PolyN: A python library for modeling and testing plane stress yield functions

(Usage instructions)

Stefan C. Soare / 2023-04-07

Content

1. Overview

This code implements the theory in *Calibration and fast evaluation algorithms for homogeneous orthotropic polynomial yield functions*, by S.C. Soare and M. Diehl - referred throughout this document as **SSMD23**[a]. Currently, the library is structured on two problems:

(1) **Modeling:** Scripts `PolyN_V3F.py` and its driver `PolyN_main.py`, and `PolyN_symb.py` and its driver `PolyN_symMain.py`.

(2) **Cup-Drawing simulation:** Script `abqPolyNshell.py`, its driver `abqPolyN_mains.py`, and UMAT subroutines `abqUMAT_Hill48.for`, `abqUMAT_PolyN.for`.

The code is written in `python`. **Modeling** requires the following packages:
`numpy` (`https://numpy.org/install/`)
`matplotlib` (`https://matplotlib.org/stable/users/installing.html`)
`cvxopt` (`https://cvxopt.org/install/index.html`)
`sympy` (`https://docs.sympy.org/latest/install.html`)

Cup-Drawing simulation requires the commercial FEA software Abaqus.

The simplest work-flow (for **Modeling**) is:

- Install `Python` (version ≥ 3.7) (`https://www.python.org/`)
- Make sure python(.exe) is on the search path (i.e., it can be run from a command prompt/shell); If not, add it to the 'PATH'/shell environment variable
- Install the required packages via `pip3`; In a command shell run:
  ```
  pip3 install numpy, matplotlib, cvxopt, sympy
  ```

Note, however, that this mixing of specific libraries (such as `cvxopt` and `sympy`) onto the global python installation is not recommended. The best work-flow is via a python virtual environment where libraries and specific applications contexts are kept in isolation from the main installation. Thus after installing Python, create a directory where work on material modeling is to take place. From now on it will be assumed that this folder is named[b] 'MATM'. Then:

[a]References are made to preprint 10.13140/RG.2.2.16581.32488.

[b]For an easy highlight within text, folder names are enclosed in single quotes. They are not part of the name itself.

- In a command shell navigate to 'MATM' and execute:
  `python -m venv OPTIM`
  (the above command will create a subdirectory 'OPTIM' - the virtual environment - with a bare bones pseudo-installation of Python)

- navigate to 'OPTIM' and execute:
  `.\Scripts\activate` (or on Windows: `.\Scripts\Activate.ps1`)
  (the above activates the virtual environment 'OPTIM')

- Then install the required libraries by executing

```
pip3 install numpy
pip3 install matplotlib
pip3 install cvxopt
pip3 install sympy
```

- one can check the list of installed packages with the command:
  `pip3  list`

- one can exit a virtual environment by executing:
  `deactivate`

Folder 'OPTIM' is used just as a local installation directory (won't be needed from now on). Since a lot of input/output is to be generated, navigate back to 'MATM' and create a new sub-directory called 'PolyN'. In 'PolyN':

- create a sub-directory called 'figsPolyN'

- in 'figsPolyN' create two sub-directories called 'DATA' and 'PLOTS'

- copy the files

  `PolyN_V3F.py`, `PolyN_main.py`, `PolyN_symb.py`, `PolyN_symMain.py`

  in sub-dir 'PolyN'.

The final directory structure should look as below:

```
|-- MATM
    |-- OPTIM
    |-- PolyN
        |-- figsPolyN
            |-- DATA
            |-- PLOTS
        PolyN_V3F.py
        PolyN_main.py
        PolyN_symb.py
        PolyN_symMain.py
```

Folder 'DATA' is used for both input and output (report and parameter text-files); Folder 'PLOTS' is used only for output (figures). Within the 'DATA' folder the input data may be further structured on specific materials, as described later.

From now on it is assumed that the OPTIM virtual environment is activated and that the current shell location is 'PolyN'.

2. The input file for a material[c]

The most important input file is the material data. This is where the mechanical properties and the PolyN model are specified. The following example of material input file illustrates its structure[d]

```
# This is a comment line (ignored by the file parser)
name=AA6016T4_Yld2000_Siegen
degree= 6
## Elasticity
EE= 70000.000000000000
NU= 0.330000000000
## Note: Swift hardening: H = A*(B+eps)**C
AA= 498.8000000000
BB= 0.008900000000
CC= 0.285000000000
## Directional (uniaxial data). Note: s0 must always be =1 (on data row 'd=0.0,0.0,...')
d=0.0, 0.0, 1.000, 0.526, a
d=0.0, 15, 0.944, 0.344, a
d=0.0, 30, 0.913, 0.301, a
d=0.0, 45, 0.908, 0.253, a
d=0.0, 60, 0.898, 0.294, a
d=0.0, 75, 0.928, 0.393, a
d=0.0, 90, 0.983, 0.601, a
## balanced-biaxial data
d=1.0, 0.0, 1.0, -0.5, a
## plane strain along RD:
d=0.5,0.0,1.025,0.0,a
## plane strain along TD:
d=0.500,90,1.0,0.0,a
## other plane strain or extended data (q<0)

## Weights
wa=0.975
war=0.1
wvr=0.2

###'qHill' specifies whether samples from the corresponding Hill'48 quad are used
###This must be used only if no virtual data is available and N is high (>6)
## No samples from Hil'48 are used if qHill=0
qHill=1
```

This file is used for fitting and generating parameters output for FE-simulations (in which case all entries may be used), or just for plotting (in which case only the `name`, the `degree` and the directional uniaxial data are used). Since most entries are self-explanatory, see also SSMD23, we highlight here only the most important features.

The field `name` is essential since it is used for naming all its corresponding output by appending to it various extensions. *Thus it must be uniquely associated with a model* (If reused in a different material input file, any previous model associated with this `name` will be overwritten by new output). Do not use blank spaces and commas in `name` since this may have unintended consequences.

The field `degree` specifies which PolyN is to be used when fitting the data, or when generating output for FE-simulations.

[c] All input is case-sensitive. All files illustrated here, and more, can be downloaded from the GitHub repository of PolyN: `https://github.com/stefanSCS/PolyN`.

[d] In all input files, lines starting with the character '#' are ignored.

The format for specifying the sheet metal plastic properties is

d = q, θ, $\overline{\sigma}(q,\theta)$, $r(q,\theta)$, type

where `d=` tells the parser that this is a plasticity input line, with five entries separated by commas; q, θ, $\overline{\sigma}(q,\theta)$, and $r(q,\theta)$ are described in Section-2 of SSMD23; `type` must be either `a` (indicating an *actual* value), or `v` (indicating a *virtual* value), as detailed in Section-3 of SSMD23.

3. Modeling: Generating PolyN clones

This feature of the library can be used to convert any yield function proven to be a homogeneous polynomial to a PolyN function. It does so by using the `SymPy` python library to calculate the corresponding polynomial parameters. In conjunction with the UMAT subroutine, which is applicable to any PolyN, this feature facilitates FE-simulations with virtually any yield function that is proven to have polynomial nature. Yield functions currently implemented (more will be added as the code evolves):

Yld'89, Yld2000-2D, BBC2005, Yld2004-18p (plane stress), FACET (plane stress), Caz2018-Ort (plane stress).

No python programming skills are assumed and hence every code functionality can be accessed via text file interfaces. For PolyN-cloning the input file has the following structure:

```
###Input file for the Python script PolyN_symMain.py
##Input data format:
##object:value
##where 'object' can be: func, degree, alpha_1, alpha_2, fileInpData, fileFACET

##Select yield function to transform to PolyN.
##Available options: Yld89, Yld2000_2D, BBC2005, Yld2004_18p, FACET, Caz2018_Ort
func:  Yld2000_2D

##Select the degree N of PolyN
degree:  8

## Parameters of the yield function (Separator must be ',')
alpha_1: 0.9238, 0.9967, 0.9365, 1.0227, 1.0303, 1.0075, 0.8385, 1.3761

# Input data file (experimental data, used mostly for plotting and exporting for FE-sims)
fileInpData:   matAA6016T4_TUAT_Siegen.txt
```

The above specifies that a Yld2000-2D model, of degree 8, with parameters on line `alpha_1`, is to be cloned by Poly8. The rest of the data is to be found in the material input file `matAA6016T4_TUAT_Siegen.txt` (the content of which may be assumed identical to the listing in the previous section). Assuming the above was saved in a file named `symb_AA6016T4_Yld2000_Postech.txt`, located in a sub-directory 'AA6016T4' of 'DATA', this information is entered in the driver script `PolyN_symMain.py` as follow:

```
fName = 'symb_AA6016T4_Yld2000_Postech.txt'
subDir = 'AA6016T4'
```

The first row tells the script which model is to be cloned; the second - where to look for it. Note: Both input files (`symb_*.txt` and `matAA60*.txt` must be located in the same sub-directory specified by `subDir`). Save and execute in shell:

```
python  PolyN_symMain.py
```

Assuming all went well, all output is saved in the provided `subDir` of both 'DATA' and 'PLOTS' as follow:

```
|-- DATA
    |-- AA6016T4
        matAA6016T4_TUAT_Siegen.txt
        symb_AA6016T4_Yld2000_Postech.txt
        AA6016T4_Yld2000_Siegen_deg8_Err_and_Coeff.txt
        AA6016T4_Yld2000_Siegen_deg8_Export.txt
        AA6016T4_Yld2000_Siegen_deg8_FEdata.txt
|-- PLOTS
    |-- AA6016T4
        AA6016T4_Yld2000_Siegen_P8_rValue.png
        AA6016T4_Yld2000_Siegen_P8_Sigma.png
        AA6016T4_Yld2000_Siegen_P8_sxySections.png
```

In 'DATA': The file `AA6016T4_Yld2000_Siegen_deg8_Err_and_Coeff.txt` provides a basic assessment and PolyN parameters; The file `AA6016T4_Yld2000_Siegen_deg8_Export.txt` provides an export of some sampling values of the model; Finally, `AA6016T4_Yld2000_Siegen_deg8_FEdata.txt` is the file that can be used as input for the PolyN-UMAT subroutine to be used in FE-simulations.

In 'PLOTS': The files `AA6016T4_Yld2000_Siegen_P8_*.png` show plots of the generalized directional stresses, r-values and of several $\sigma_{xy} = const$ sections through the yield surface.

4. Modeling: Fitting with PolyN

Working with PolyN models is done via the script `PolyN_main.py`. Input to this script is provided via a text file where one specifies the `task` to be executed:

`modelFit`, `testPoly`, `convexCheck`

● **modelFit**

With this task one fits a PolyN model to data. Must provide the material input file (described above in Section-2) and the local sub-directory where it is located. The input file for the script is structured as follow:

```
## input file for PolyN_main.py

## select option (what the script does): modelFit, convexCheck, testPoly
task: modelFit

## specify material input file
fileInpData: matAA6016T4_TUAT_P6.txt
```

Assuming the above was saved in a file named `polyn_AA6016T4_TUAT_P6.txt`, located in the sub-directory 'AA6016T4' of 'DATA', this information is entered in the driver script `PolyN_main.py` as follow:

```
fName = 'polyn_AA6016T4_TUAT_P6.txt'
subDir = 'AA6016T4'
```

Note: Both input files (`symb_*.txt` and `matAA60*.txt` must be located in the same sub-directory specified by `subDir`). Save and execute in the shell:

```
python  PolyN_main.py
```

Assuming all went well, all output is saved in the provided `subDir` of both 'DATA' and 'PLOTS' as follow:

```
|-- DATA
    |-- AA6016T4
        matAA6016T4_TUAT_P6.txt
        polyn_AA6016T4_TUAT_P6.txt
        AA6016T4_P6_deg6_Err_and_Coeff.txt
        AA6016T4_P6_deg6_Export.txt
        AA6016T4_P6_deg6_FEdata.txt
|-- PLOTS
    |-- AA6016T4
        AA6016T4_P6_P6_rValue.png
        AA6016T4_P6_P6_Sigma.png
        AA6016T4_P6_P6_sxySections.png
```

A note about the `*_Export.txt` files is in order. They contain a range of sampled values of PolyN models and can be used as material input files for fitting with PolyN+2 models (this is the default implemented behavior; However, the degree can be adjusted by manually editing the exported file).

- **testPoly**

This task allows for testing any set of PolyN parameters (not necessarily obtained by the PolyN fitting algorithm in SSMD23). An example of input file, named `polyn_AA6016T4_TUAT_P6_test.txt` is shown below:

```
## input file for PolyN_main.py

## select option (what the script does): modelFit, convexCheck, testPoly
task: testPoly

## specify material input file
fileInpData: matAA6016T4_TUAT_P6_test.txt



## specify parameters (one line, comma separated)
alpha: 1.00000, -2.06815, 3.53177, -3.93091, 3.91107, -2.49639, 1.10835, 16.41230 ...
```

Assuming the above file is in sub-dir 'AA6016T4' of 'DATA', this info is specified in the `PolyN_main.py` driver script as follow:

```
fName = 'polyn_AA6016T4_TUAT_P6_test.txt'
subDir = 'AA6016T4'
```

Save and execute in the shell:

```
python  PolyN_main.py
```

The output is similar to that of a `modelFit` task.

- **convexCheck**

This task allows for testing convexity at a grid of points denser than used in the optimization algorithm (Note that this task is computationally expensive for high order polynomials: the optimization algorithm uses a quasi-uniform grid on the unit sphere of about 10^5 points; this convexity check doubles that to about 2×10^5 points).

An example of input file is as follow:

```
## input file for PolyN_main.py

## select option (what the script does): modelFit, convexCheck, testPoly
task: convexCheck

## either specify parameters input file (*Err_and_Coeff.txt, *FEdata.txt)
```

```
#fileCVXData:  AA6016T4_P6_deg6_Err_and_Coeff.txt
fileCVXData:  AA6016T4_P6_deg6_FEdata.txt


## or directly specify parameters
#alpha: 1.00000, -2.06815, 3.53177, -3.93091, 3.91107, -2.49639, 1.10835, 16.41230, ...

## in the latter case must specify degree
#degree: 6
```

Assuming the above info is saved in a file `polyn_AA6016T4_TUAT_P6_cvxCheck.txt` in sub-dir 'AA6016T4' of 'DATA', save this info to the driver script as

```
fName = 'polyn_AA6016T4_TUAT_P6_cvxCheck.txt'
subDir = 'AA6016T4'
```

Save and execute in the shell:

```
python  PolyN_main.py
```

The output (for the above particular example) is:

```
Checking convexity.......
Convexity checked at 214436 locations:
min Gauss KG =  0.000935475511055033
min det1, det2, det3: 0.0, -1.8568480086855743e-13, -1.742507180213515e-12
```

5. Tensile test and cup-drawing simulation

This is the python-module for Abaqus simulations. Currently, only the S4R element is used for meshing. Python scripts `abqPolyNshell.py`, `abqPolyN_mains.py`, and the UMAT subroutines `abqUMAT_Hill48.for`, `abqUMAT_PolyN.for`[e] must be located in the same directory. The latter is usually a directory dedicated to Abaqus work and we assume it is called 'abqPolyN'. In this folder create two sub-directories called 'DATA' and 'RESULTS'. The two sub-folders serve similar purposes as for PolyN - to structure the input and the output of the scripts and of the Abaqus jobs. Thus 'abqPolyN' should look as follow:

```
|-- abqPolyN
    |-- DATA
    |-- RESULTS
    abqPolyNshell.py
    abqPolyN_mains.py
    abqUMAT_Hill48.for
    abqUMAT_PolyN.for
```

The data required for the setup of a simulation is managed via an input text file. Two types of simulations can be performed: uniaxial tensile test and cup-drawing (Swift cupping test).

● **Uniaxial tensile tests**
For simulating tensile tests an example of input file - `abq_uniaxial_AA6022T4.txt` - is shown below

```
## input file for abqPolyN_mains.py (Abaqus uniaxial tensile tests)


## set 'uax' to True to perform uniaxial tests simulations
```

[e]Note: On Linux systems the extension for Fortran files might need to be changed from '.for' to '.f'

```
## set 'uax' to False for cup drawing simulations
uax=True
## set 'caeOnly' to True if only the *.cae model database file is desired (no sims)
## set 'caeOnly' to False if simulations are to be run
caeOnly=False

## Abaqus specifics: max increment, max numb of incr
dtMax:0.003
maxNInc:3000

## for Hill'48 sims must provide additional details
## Hardening type: a string in ['Swift','Voce']
hLaw:Voce,
## Elasticity parameters: (E,nuPoisson,muG)
eParam:70000.0,0.33,26200.0
## Hardening parameters: (A,B,C)
hParam: 436.0,222.0,6.75
## R-values required by Hill48: (r0,r45,r90) from experiments
rParam:0.8,0.37,0.54
## thickness(mm) of the metal sheet
hThick:1.0

## HRATIO is used to calculate the boundary displacement (see doc)
HRATIO: 7.48

##---------- List of tests
## Note: the specification of a test has the format (see doc)
## sim: inputFile | UMAT | PolyN | angles

## one can perform a single test (one material, one angle)
sim: AA6022T4_0_HillAbaqus | False | False | 0.0

## delete the rest of this file if a single test is desired

## or several tests (batch testing)

sim: AA6022T4_0_HillUMAT | umatHill | False | 0.0,15.0,30.0,45.0,60.0,75.0,90.0

sim: AA6022T4_H_ExptP6_ExptP8_ExptP10_deg12_FEdata.txt| umatPoly| True| 0.0,15.0
```

The first part (basic parameters) of the input file is self-explanatory. It is a good idea, before attempting any simulation, to first test the script in CAE mode: `caeOnly = True`. This will generate only the CAE model (no simulation is run). The CAE-model can then be opened with Abaqus-CAE for inspection; It can also be used as an alternative mean of running jobs (with Abaqus-CAE).

A critical parameter is `HRATIO` (hardening ratio decrease). This determines the potential maximum uniaxial strain ϵ_{max} via the relationship

$$H'(0) = h_{rat} H'(\epsilon_{max})$$

where we used h_{rat} for `HRATIO`. Too large a hardening ratio leads to too large strains and failure to complete a simulation. This parameter must be carefully set especially when doing batch testing. A few trial and error preliminary tests using Hill'48 for several angles are usually enough to trap a high enough hardening ratio. Also critical for a simulation is the time increment: It should be small enough to allow proper elastic-plastic transitions and also to comply with the implementation of the return mapping algorithm (the two UMATs are not designed to handle large increments).

The second part of the input file specifies the simulations to be performed[f]. A uniaxial test is specified by (the order is important):

- `inpFile`: For PolyN this is the `*_FEdata.txt` file containing the elasticity, hardening and PolyN parameters. All Abaqus based generated output is named by appending specific extensions to this file name. Note that for Hill'48 based simulations no such parameter input file is needed (because it is generated based on info in the first part of the above input file). Nevertheless, a name is still required for identifying output.
- `UMAT`: can be `False`, `umatHill`, or `umatPoly`. In the first case, the default Abaqus implementation of Hill'48 is used; In the second and third, the UMAT implementations of Hill'48 and PolyN, respectively.
- `PolyN`: can be `False` or `True`. It is the latter when PolyN is used.
- `angles`: A sequence (comma separated) of angles (in degrees, measured from RD) at which the test sample is cut-out from the sheet.

Important: Make sure the appropriate hardening law subroutine is activated in UMAT (by renaming).

Save the following data in the driver script `abqPolyN_mains.py` (to tell it where to find the input file)

```
fName='abq_uniaxial_AA6022T4.txt'
subDir='AA6022T4'
```

Then navigate to 'abqPolyN' and run the script with the shell command

```
abaqus cae noGUI=abqPolyN_mains.py
```

Assuming all went well, the output in the 'RESULTS/AA6022T4' directory should look as below

```
|-- DATA
    |-- AA6022T4
        abq_uniaxial_AA6022T4.txt
|-- RESULTS
    |-- AA6022T4
        AA6022T4_0_HillAbaqus_ES_DATA_0.txt
        AA6022T4_0_HillAbaqus_JOB_0.txt
        AA6022T4_0_HillAbaqus_Node_0.txt
        AA6022T4_0_HillAbaqus_Screen_0.png
        AA6022T4_0_HillUMAT_ES_DATA_0.txt
        AA6022T4_0_HillUMAT_JOB_0.txt
        AA6022T4_0_HillUMAT_Node_0.txt
        AA6022T4_0_HillUMAT_Screen_0.png
        AA6022T4_0_HillUMAT_ES_DATA_15.txt
        AA6022T4_0_HillUMAT_JOB_15.txt
        AA6022T4_0_HillUMAT_Node_15.txt
        AA6022T4_0_HillUMAT_Screen_15.png
        ......
        AA6022T4_0_HillUMAT_ES_DATA_90.txt
        AA6022T4_0_HillUMAT_JOB_90.txt
        AA6022T4_0_HillUMAT_Node_90.txt
        AA6022T4_0_HillUMAT_Screen_90.png
        AA6022T4_H_ExptP6_ExptP8_ExptP10_deg12_FEdata_ES_DATA_0.txt
        AA6022T4_H_ExptP6_ExptP8_ExptP10_deg12_FEdata_JOB_0.txt
        AA6022T4_H_ExptP6_ExptP8_ExptP10_deg12_FEdata_Node_0.txt
```

[f] The script reuses the CAE-model and the corresponding Abaqus-JOB files (status, message, output, etc) and only specific output is saved. Thus in CAE-mode only one test can be specified (if a list is provided, only the first is selected).

```
        AA6022T4_H_ExptP6_ExptP8_ExptP10_deg12_FEdata_Screen_0.png
        AA6022T4_H_ExptP6_ExptP8_ExptP10_deg12_FEdata_ES_DATA_15.txt
        AA6022T4_H_ExptP6_ExptP8_ExptP10_deg12_FEdata_JOB_15.txt
        AA6022T4_H_ExptP6_ExptP8_ExptP10_deg12_FEdata_Node_15.txt
        AA6022T4_H_ExptP6_ExptP8_ExptP10_deg12_FEdata_Screen_15.png
```

To each test angle/job (identified in the file name by the appended number) there correspond four files ('*' servs as a placeholder):

`*_ES_DATA_*`: This file records the (total) strain components at the integration point of the middle element shown in Fig.12 of SSMD23.

`*_Node_*`: This file records the displacements at the nodes specified in Fig.12 of SSMD23.

`*_JOB_*`: This files records the wall-clock time of the test.

`*_Screen_*`: This file contains a snapshot of the final configuration of a simulation.

There are quite a few reasons a simulation may not complete successfully. Some have been mentioned above. A symptom is the appearance of `nan` in the first two data files listed above. Another, is an empty snapshot. Mesh refinement is often a solution (not exposed in the current version of the input file for the uniaxial test case). Another, is to decrease the allowed max time increment.

Finally, some post-processing of the above files can be performed by re-running the driver script in python mode:

```
python  abqPolyN_mains.py
```

Upon execution of this command additional output is added to 'RESULTS/AA6022T4':

```
|-- RESULTS
    |-- AA6022T4
        ......
        AA6022T4_0_HillUMAT_Hcurve_0.png
        AA6022T4_0_HillUMAT_Hcurve_15.png
        ......
        AA6022T4_0_HillUMAT_Hcurve_90.png
        AA6022T4_0_HillUMAT_RVals.png
        AA6022T4_0_HillUMAT_RVals.txt
        AA6022T4_0_HillUMAT_RVals_0.png
        ......
        AA6022T4_0_HillUMAT_RVals_90.png
        AA6022T4_0_HillUMAT_JOB.txt
```

The `*_Hcurve_*` files show the stress-strain curve of the test (against the theoretical prediction); The `*_Rvals_*.png` files show the evolution of the r-value of the test; The files `*_Rvals.png` gather all r-values predicted from simulations (at the end of each job specified in the input file) and compares them against the theoretical prediction (the text version of this files simply records values); Finally, the `*_JOB.txt` collects all wall-clock times for a material model (defined by a UMAT input parameters file).

- **Cup-Drawing test**

This simulation proceeds in 4 steps:

Step-1: Make contact between Holder and Blank and Die and Blank

Step-2: Make contact between Punch and Blank

Step-3: Move Punch until the Blank is nearly out of the Die shoulder (drawing step)

Step-4: Move both Punch and Die (in opposite directions) to remove contact

Modulo parameters that are specific to the problem (Fig.15 of SSMD23), the input to the Swift cupping test is structured similarly to that used for the uniaxial test. An example of input file - `abq_cpDraw_AA6016T4.txt` - is shown below:

```
## input file for abqPolyN_mains.py (Abaqus cup-drawing)

## set 'uax' to False for cup drawing simulations
uax: False
## set 'caeOnly' to True if only the *.cae model database file is desired (no sims)
## set 'caeOnly' to False if simulations are to be run
caeOnly: False

## Abaqus specifics: initial increment, max incr, max numb of incr
dtInitial:1.0e-5
dtMax:0.001
maxNInc: 1e+5

## Elasticity parameters: (E,nuPoisson,muG)
eParam:70000.0,0.33,26200.0
## Hardening type: a string in ['Swift','Voce']
hLaw: Voce
## Hardening parameters: (A,B,C)
hParam: 330.2, 190.0, 12.4

## thickness(mm) of the metal sheet
hThick:1.0
##blankDiam=Diameter of the disc-shaped blank
blankDiam:107.5
dieOpeningDiam:62.4
#dieShoulderRad=Radius of the arc between the Die-flange and the Die-wall
dieShoulderRad:10
punchDiam:60
punchNoseRad:5
holderClearance:1.04
frictionCoeff:0.07

## mesh control: multipliers along hoop and radial directions
##(if the blank diameter is large, one can safely use the default values: 1.0)
seedScaleC: 1.12
seedScaleR: 1.15

## contact parameters: contact pressures between Holder-, Die-, and Punch-Blank
pSoft: 1.5,1.0,1.0

## Punch travel (mm) in Steps-3,4
scaleP: 1.05
deltaPunch: 3.0

readOdb: 0
nCPUs: 1

sim: AA6016T4_Yld2000_UGent_ExptP6.11_deg8_FEdata.txt | umatPoly | True
```

The above file defines a cup-drawing model for the aluminum alloy AA6016-T4. The last line tells the script that `abqUMAT_PolyN.for` is to be used with input parameters specified in

`AA6016T4_Yld2000_UGent_ExptP6.11_deg8_FEdata.txt`

This file defines a Poly8 fit of a Yld2000-2D model of this alloy, constructed with a non-integer exponent (6.11). As shown in SSMD23, the Poly8 model is for all practical purposes identical to the Yld2000-2D model and hence it can be used as an alternative.

Parameters `seedScaleC` and `seedScaleR` (both with a default value of 1) can be used to increase the number of mesh seeds along the hoop and radial directions, respectively. Given the simple geometry of the blank, the default numbers of seeds along the two directions are taken proportional to the blank radius. For example, in the hoop direction the default number of seeds is the integer part of $\pi r/4$, where r is the blank radius. For smaller blanks the default mesh density may be deemed insufficient. It can be increased by scaling the default seeding via `seedScaleC` and `seedScaleR`.

The default punch travel in Step-3 is calculated based on the plane strain isotropic approximation of the problem. This in general guarantees that the rim of the partially formed cup is somewhere below the middle of the die shoulder at the end of Step-3. Increasing this default value of the travel can be useful (to get the rim of the partially formed cup as close as possible to the die shoulder base). This can be achieved by scaling up the default travel via parameter `scaleP`. Note however that increasing too much the punch travel in Step-3 will lead to a larger `dtMax` (not advisable in the drawing step) which then would need to be decreased; It can also lead to a complete push of the cup out of the die cavity (which is the purpose of Step-4).

In the final step, an additional translation is performed (punch still moving downwards, while die begins to move upwards). This is controlled by `deltaPunch` and it should be large enough to ensures that the fully formed cup is freed from its contact with the die (at the end of Step-4). Note, however, that increasing (too much) this translation increases also the default max increment in this step which may cause convergence problems.

Some simulation parameters may not work the same for all material models. The contact pressure, the (drawing) step increment, the mesh density, etc, may require fine tuning via some trial-and-error. It is not uncommon that Steps 1-3 complete while Step-4 fails to converge (precisely at the moment when the rim/ears are to be released from their contact with the die). Restart files can be used to experiment (this requires that simulations be run from Abaqus-CAE). However, the results at the end of Step-3 may still be useful and to extract them (by default, results are reported for Step-4) one can set `readOdb` to 3, in which case no simulation is performed and only the output at the end of Step-3 of a previous simulation is extracted. One can also set `readOdb=4`, in which case the results at the end of Step-4 are re-exported. The default value of `readOdb` is 0 (simulations are performed and the new data is exported as usual, at the end of Step-4).

Important: Make sure the appropriate hardening law subroutine is activated in UMAT (by renaming).

To run a simulation with the parameters specified in the input file `abq_cpDraw_AA6016T4.txt`, edit the driver script `abqPolyN_mains.py` as follow:

```
import abqPolyNshell as abqp

##--- input data block
fName='abq_uniaxial_AA6022T4.txt'
subDir='AA6022T4'

fName='abq_cpDraw_AA6016T4.txt'
subDir='AA6016T4'

###---execution block
inpD=abqp.readInpData(fName,subDir)
abqp.zexec(inpD)
```

Then navigate to 'abqPolyN' and execute in the shell:

```
abaqus cae noGUI=abqPolyN_mains.py
```

Assuming all went well, the output in the folder 'RESULTS/AA6016T4' should look as bellow:

```
|-- RESULTS
    |-- AA6016T4
        AA6016T4_Yld2000_UGent_ExptP6.11_deg8_FEdata_cpdrwJOB.txt
        AA6016T4_Yld2000_UGent_ExptP6.11_deg8_FEdata_cpdrwPROFILE.txt
        AA6016T4_Yld2000_UGent_ExptP6.11_deg8_FEdata_cpdrwScreen.png
        AA6016T4_Yld2000_UGent_ExptP6.11_deg8_FEdata_cpdrwSTATUS.txt
```

The content of each file is as follow:

- File `*JOB.txt` records the wall-clock time (for example, the simulation illustrated here, takes 8058s on Mach-2 specified in Table-3/SSMD23, i.e., approx, 2h:15m).
- File `*STATUS.txt` is a copy of the status file reported by Abaqus (Recall that new simulations overwrite all previous Abaqus-related files in the root directory 'abqPolyN').
- File `*cpdrwPROFILE.txt` records the initial positions of the nodes on the outer edge of the blank and their final displacements; Last row records the initial and final displacement of the center node of the blank.
- File `*cpdrwScreen.png` contains a snapshot of the final configuration.

Finally, one can generate a table of the calculated profile and a raw graph of the profile by executing the driver script in `python`-mode:

```
python  abqPolyN_mains.py
```

This command calculates the height profile, which is recorded in
`AA6016T4_Yld2000_UGent_ExptP6.11_deg8_FEdata_thetaPROFILE.txt`
and saves its plot in
`AA6016T4_Yld2000_UGent_ExptP6.11_deg8_FEdata_cpdrwPROFILE.png`

6. Future updates and developments

The current version of the code is exploratory in nature, but proves the essence of the concept: in conjunction with the calibration and evaluation schemes detailed in SSMD23, PolyN is a powerful modeling tool for metal plasticity. In addition, the PolyN-clone feature makes it possible to run simulations based on other yield surface models using PolyN-UMAT (which is likely to be faster than implementations based on the original formulations, particularly in the case of yield functions formulated in terms of principal values of transformed stresses). A potential development path is sketched below (support is welcome):

- Addition of new yield functions (that admit PolyN form) and implementation of their corresponding calibration algorithms; More options for the cup-drawing setup (e.g., current version allows only for fixed holder clearance; also, only S4R is used for meshing); Option for custom (user defined in UMAT) hardening law; Implement anisotropic hardening.
- Potential linkage with more (ideally non-commercial) FEA codes.
- Input interface and managing data (Handling input and data via text files is laborious and error prone; some GUI elements and a local `SQLite`-database to store material models are planned); Options for more customized plotting and reporting.
- Crystal plasticity module for fitting and generating data.
- Extension to 3D stress states.
- Exploration of the polynomial space via machine learning techniques with potential for optimization enhancements.