OOFEM Python bindings

Release 1.0

Bořek Patzák

May 03, 2021

CONTENT

CHAPTER

ONE

FIRST STEPS

This section covers the basic usage of oofem python bindings. It covers the installation, consisting of generating the python bindings and demonstrates the basic usage.

1.1 Installation

1.1.1 Prerequisities

The python bindings require pybind11 (https://pybind11.readthedocs.io/en/stable/). Pybind11 requires **python-dev** or **python3-dev** packages as well as **cmake**. The recommended procedure to install pybind11 is to clone pybind11 github repository

```
git clone https://github.com/pybind/pybind11.git
cd pybind11
mkdir build
cd build
cmake ..
make check -j 4
make install
```

1.1.2 Generate binding code

Configure oofem target to build python bindings

```
cd oofem.build
cmake path_to_oofem_git_repository -D USE_PYTHON_BINDINGS=ON
make
```

Building the above project will produce a binary module file that can be imported to Python.

1.2 Basic usage

In the previous section we have built the oofem python module. Assuming that the compiled module is located in the current directory, the following interactive Python session shows how to load and execute the example:

```
$ python3
Python 2.7.10 (default, Aug 22 2015, 20:33:39)
Python 3.6.8 (default, Oct  7 2019, 12:59:55)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import oofempy
>>> dr=oofempy.OOFEMTXTDataReader("patch010.in")
>>> problem=oofempy.InstanciateProblem(dr, oofempy.problemMode.processor, False, None,
↪ False)
>>> problem.init()
>>> problem.solveYourself()
Computing initial guess
StaticStructural :: solveYourselfAt - Solving step 1, metastep 1, (neq = 3)
NRSolver: Iteration ForceError
--------------------------------------------------------------------------------
NRSolver: 0      D_u:  0.000e+00
Checking rules...
EngngModel info: user time consumed by solution step 1: 0.00s
>>> problem.terminateAnalysis()
ANALYSIS FINISHED
Real time consumed: 000h:00m:45s
User time consumed: 000h:00m:00s
```

CHAPTER

TWO

USING OOFEM CLASSES

The binding code is generated for all fundamental classes of OOFEM. This section presents the simple use cases, where selected OOFEM classes are used directly in python code.

OOFEm comes with built in representation of vectors and matrices. The full feature Python interface is provided in terms of all methods, but also the convenience constructors (from any sequence) and usual operators are provided.

```
$ python3
Python 2.7.10 (default, Aug 22 2015, 20:33:39)
Python 3.6.8 (default, Oct  7 2019, 12:59:55)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import oofempy
>>> a = oofempy.FloatArray((1.0, 2.0, 3.0))
>>> b = oofempy.FloatArray((0.0, -1.0, 1.0))
>>> c = a+b
>>> print (c)
<oofempy.FloatArray: {1.000000, 1.000000, 4.000000, }>
>>> c += a
```

The following example illustrates how to use FloatMatrix class to solve simple linear system

```
$ python3
>>> import oofempy
>>> A = oofempy.FloatMatrix(3,3)
>>> A.beUnitMtrx()
>>> A[0,1]=-2.
>>> A[1,2]=3
>>> A.printYourself()
FloatMatrix with dimensions : 3 3
1.000e+00  -2.000e+00   0.000e+00
0.000e+00   1.000e+00   3.000e+00
0.000e+00   0.000e+00   1.000e+00
>>> b=oofempy.FloatArray((-3, 11, 3))
>>> x = oofempy.FloatArray(3)
>>> A.solveForRhs(b, x, False)
>>> print (x)
<oofempy.FloatArray: {1.000000, 2.000000, 3.000000, }>
```

CHAPTER

THREE

CREATING OOFEM MODEL

3.1 Creating model from input file

The problem can be instatiated form a stadard oofem input file, as illustrated form the following example

```
$ python3
Python 2.7.10 (default, Aug 22 2015, 20:33:39)
Python 3.6.8 (default, Oct  7 2019, 12:59:55)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import oofempy
>>> dr=oofempy.OOFEMTXTDataReader("patch010.in")
>>> problem=oofempy.InstanciateProblem(dr, oofempy.problemMode.processor, False, None,
↪ False)
```

After that, the problem and its components can be manipulated from python3

```
>>> problem.init()
>>> problem.solveYourself()
Computing initial guess
StaticStructural :: solveYourselfAt - Solving step 1, metastep 1, (neq = 3)
NRSolver: Iteration ForceError
--------------------------------------------------------------------------------
NRSolver: 0      D_u:  0.000e+00
Checking rules...
EngngModel info: user time consumed by solution step 1: 0.00s
>>> problem.terminateAnalysis()
ANALYSIS FINISHED
Real time consumed: 000h:00m:45s
User time consumed: 000h:00m:00s
```

3.2 Building model from script

It is also possible to create individual problem components directly from Python, as illustrated in the following code

```
import oofempy
import util # some utility functions
problem = oofempy.linearStatic(nSteps=1, outFile="test2.out") # engngModel
domain = oofempy.domain(1, 1, problem, oofempy.domainType._2dBeamMode, tstep_all=True,
↪ dofman_all=True, element_all=True) # domain aka mesh
problem.setDomain(1, domain, True) # associate domain to the problem
```

One can create individual components directly from python. The convenience constructors accepting all keyword-value pairs of component records are provided.

```
# load time function
ltf1 = oofempy.peakFunction(1, domain, t=1, f_t=1)
ltfs = (ltf1, )
# boundary conditions
# loadTimeFunction parameter can be specified as int value or as LoadTimeFunction␣
↪itself (valid for all objects with giveNumber() method)
bc1   = oofempy.boundaryCondition(    1, domain, loadTimeFunction=1,    ␣
↪prescribedValue=0.0)
nLoad = oofempy.nodalLoad(            2, domain, loadTimeFunction=1,    components=(-
↪18.,24.,0.))
bcs = (bc1, nLoad)
```

Next we create nodes and other components. See, that reference to other components can be specified by the component number of by passing the object itself.

```
# nodes
# if one value is passed as parameter where oofem expects array of values, it must be␣
↪passed as tuple or list (see load in n4)
n1 = oofempy.node(1, domain, coords=(0.,  0., 0. ), bc=(1,1,1))
n2 = oofempy.node(2, domain, coords=(2.4, 0., 0. ), bc=(0,0,0), load = (nLoad,))
nodes = (n1, n2)
# material and cross section
mat = oofempy.isoLE(1, domain, d=1., E=30.e6, n=0.2, tAlpha=1.2e-5)
cs  = oofempy.simpleCS(1, domain, area=0.162, Iy=0.0039366, beamShearCoeff=1.e18,␣
↪thick=0.54)
# elements
e1 = oofempy.beam2d(1, domain, nodes=(1,n2),  mat=1,   crossSect=1)
elems = (e1, )
# add eveything to domain
util.setupDomain (domain, nodes, elems, (mat,), (cs,), bcs, ltfs, ())
```

After setting up the problem we can solve it.

```
>>> print("\nSolving problem")
Solving problem
>>> problem.checkProblemConsistency()
>>> problem.init()
>>> problem.postInitialize()
>>> problem.setRenumberFlag()
>>> problem.solveYourself()
Solving ...
EngngModel info: user time consumed by solution step 1: 0.00s
>>> problem.terminateAnalysis()
ANALYSIS FINISHED
Real time consumed: 000h:00m:00s
User time consumed: 000h:00m:00s
```

The more elaborated example can be found in test_2.py (https://github.com/oofem/oofem/blob/master/bindings/python/tests/test_2.py).

CHAPTER

FOUR

EXTENDING OOFEM IN PYTHON

The python interface allows user to define custom derived classes in Python and inject them inside oofem. We first illustate the concept on defining user defined time function (called MyFunc) in Python. The class should be derived from corresponding oofem base class (Function). It is necessary to implement at least methods to evaluate the function value (evaluateAtTime) and its derivatives (evaluateVelocityAtTime and evaluateAccelerationAtTime).

```
import oofempy
class MyFunc(oofempy.Function):
    def __init__(self, num, domain):
        oofempy.Function.__init__(self, num, domain)
    def evaluateAtTime (self, t):
        return 2.5*t
    def evaluateVelocityAtTime(self, t):
        return 0.0;
    def evaluateAccelerationAtTime(self, t):
        return 0.0;
```

After the definition, we can directly use the MyFunc class in the problem setup:

```
problem = oofempy.linearStatic(nSteps=1, outFile="test3.out")
domain = oofempy.domain(1, 1, problem, oofempy.domainType._1dTrussMode, tstep_
↪all=True, dofman_all=True, element_all=True)
problem.setDomain(1, domain, True)
ltf1 = oofempy.peakFunction(1, domain, t=1, f_t=1)
ltf2 = MyFunc(2, domain) # use custom ltf here
ltfs = (ltf1, ltf2)
# boundary conditions
bc1   = oofempy.boundaryCondition(    1, domain, loadTimeFunction=1,
↪prescribedValue=0.0)
n2    = oofempy.nodalLoad(            2, domain, loadTimeFunction=2,    components=(1.
↪,), dofs=(1,))
bcs = (bc1, n2)
#nodes
n1 = oofempy.node(1, domain, coords=(0.,  0., 0. ), bc=(bc1,))
n2 = oofempy.node(2, domain, coords=(2.4, 0., 0. ), load=(n2,))
nodes = (n1, n2)
# material and cross section
mat = oofempy.isoLE(1, domain, d=1., E=30.e6, n=0.2, tAlpha=1.2e-5)
cs  = oofempy.simpleCS(1, domain, area=0.5, Iy=0.0, beamShearCoeff=1.e18, thick=0.5)
# elements
e1 = oofempy.truss1d(1, domain, nodes=(1,n2),  mat=1,   crossSect=1)
elems = (e1,)
# add eveything to domain (resize container first to save some time, but it is not
↪necessary 0 see ltfs)
domain.setup(nodes, elems, (mat,), (cs,), bcs, ltfs, (,))
```

(continues on next page)

(continued from previous page)

```
problem.checkProblemConsistency()
problem.init()
problem.postInitialize()
problem.solveYourself()
problem.terminateAnalysis()
print("\nProblem solved")
```

Follow https://github.com/oofem/oofem/blob/master/bindings/python/tests/test_3.py for a full example.

4.1 Custom element

This section illustrates how to implement custom element in Python. In this case, we are implementing element for structural analysis, so it should be derived from cooresponding oofem base class, which is StructuralElement in this case. There are various options how to implement an element. The simplest option is to directly define how to evaluate element stifness matrix and internal force vector. Additional functuos to determine degrees of freedom required by the element anre also to be provided. Other possibilities exists, for example one can rely on default implementation of StructuralElement to evalaute stifness matrix, but then methods delivering strain-displacement and constituve matrix have to be provided together with element integration rules. For details, please consult oofem manuals.

```
import oofempy
class MyElement(oofempy.StructuralElement):
def __init__(self, num, domain):
    oofempy.StructuralElement.__init__(self, num, domain)
    self.setNumberOfDofManagers(2)
def computeStiffnessMatrix(self, answer, rMode, tStep):
    answer.resize(2,2);
    answer[0,0] = 1.0;
    answer[0,1] = -1.0;
    answer[1,0] = 1.0;
    answer[1,1] = -1.0;
def computeBmatrixAt(self, gp, answer, lowerIndx, upperIndx):
    answer.resize(1,2);
    answer[0,0]=-1.0;
    answer[0,1]=1.0;
def giveInternalForcesVector(self, answer, tStep, useUpdatedGpRecord):
    u = oofempy.FloatArray()
    k = oofempy.FloatMatrix(2,2)
    self.computeVectorOf(oofempy.ValueModeType.VM_Total, tStep, u)
    self.computeStiffnessMatrix(k, oofempy.CharType.StiffnessMatrix, tStep)
    answer.beProductOf (k, u)
def giveNumberOfDofs(self):
    return 2;
def computeNumberOfDofs(self):
    return 2;
def giveDofManDofIDMask(self, inode, answer):
    print ("giveDofManDofIDMask for %d"%(inode,))
    print (answer)
    answer.resize(1)
    answer[0] = oofempy.DofIDItem.D_u
    #answer.pY()
    print(answer)
def giveClassName(self):
    return "MyElement"
```

(continues on next page)

(continued from previous page)

```
def giveInputRecordName(self):
    return "MyElement"
```

The element can again added into domain and used from oofem

```
# nodes
n1 = oofempy.node(1, domain, coords=(0.,  0., 0. ), bc=(bc1,))
n2 = oofempy.node(2, domain, coords=(2.4, 0., 0. ), load=(n2,))
nodes = (n1, n2)
# elements
e1 = MyElement(1, domain) # additinal entries should go to to the custom element␣
↪constructor
e1.setDofManagers((1,2))
ir = oofempy.OOFEMTXTInputRecord()
ir.setRecordString ("nodes 2 1 2 mat 1 crosssect 1")
# pass input record to elem
e1.initializeFrom(ir)
elems = (e1,)
...
```

You can follow https://github.com/oofem/oofem/blob/master/bindings/python/tests/test_3.py for a complete illustration.

4.2 Custom material

This section illustrates how to implement custom constitutive model in Python. First we have to define Python class implementing the model derived from corresponding oofem class. In this case, we are going to implement constututive model for structural analysis, so we derive our class from StructuralMaterial. The presented implementation is mininimalistic one, we overload or define just methods to support 1d stress strain state, by overriding give1dStiffnessMatrix and giveRealStressVector_1d methods, that simply returns constitutive matrix and evaluate 1d stress from given strain. The more elaborate implementation would be necessary to support sevaral stress-strain modes, nonlinear materials (need to create custom status to track history variables). For more details, please refer to oofem programmer's manual. Each matteril model must define its material status contatining its internal state variables. Even if material models does not need to track internal variables, it should provide status. In such case, it is sufficient to create instance of StructuralMaterialStatus. In oofem, the method responsible for status creation is CreateStatus, that would normally be overriden in Python as well. However, doe to the issue in Pybind11 (https://github.com/pybind/pybind11/issues/1962) this is not yet possible. The workaroud is to override giveStatus, as illustrate in the following example.

```
class MyMaterial(oofempy.StructuralMaterial):
  def __init__(self, num, domain):
      oofempy.StructuralMaterial.__init__(self, num, domain)
      self.k = 1.5;
  def giveClassName(self):
      return "MyMaterial"
  def giveInputRecordName(self):
      return "MyElement"
  # Overloading this method is not yet possible in pybind11
  # see https://github.com/pybind/pybind11/issues/1962
  # However a workaround is to override giveStatus, see below
  def CreateStatus(self, gp):
      return oofempy.StructuralMaterialStatus (gp)
  def give1dStressStiffMtrx(self, answer, mode, gp, tStep):
      answer.resize(1,1)
```

(continues on next page)

(continued from previous page)

```
        answer[0,0] = self.k
        return
    def giveRealStressVector_1d (self, answer, gp, reducedStrain, tStep):
        answer.resize(1)
        answer[0] = self.k * reducedStrain[0]
        status = self.giveStatus(gp)
        status.letTempStrainVectorBe(reducedStrain);
        status.letTempStressVectorBe(answer);
        return
    def giveStatus (self, gp):
        print ("getStatus")
        if (gp.giveMaterialStatus() is None):
            print ("getStatus creating")
            status = oofempy.StructuralMaterialStatus (gp)
            gp.setMaterialStatus(status)
        return gp.giveMaterialStatus()
```

You can follow https://github.com/oofem/oofem/blob/master/bindings/python/tests/test_4.py for a complete illustration.

CHAPTER

FIVE

POSTPROCESSING

This section covers the basic postprocessing when using oofem python bindings.

One can always directly access results using oofem API after solving each solution step. Another option consist of using existing or developing custom export module. OOFEM comes with many built-in export modules to produce output in desired format.

In the following sections, we will cover some techniques in more detail.

5.1 VTK postprocessing

This section illustrates how to query the data from oofem in python and use vtk library to visualize the results.

5.1.1 Prerequisites

Here we will use pyvista python module (https://pyvista-doc.readthedocs.io/en/latest/) to interface with Visualization toolkit (VTK). The recommended procedure to install pyvista is to use python package installer:

```
pip install pyvista
```

The approach uses built-in vtkmemory export module. This module allows in memory pythonic access to oofem grid data and variables. First, we use existing model. The model is read from input file and solved:

```
import oofempy
import numpy as np
import pyvista as pv

dr=oofempy.OOFEMTXTDataReader("concrete_3point.in")
problem=oofempy.InstanciateProblem(dr, oofempy.problemMode.processor, False, None,␣
↪False)
problem.init()
problem.solveYourself()
```

Next, we create instance of oofem vtkmemory export module and associate it to our problem. The constructor allows to filter the output to specific solution steps (here we use `tstep_all=True` to match all solution steps) and allows to select variables to export (we request dipslacement vector as primary variables, stress and strain tensors as internal variables and element number as cell variables, see oofem Input manual for details).

Further, the module is initialized and output is prepared.

```
vtkxmlPy = oofempy.vtkmemory(1, problem, domain_all=True, tstep_all=True, dofman_
↪all=True, element_all=True, vars=(1,4), primvars=(1,), cellvars = (47,), stype=1,␣
↪pythonExport=1)
vtkxmlPy.initialize()
vtkxmlPy.doOutput(problem.giveCurrentStep(), False)
```

The vtkmemory module allows in memory access to so called VTKPieces, objects representing piece (subset of mesh) to visualize. VTKPieces contain all needed information and data about piece geometry, connectivity and about exported variables. We start looping over the pieces, request data needed to instanciate pyvista UnstructuredGrid instance and attach exported variables to it.

```
for p in vtkxmlPy.getVTKPieces():
#p = vtkxmlPy.getVTKPieces()[0]
    print ("Piece:", p)
    print(p.getVertices())
    print(p.getCellConnectivity())
    print(p.getCellTypes(vtkxmlPy))
    disp = p.getPrimaryVertexValues(oofempy.UnknownType.DisplacementVector)
    sig = p.getInternalVertexValues(oofempy.InternalStateType.IST_StressTensor)
    sigx = sig[:, 0]

    grid = pv.UnstructuredGrid(p.getCellConnectivity(), p.getCellTypes(vtkxmlPy), p.
↪getVertices())
    grid.point_arrays['Sigma_xx'] = sigx
    grid['Disp'] = disp
    print(grid.active_vectors)
    warped = grid.warp_by_vector('Disp', factor=1000.)
    p = pv.Plotter()
    p.add_mesh(warped, scalars='Sigma_xx')
    p.add_mesh(grid, style='wireframe', color='black')
    p.set_viewup((0,1,0))
    p.show()
problem.terminateAnalysis()
```

You can follow https://github.com/oofem/oofem/blob/master/bindings/python/examples/vtkdemo.py for a full example.

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search