parametric models, that's to say, the definition of geometry, material, loads, ..., can be based on properties that, if your data change, the problem is recalculated accordingly.

Lines 10 to 20 set the value of the parameters which will be used later during the model generation.

```
10 width=0.4      #width (cross-section coordinate Y)
11 depth=0.6      #depth (cross-section coordinate Z)
12 cover=0.04      #cover
13 A_s=2712e-6     #area of bottom reinforcement layer (6 fi 24)
14 A_sp=452e-6     #area of top reinforcement layer (4 fi 12)
15 M_y=-300e3       #bending moment [Nm]
16 nDivIJ= 20  #number of cells (fibers) in the IJ direction (
       cross-section coordinate Y)
17 nDivJK= 20  #number of cells (fibers) in the JK direction (
       cross-section coordinate Z)
18 areaFi24=math.pi*(24e-3)**2/4.0
19 areaFi12=math.pi*(12e-3)**2/4.0
20 l= 1e-7      # Distance between nodes
```

Listing 2: Parameters.

Finite element problem. The type of problem defined is StructuralMechanics3D (line 24), that's to say, nodes are defined by three coordinates (x,y,z) with six degrees of freedom (u_x, u_y, u_z, θ_x, θ_y, θ_z). This function takes as argument the handler of nodes, that is retrieved from preprocessor in line 23.

```
21 feProblem=xc.FEProblem()
22 preprocessor=feProblem.getPreprocessor
23 nodes= preprocessor.getNodeHandler
24 modelSpace= predefined_spaces.StructuralMechanics3D(nodes)
```

Listing 3: Model definition.

Definition of nodes. Sentences in lines 16-17 place two nodes, almost coincident at the origin of the global coordinate system, defined by its coordinates (x,y,z).

```
25 nodA= nodes.newNodeXYZ(1.0,0.0,0.0)
26 nodB= nodes.newNodeXYZ(1.0+l,0.0,0.0)
```

Listing 4: Nodes.

Definition of material The materials C30-37 (concrete f_{ck}=30 MPa) and S450C (reinforcing steel f_{yk}=450 MPa) are selected from module `EC2_materials` to make up the RC section. Their respective stress-strain diagrams are called in lines 29-30.

→ Find out more about materials in XC

```
27 concrete=EC2_materials.C30
28 rfSteel=EC2_materials.S450C
29 steelDiagram= rfSteel.defDiagK(preprocessor)
30 concrDiagram=concrete.defDiagK(preprocessor)
```

Listing 5: Material definition.

Geometry of the RC section In this piece of code, the geometry of the cross-section is defined and concrete material assigned to it.

In our case, the section is simply a rectangle, that is defined in local axis Y (parallel to with) and Z (parallel to height) with origin in the center of the polygon. The number of divisions declared (lines 36-37) corresponds to the matrix of fibers that discretize the section, where nDivIJ and nDivJK are, respectively, the number of elements y Y-axis and X-axis directions. The material to make up the concrete fibers is assigned in line 35.

```
31 geomSectFibers= preprocessor.getMaterialHandler.
       newSectionGeometry("geomSectFibers")
32 y1= width/2.0
33 z1= depth/2.0
34 regions= geomSectFibers.getRegions
35 concrSect= regions.newQuadRegion(concrete.nmbDiagK)
36 concrSect.nDivIJ= nDivIJ
37 concrSect.nDivJK= nDivJK
38 concrSect.pMin= geom.Pos2d(-y1,-z1)
39 concrSect.pMax= geom.Pos2d(+y1,+z1)
```

Listing 6: Section geometry.

Layers of reinforcement bars We add to the cross-section model the fibers that match the reinforcing bars. Each rebar is modeled with one fiber that is placed in its exact position refered to the local axis of the section. The way it is done is by defining two layers of rebars, one bottom (6Φ24) and one top (4Φ12). Each layer take as parameters the number of bars, their diameter and the local (y,z) center coordinates of the start and end bars that define the straight reinforcement layer.

```
40 reinforcement= geomSectFibers.getReinfLayers
41 reinfBottLayer= reinforcement.newStraightReinfLayer(rfSteel.
       nmbDiagK)
42 reinfBottLayer.numReinfBars= 6
43 reinfBottLayer.barArea= areaFi24
44 yBotL=(width-2*cover-0.024)/2.0
45 zBotL=-depth/2.0+cover+0.024/2.0
46 reinfBottLayer.p1= geom.Pos2d(-yBotL,zBotL)
47 reinfBottLayer.p2= geom.Pos2d(yBotL,zBotL)
48 reinfTopLayer= reinforcement.newStraightReinfLayer(rfSteel.
       nmbDiagK)
```

```
49 reinfTopLayer.numReinfBars= 4
50 reinfTopLayer.barArea= areaFi12
51 yTopL=(width-2*cover-0.012)/2.0
52 zTopL=depth/2.0-cover-0.012/2.0
53 reinfTopLayer.p1= geom.Pos2d(-yTopL,zTopL)
54 reinfTopLayer.p2= geom.Pos2d(yTopL,zTopL)
```

Listing 7: Reinforcement.

Fiber-section material In lines 55 to 59 a material of type fiber-section is created and added to the material handler. This material represents the force-deformation relationship of the section, obtained by integration of the stress-strain response for each of the fibers in which the section has been discretized.

```
55 materiales= preprocessor.getMaterialHandler
56 sctFibers= materiales.newMaterial("fiber_section_3d","
       sctFibers")
57 fiberSectionRepr= sctFibers.getFiberSectionRepr()
58 fiberSectionRepr.setGeomNamed("geomSectFibers")
59 sctFibers.setupFibers()
```

Listing 8: Material section.

Definition of elements. One zero-length section element is defined linking the two nodes previously created at the same location. The nodes are conected by the section force-deformation we have just defined (`sctFibers` object).

```
60 elements= preprocessor.getElementHandler
61 elements.defaultMaterial='sctFibers'
62 ele1= elements.newElement("ZeroLengthSection",xc.ID([nodA.tag,
       nodB.tag]))
```

Listing 9: Elements.

Definition of constraints. Lines 63 to 65 introduce single-point boundary constrainsts in both nodes, restraining its six degrees of freedom for the first one and allowing displacement u_x and rotations θ_y and θ_x in the end node.

```
63 constraints= preprocessor.getBoundaryCondHandler
64 modelSpace.fixNode000_000(nodA.tag)
65 modelSpace.fixNodeF00_0FF(nodB.tag)
```

Listing 10: Constraints.

Definition of loads. First, the handlers of loads and load patterns are successively called (lines 66-67). Then, we specify a linear time serie as default and create a new load pattern, giving as arguments its type and an arbitrary name. The bending moment about global Y-axis is applied to the end node (lines 72) by means of the vector defined in line 71, whose components represent the values of external forces and moments $[F_x, F_y, F_z, M_x, M_y, M_z]$ expressed in the global coordinate system.

```
loads= preprocessor.getLoadHandler
lpatt= loads.getLoadPatterns
ts= lpatt.newTimeSeries("constant_ts","ts")
lpatt.currentTimeSeries= "ts"
lp0= lpatt.newLoadPattern("default","0")
pointLoad= xc.Vector([0.0,0.0,0.0,0,M_y,0.0])
lp0.newNodalLoad(nodB.tag,pointLoad)
lpatt.addToDomain("0")
```

Listing 11: Loads.

Obtaining solution. The solution is obtained using the well-known Newton-Raphson method. The shortcut to do that is to call the corresponding method in `predefined_solutions` module, which pre-determine all the components involved in the solution procedure (constraint-handler, numberer, solution algorithm, convergence criterion, integrator, ...)

```
solProc= predefined_solutions.PlainStaticModifiedNewton(
    feProblem, convergenceTestTol= 1e-9)
analOk= solProc.solve()
if(analOk!=0):
    lmsg.error('Failed to solve for: '+lp0.name)
    quit()

nodes.calculateNodalReactions(True,1e-7)
```

Listing 12: Solution.

Review of results. First, we retrieve from the zero-length element the copy of the material fiber-section that is specifically assigned to this element and represents its tensional state.

The following properties are obtained (lines 83-93):

`x`: Neutral axis depth.

`zNA`: Neutral axis plane.

`zCP`: Compression plane.

`zTP`: Tension plane.

`zIFA`: Internal forces axis.

`levArm`: Lever arm.

`effDepth`: Effective depth.

`zEffConcA`: Limit of concrete effective area.

```
sccEl1= ele1.getSection()
fibersSccEl1= sccEl1.getFibers()
RNA= nodA.getReaction[0]
RNB= nodB.getReaction[0]
x= sccEl1.getNeutralAxisDepth()
zNA=sccEl1.getNeutralAxis().getParamB()
slopeBP=sccEl1.getBendingPlaneTrace().getParamA()
```

```
zCP=sccEl1.getCompressedPlaneTrace().getParamB()
zTP=sccEl1.getTensionedPlaneTrace().getParamB()
zIFA=sccEl1.getInternalForcesAxes().getParamB()
levArm=sccEl1.getLeverArmSegment().getLength()
levArm2=sccEl1.getMechanicLeverArm()
effDepth=sccEl1.getEffectiveDepthSegment().getLength()
heffmax_EC2=min(2.5*(depth-effDepth),(depth+x)/3.,depth/2.0)
zEffConcA=sccEl1.getEffectiveConcreteAreaLimitLine(heffmax_EC2
    ).getParamB()
```

Listing 13: Results.

Plot of results. We can plot these results over the section to obtain sketchs like those shown below.

```
fsPlot=plot_fiber_section.fibSectFeaturesToplot(fiberSection=
    sccEl1)
fsPlot.colorNeutralAxis = 'r'
fsPlot.colorBendingPlane='g'
fsPlot.colorCompressionPlane='b'
fsPlot.colorTensionPlane='m'
fsPlot.colorIntForcAxis='c'
fsPlot.colorLeverArm='orange'
fsPlot.colorEffDepth='purple'
fsPlot.colorEffConcrArea='brown'
fsPlot.MaxEffHeight=heffmax_EC2
fsPlot.colorGrossEffConcrAreaContours='m'
fig1,ax2d=fsPlot.generatePlot()
fig1.savefig('fig1.png')
```

Listing 14: Plot.