Creating models using Python in ABAQUS

General procedure

You can record Python(version 2.7.3)-commands of Abaqus CAE and the Abaqus Viewer in three ways:

- *.rpy* file (all commands,in work directory)
- *.jnl* file (model commands along the .cae file)
- File/Macro Manager in Abaqus CAE (set *Work* directory) to write into *abaqusMacros.py* (all commands, only option in *Student Edition*)

Run Python scripts in Abaqus:

- In Abaqus Command Line (bottom CAE window: >>>)
  → whole files with `execfile('mymodel.py')`
- with *File/Run Script* in Abaqus CAE/Viewer
- in the windows command line / shell:

```
abaqus cae script=mymodel.py    (opens ABAQUS CAE)
abaqus cae noGUI=mymodel.py     (in the background)
```

Building a scripted model

Building a Python script *model.py* from an existing Abaqus CAE model:
Use header from below to start your script. Use a meaningful name, not *model.py*, and run the script in Abaqus (e.g. by `execfile('model.py')`)
Build the script step by step (feature by feature) and copy the new lines into your *model.py* file:
Click arround in ABAQUS CAE→ Get commands from *.jnl* / *.rpy* /*macro.py* file → copy relevant commands into *model.py*

Clean up the code that you copy to your file using shorter variables for the model (m), sketches (s, s1, s2, …), etc. and replace numbers by your model parameters, e.g. your width *b* and height *h*:

.jnl, .rpy or macro file:

```
mdb.models['Model-1'].ConstrainedSketch(name='sketch0',
                                        sheetSize=200.0)
mdb.models['Model-1'].sketches['Zugstab-viertel'].
       Line(point1=(0.0, 0.0), point2=(75.0, 0.0))
```

model.py:

```
m = mdb.models['Model-1']
s = m.ConstrainedSketch(name='sketch0',
                        sheetSize=200.0)
s.rectangle(point1=(0.0, 0.0), point2=(b, h))
```

Run the new *model.py* file in Abaqus CAE and continue

Header for model scripts

```
from abaqus import *
from abaqusConstants import *
from caeModules import *
import os

session.journalOptions.setValues(
              replayGeometry=COORDINATE,
              recoverGeometry=COORDINATE)
DIR0 = os.path.abspath('')
TOL = 1e-6
Mdb()          # Reset model
```

Creating geometry

Draw in sketch

```
s = m.ConstrainedSketch(name='sketch0',
                        sheetSize=200.0)
s.Line(point1=(x1,y1), point2=(x2,y2))
s.rectangle(point1=(x1,y1), point2=(x2,y2))
s.Spline(points=((x1,y1), (x2,y2), ...))
s.CircleByCenterPerimeter(center=(xc,yc),
                          point1=(x1,y1))
```

Create part (2D: TWO_D_PLANAR, 3D: THREE_D)

```
p = m.Part(dimensionality=TWO_D_PLANAR, name='plate',
           type=DEFORMABLE_BODY)
p.BaseShell(sketch=s)                    # bei 2D
p.BaseSolidExtrude(depth=t, sketch=s)    # bei 3D
```

Refernce points and datum planes

```
a = m.rootAssembly
rp1 = a.ReferencePoint(point=(x,y,z))
cut0 = p.DatumPlaneByPrincipalPlane(offset=off_z,
                    principalPlane=XYPLANE)
p.PartitionCellByDatumPlane(cells=p.cells[:],
                datumPlane=p.datums[cut0.id])
```

Cut and partitioning (e.g. 2D with second sketch)

```
p.Cut(sketch=s)
p.PartitionFaceBySketch(faces=p.faces[:], sketch=s)
```

Steps, loads and boundary conditions

```
s1 = m.StaticStep(name='step1',initialInc=1,
       minInc=0.1,maxInc=1,previous='Initial')
m.DisplacementBC(name='fix_00',u1=0,u2=0,ur1=0,
    createStepName='Initial', region=i.sets['punkt0'])
m.ConcentratedForce(name='Load-1', cf2=20.0,
    createStepName='step1', region=i.sets['punkt0']))
```

Materials and sections

```
a = m.rootAssembly
mat = m.Material(name='Stahl')
mat.Elastic(table=((210000.,0.3), ))

m.HomogeneousSolidSection(material='Stahl',name='Stahl')
p.SectionAssignment(region=p.sets['all'],
                    sectionName='Stahl')
inst = a.Instance(dependent=ON, name='Inst-1', part=p)
```

Move and rotate instances

```
a.rotate(instanceList=('Inst-1', ),axisPoint=(0,0,0),
         axisDirection=(0,1,0), angle=45.0)
a.translate(instanceList=('Inst-1', ), vector=(0,1,0))
```

Methods and funtions of object: `dir(object)`

Selecting, sets, surfaces

```
p.Set(name='all', faces=p.faces[:])
p.Set(name='rp', referencePoints=(
      p.referencePoints[rp1.id],))
```

Select by coordinates

```
p.faces.findAt(((x1,y1,z1),),((x2,y2,z2),))
p.faces.getByBoundingBox(xMin=x1-TOL,xMax=x2+TOL)
pos_list = [[x1,y1,z1], [x2,y2,z2]]
p.faces.findAt(coordinates=pos_list)
```

Select by properties

```
edge_list = [i for i in p.edges if i.getSize() > 5]
p.Set(name='edg_set', edges=
      [p.edges[i.index:i.index+1] for i in edge_list])
```

Create a surface

```
p.Surface(name='name', side1Edges=edges)
```

Combine sets

```
p.SetByBoolean(name='Set3', sets=(p.sets['Set1'],
                                  p.sets['Set2'],))
```

oder mit `operation=DIFFERENCE/INTERSECTION`

Mesh

Seeding and meshing

```
p.seedPart(size=el_mesh)
p.seedEdgeBySize(edges=p.sets['tie'].edges, size=0.2)
p.generateMesh()
# failed elements?!
mesh_qual = p.verifyMeshQuality(criterion=ANALYSIS_CHECKS)
failed_elem = mesh_qual['failedElements']
```

Different element type (always two of them: e.g. CPE8R, CPE6)

```
a.setElementType(elemTypes=(mesh.ElemType(elemCode=CPE8R,
       elemLibrary=STANDARD, secondOrderAccuracy=OFF,
       hourglassControl=DEFAULT,
       distortionControl=DEFAULT), mesh.ElemType(...)),
       regions=(i.faces[:], ))
```

Running the job

```
j = mdb.Job(memory=90, memoryUnits=PERCENTAGE,
       model='Model-1', name='Job-1',
       multiprocessingMode=THREADS, numCpus=2,
       numDomains=2, type=ANALYSIS)
mdb.saveAs(pathName='test.cae')
j.submit(consistencyChecking=OFF)
j.waitForCompletion()
```

Martin Pletz • martin.pletz@unileoben.ac.at • www.martinpletz.com/fe-scripting • version 0.2 • updated: 2020-10
Python: www.python.org, ABAQUS® (Version 2017): www.simulia.com

Evaluating results using Python in ABAQUS

General procedure

In the .odb file, all the results of an Abaqus job are saved which are defined in the model. Those results are time-dependent result variables (*History Output*) and fields of result variables (*Field Output*). The *Field Output* can be outup at nodes (displacement, temperature, etc.) or at integration points (stress, srain, heat flux, etc.).

In the Abaqus Viewer it is also possible to evaluate Field Output results along a path. The results are interpolated to do so, which can be rather slow.

You can view the *Field Output* and the *History Output* in the Abaqus Viewer which is also included in Abaqus CAE. Also, you can create and evaluate output variables along paths. To view *Field Output* outside Abaqus, it's best to create images of it. Results along paths or *History Output* is best written into .dat files, that can be used outside of Abaqus for further evaluations and diagram creation (e.g. using the Python module matplotlib), because the diagrams that Abaqus creates look ugly ☺.

Preparing the Abaqus model

Default *Field Output:* (L)E, S, U, RF, RM, PE, PEEQ, …
To define more filed output write:

```
m.FieldOutputRequest(name='F-Output-2',
       createStepName='Step-1',variables=('IVOL',))
```

Create history output in that way:

```
m.HistoryOutputRequest(name='H-Output-2',
        createStepName='Step-1', variables=('U1', ),
        region=inst.sets['Set-1'])
```

The best way of plotting field output outside of Abaqus is to also output the coordinates (at least in the initial step) of all nodes (N) and integration points (IP) as field output. For computing mean values of result variables like stresses, you also need the integration point volumes (IVOL). This output for the integration points is not possible in Abaqus CAE, so you have to write it to the .inp file, which you can do like this in Abaqus:

```
model.keywordBlock.synchVersions()
block_list = model.keywordBlock.sieBlocks
pos_i = [i for i, j in enumerate(block_list) if
         'Output, field' in j]
# Koordinaten an IP (COORD) & Volumen IP (IVOL)
str_out = "**\n*Element Output, directions=YES\n\
           COORD,IVOL,\n**\n*Node Output\nCOORD,"
# String einfuegen!
model.keywordBlock.insert(pos_i[0], str_out)
```

Header for evaluation scripts

```
from abaqusConstants import *
import os, sys, pickle
import numpy as np

DIR0 = os.path.abspath('')

odb = session.openOdb(name=odb_name + '.odb')
```

Plotting view output (images)

Change size of viewport (e.g. 300x200 pixel)

```
vp = session.viewports['Viewport: 1']
vp.setValues(displayedObject=odb)
vp.restore()
vp.setValues(origin=(5,5))
vp.setValues(width=300, height=200)
```

Showing Field Output, e.g. Mises stress (INVARIANT, also COMPONENT)

```
vp.odbDisplay.display.setValues(plotState=
                                (CONTOURS_ON_DEF, ))
vp.odbDisplay.setPrimaryVariable(variableLabel='S',
        outputPosition=INTEGRATION_POINT,
         refinement=(INVARIANT, 'Max. Principal'),)
```

Save and load view

Rotate, move and zoom view until ok, then:
View/Save with „User 1" and „save current", copy recorded command, e.g..:

```
session.View(name='User-1', nearPlane=150,
       farPlane=240, width=63, height=40,
       projection=PERSPECTIVE, cameraPosition=(15,0,20),
       cameraUpVector=(0,1,0), cameraTarget=(15,0,0),
       viewOffsetX=-3.5, viewOffsetY=0.02, autoFit=OFF)
```

Then you can load a view in the Python file by::

```
vp.view.setValues(session.views['User-1'])
```

Print image to file

```
vp.viewportAnnotationOptions.setValues(legendFont=
      '-*-verdana-medium-r-normal-*-*-140-*-*-p-*-*-*')
vp.viewportAnnotationOptions.setValues(triad=OFF,
       state=OFF, legendBackgroundStyle=MATCH,
       annotations=OFF, compass=OFF, title=OFF)
# output image
session.printOptions.setValues(reduceColors=False)
session.printToFile(fileName=odb_name, format=PNG,
                    canvasObjects=(vp1,))
```

Evaluating history output

```
step1 = odb.steps['Belastung']
# in steps: History Regions with history output
# .dat: List of time/value pairs
ae_data = step1.historyRegions['Assembly ASSEMBLY'].\
                    historyOutputs['ALLAE'].data
# save in .dat file (np.save(): binary)
np.savetxt('res_ae.dat', np.array(ae_data))
```

Load data into Python

```
import numpy as np
res_ae = np.loadtxt('res_ae.dat') # binary: np.load()
```

Commands from recorded macro or .rpy file

Evaluating field output

```
# last frame
frame_end = odb.steps.values()[-1].frames[-1]
field_out = frame_end.fieldOutputs['S']
# pos can be NODAL or INTEGRATION_POINT
field_out = field_out.getSubset(position=pos)
```

Directly (slow, but robust)

```
s_out= np.array([i.data for i in field_out.values])
```

With BulkDataBlocks (fast, but with some bugs)

```
s_bulk = field_out.bulkDataBlocks
s_out = np.concatenate([i.data for i in s_bulk])

# get field output for one element type
# i=0 to element number, np.copy neccessary (bug!)
s_bulk = np.copy(field_out.bulkDataBlocks[i].data)
```

Write to .dat file

```
np.savetxt(odb_name+'_s.dat', s_out)
```

Evaluate results along paths

Create path from coordinates

```
pth = session.Path(name='Path-1', type=POINT_LIST,
        expression=((0,0,0), (20,0,0), (50,0,0)))
```

Field output to be evaluated along path

```
vp.odbDisplay.setPrimaryVariable(variableLabel='E',
        outputPosition=INTEGRATION_POINT,
        refinement=(COMPONENT, 'E22'))
```

Output for 20 eqiuidistant points on path

(instead of TRUE_DISTANCE : TRUE_DISTANCE_X or X_COORDINATE possible*)*
shape: If path over DEFORMED or UNDEFORMED geometry

```
xy = session.XYDataFromPath(name='XYData-1', path=pth,
      includeIntersections=False, shape=DEFORMED,
      projectOntoMesh=False, pathStyle=UNIFORM_SPACING,
      numIntervals=20, projectionTolerance=0,
      labelType=TRUE_DISTANCE)

np.savetxt('path_out_e22.dat',np.array(xy))
```

Martin Pletz • martin.pletz@unileoben.ac.at • www.martinpletz.com/fe-scripting • version 0.2 • updated: 2020-10
Python: www.python.org, ABAQUS® (Version 2017): www.simulia.com

Diagrams & Python-tricks

Some handy functions and tools

Print to command line from Abaqus

```
print >> sys.__stdout__,'hello',
```

Pass arguments to the python code:

```
abaqus cae noGUI=model.py -- 'hello!'

# read the string 'hello!' in Abaqus:
sys.argv[-1]
```

Create/delete folders

```
import os, shutil

# absolute working path
DIR0 = os.path.abspath('')

# create folder (delete existing folder)
if os.path.exists(dir_name):
    shutil.rmtree(dir_name)

os.mkdir(dir_name)

# change the work directory
os.chdir(dir_name)

# get a list of files and folders in directory
file_list = os.listdir(dir_name)
```

shutil.rmtree()

Run Abaqus from Python script

```
import subprocess

subprocess.call('abaqus cae noGUI=f.py', shell=True)
subprocess.call('abaqus interactive job=inp_file',
                shell=True)
```

Suggested Python editors

Visual Studio Code (Microsoft) ☺
code.visualstudio.com

PyCharm (JetBrains) ☺
atom.io

Spyder (Anaconda) 😐
www.anaconda.com

Simple diagrams (matplotlib)

```
import numpy as np
import matplotlib.pyplot as plt

eps = np.linspace(0,1)
sigma = eps**2

fig, ax = plt.subplots(ncols=1, nrows=1,
                       figsize=np.array((8.,5.))/2.54)

# realstring for special characters / latex formulas
ax.set_title(r'stress $\sigma$ over strain
               $\varepsilon$', fontsize=10)

# plot a curve
ax.plot(eps, sigma, 'r', label='exp.')
```

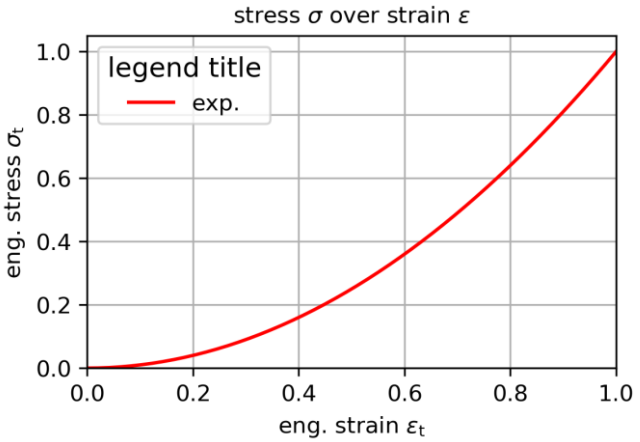
Format string, e.g. 'r' for red

```
'r', 'b', 'g', 'c', 'm', 'y', 'k', 'w'
```

'-' ▬, '--' ▬ ▬, '.' ● ●, '^' ▲▲, 'o' ●●

```
# axis labels
ax.set_xlabel(r'eng. strain $\varepsilon_\mathrm{t}$')
ax.set_ylabel(r'eng. stress $\sigma_\mathrm{t}$')

# change the axis limits
ax.set_xlim(0,1)
ax.set_ylim(ymin=0)

# display a legend
leg = ax.legend(ncol=1)
# set a legend title
leg.set_title('legend title', prop={'size':12})
# show a grid
ax.grid()

# fit diagram into image size
fig.tight_layout(pad=0)

# save as .png (dpi:resolution) or .pdf
fig.savefig('sig_eps.png', dpi=300)

fig.show() # display plot in Python
```

matplotlib.org



Load modules

```
# additinal path with modules
sys.path.append(r'C:/FEM-Data/Py-Pakete')

from my_functions import *
```

Save/load data

```
import numpy as np
import pickle
```

With Pickle (alle objects, e.g. dictionaries)

```
pickle.dump(var_list, open('daten.dat', 'wb'))

var_list = pickle.load(open('daten.dat', 'rb'))
```

With NumPy, for arrays (binary: faster)

```
# Save in .dat file (binary np.save())
np.savetxt('res_ae.dat', np.array(ae_data))

# load data from .dat file (binary: np.load())
res_ae = np.loadtxt('res_ae.dat')
```

Martin Pletz • martin.pletz@unileoben.ac.at • version 0.2 • updated: 2020-10
Python: www.python.org