Getting started with XC. Axial force in a bar subjected to a uniform temperature increase.

This tutorial deals with a steel bar of uniform cross section whose ends are fixed from translational motion. The Young's modulus of steel is $E = 210\ GPa$ and its coefficient of thermal expansion $\alpha = 1.2\text{e}{-5}°C^{-1}$. The area and length of the bar are assumed to be $L = 1\ m$ and $A = 4\text{e}{-4}\ m^2$, respectively. The goal is to find the axial force in the bar subjected to a uniform temperature increase of $\Delta T = 10°C^{-1}$.

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

```python
import xc_base
import geom
import xc
from model import predefined_spaces
from materials import typical_materials
from solution import predefined_solutions
```

Listing 1: Imported modules.

The first 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 4-7 have to do with the following modules:

- `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.
- `typical_materials`: several often-used materials are predefined in this module: elastic uniaxial with or without tension branch, prestressed cable, concrete, steel, ...
- `predefined_solutions`: provides access to several solution procedures for which a set of properties have been predefined. Among these properties are: the solution algorithm (linear, Newton Raphson, Broyden, ...), the integrator, the DOF numberer, the convergence tolerance, the maximum number of iterations, ...

  Some of the predefined solvers are: simple static linear, plain linear Newmark, simple Lagrange static linear, simple transformation static linear, simple Newton-Raphson and modified Newton-Raphson, penalty Newton-Raphson, penalty Newmark Newton-Rapshon and frequency analysis.

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 7 to 11 set the value of the parameters which will be used later during the model generation.

```
 7  L= 1.0 # Bar length (m)
 8  E= 210e9 # Elastic modulus (Pa)
 9  alpha= 1.2e-5 # Thermal expansion coefficient of the steel
10  A= 4e-4 # bar area expressed in square meters
11  AT= 10 # Temperature increment (Celsius degrees)
```

Listing 2: Parameters.

Finite element problem. The type of problem defined is SolidMechanics2D (line 15), that's to say, nodes are defined by two coordinates (x,y) and has two degrees of freedom (u_x, u_y). This function takes as argument the handler of nodes, that is retrived from preprocessor in line 14.

```
12  feProblem= xc.FEProblem()
13  preprocessor=  feProblem.getPreprocessor
14  nodes= preprocessor.getNodeHandler
15  modelSpace= predefined_spaces.SolidMechanics2D(nodes)
```

Listing 3: FE problem type.

Definition of nodes. Sentences in lines 16-17 place a node, defined by its coordinates (x,y), in each extremity of the bar.

```
16  nod1= nodes.newNodeXY(0.0,0.0)
17  nod2= nodes.newNodeXY(L,0.0)
```

Listing 4: Nodes.

Definition of material The material defined in line 18 is an elastic uniaxial material. The method takes as parameters the preprocessor object, a name that we'll use to assign this material to the elements, and its Young's modulus.

```
18  elast= typical_materials.defElasticMaterial(preprocessor, "
        elast",E)
```

Listing 5: Materials.

→ Find out more about materials in XC

Definition of elements. The only element in the model is created and added to the element handler (line 23). Prior to that, the material, element dimension and the tag to start element numbering are set through attributes of the element handler. We use one of the available methods to create elements, that takes as arguments a name pointing to the type of element and the tags of the nodes linked. The library of elements include types: *truss* ,

truss_section , *corot_truss* , *corot_truss_section* , *muelle* , *spring* , *beam2d_02* , *beam2d_03* , *beam2d_04* , *beam3d_01* , *beam3d_02* , *elastic_beam2d* , *elastic_beam3d* , *beam_with_hinges_2d* , *beam_with_hinges_3d* , *nl_beam_column_2d* , *nl_beam_column_3d* , *force_beam_column_2d* , *force_beam_column_3d* , *shell_mitc4* , *shell_nl* , *quad4n* , *tri31* , *brick* , *zero_length* , *zero_length_contact_2d* , *zero_length_contact_3d* , *zero_length_section*.

```
19 elements= preprocessor.getElementHandler
20 elements.defaultMaterial= "elast"
21 elements.dimElem= 2 # Dimension of element space
22 elements.defaultTag= 1 #Tag for the next element.
23 truss= elements.newElement("Truss",xc.ID([nod1.tag,nod2.tag]))
       ;
24 truss.area= A
```

Listing 6: Elements.

→ Find out more about element types in XC

Definition of constraints. Lines 25 to 29 introduce single-point boundary constrainsts in both nodes restraining their two degrees of freedom, adding these components of the model to the constraint handler. The method employed takes as arguments: the tag of the node, the degree of freedom to be limited, and the value assigned to that DOF.

```
25 constraints= preprocessor.getBoundaryCondHandler
26 spc1= constraints.newSPConstraint(nod1.tag,0,0.0)
27 spc2= constraints.newSPConstraint(nod1.tag,1,0.0)
28 spc3= constraints.newSPConstraint(nod2.tag,0,0.0)
29 spc4= constraints.newSPConstraint(nod2.tag,1,0.0)
```

Listing 7: Constraints.

→ Find out more about boundary conditions in XC

Definition of loads. First, the handlers of loads and load patterns are successively called (lines 30-31). 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 effect of the temperature increase is introduced in the element as an imposed deformation and added to the load pattern.

```
30 loadHandler= preprocessor.getLoadHandler
31 lPatterns= loadHandler.getLoadPatterns
32 ts= lPatterns.newTimeSeries("linear_ts","ts")
33 lPatterns.currentTimeSeries= "ts"
34 lp0= lPatterns.newLoadPattern("default","0")
35 eleLoad= lp0.newElementalLoad("truss_temp_load")
36 eleLoad.elementTags= xc.ID([1])
37 eleLoad.eps1= alpha*AT
38 eleLoad.eps2= alpha*AT
39 lPatterns.addToDomain("0")
```

Listing 8: Loads.

→ Find out more about loads and load patterns in XC

Obtaining solution. The solution is obtained by using the algorithm that performs a linear static analysis.

```
analisis= predefined_solutions.simple_static_linear(feProblem)
result= analisis.analyze(1)
```

Listing 9: Solver.

→ Find out more about analysis in XC

Review of results. Finally, we ask the element its axial internal force.

```
elem1= elements.getElement(1)
elem1.getResistingForce()
N= elem1.getN()
print 'N= ', N
```

Listing 10: Results.

The result must be:

$$N = -E \times A \times \alpha \times \Delta T = -10.080\ N(compression)$$