nnYS: A (second) python library for modeling and testing plane stress yield functions

(Usage instructions)

Stefan C. Soare / 2024-02-08

Content

1. Overview

This code upgrades the previous PolyN-scripts (`https://github.com/stefanSCS/PolyN`), by adding calibration procedures and FE-simulation capabilities for (artificial) neural network (NN) models of yield surfaces. The theoretical context for this upgrade is in the article *On the use of neural networks in the modeling of yield surfaces* (which will be referred to as `nnYStheory`). The repository for this code is: `https://github.com/stefanSCS/nnYS`

Currently, the code is structured on three problems:

(`PartA`) **NN-modeling of yield surfaces:** `python` notebooks.

(`PartB`) **Mapping the convexity domain of a yield function:** `python` notebooks.

(`PartC`) **FE simulation:** `abaqus-python` scripts and `abaqus-UMAT` subroutines.

(`PartD`) **Convexity:** `python` scripts for verifying the convexity of plane stress and 3D yield functions.

The notebook format used for `PartA` and `PartB` is quite self-explanatory and hence no further technical details on the calibration (training) of NN-models are required. The theory supporting `PartD` and usage instructions are detailed in Appendix-D. This document details only those usage aspects that are related to `PartC`: FE-simulations based on NN-models of the yield surface.

2. Tensile test and cup-drawing FE-simulations

Note: All files referenced in this section are located in the sub-directory /`PartC` of the repository.

• Exporting NN-model parameters

The first step consists in exporting the NN-parameters in a format suitable for input to `abaqus`. Notebooks `nb13_ExportWeights.ipyn` and `nb14_ExportWeights.ipyn` are provided as examples of how this is done. In essence, all NN-parameters are exported to a column vector, in the increasing order of the layers of the network, with the architectural parameters stored first. For example, the file `nb13_NNdata.txt` contains all the parameters of the NN-model `nb_13` in `nnYStheory`, as exported by `nb13_ExportWeights.ipyn`. The first few rows in `nb13_NNdata.txt` read:

```
3
20
8
20
2
```

```
0.02590182237327099
-0.8958836793899536
-0.8862651586532593
...............
```

where: the first row stores the number of layers (here: 3 = one input layer + two hidden layers); row 2 stores the number of nodes in the first layer; row 3 stores the homogeneity degree of the activation function of the first layer; row 4 stores the number of nodes in the second layer; row 5 stores the homogeneity degree of the activation function of the second layer; rows 6 and below store the weights (parameters) of the network.

The final input file for an `abaqus` simulation must also contain the (isotropic) elasticity and hardening parameters of the material. These properties are prepended to the file storing the NN-model parameters. For example, the (final) `abaqus`-input file `nb13_NNdataFEA.txt` corresponds to `nb13_NNdata.txt` and its first few lines read:

```
70000.000000000000
0.330000000000
478.200000000000
0.006800000000
0.289500000000
3
20
8
20
2
0.02590182237327099
-0.8958836793899536
-0.8862651586532593
...............
```

where: the first row stores the Young modulus E; row 2 stores the Poisson modulus ν; rows 3, 4 and 5 store the hardening parameters (Swift or Voce); rows 6 and bellow store the NN parameters as described above.

• Finite element simulations

The main simulation setting is the same as described in the document `PolyN_Usage.pdf` in the `PolyN` GitHub-repo. Here we present the details pertaining to the specifics of NN-models.

Simulations are performed using the `abaqus-python` scripts:

`abqSymmShell.py`

`abqSymmShell_mains.py`

The first script does the whole heavy lifting (generates the `abaqus-model` and runs the `abaqus-jobs`), while the second is simply an interface (driver) where the text file (and its location) storing the simulation parameters is specified. These two scripts must be stored in the working (root) directory as follows:

```
|-- abqNNys
    |-- DATA
        |-- AA6016T4
            abq_uniaxial_nnYS_AA6016T4.txt
    |-- RESULTS
        |-- AA6016T4
    abqSymmShell.py
    abqSymmShell_mains.py
    abqUMAT_Hill48.for
    abqUMAT_PolyN.for
    abqUMAT_nnYS.for
```

where:

abqNNys is the root (working) directory (can be any name)

DATA is the main sub-directory where all input data is stored and searched

RESULTS is the main sub-directory where all simulation output is saved

An example of driver script (abqSymmShell_mains.py) is as follows:

```
import abqSymmShell as abqp

##------- input data block

####  uniaxial tests simulations
fName='abq_uniaxial_nnYS_AA6016T4.txt'

####  or cup drawing simulations
#fName='abq_cpDraw_nnYS_AA6016T4.txt'

### specify the subDir of DATA for the input file 'fName'
subDir='AA6016T4'

####------------end of input data block

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

where:

- fName stores the name of the input file specifying simulation parameters (to be discussed next).

- subDir stores the sub-directory of DATA where the simulation input file is stored.

In the above snippet[a], the simulation input file is abq_uniaxial_nnYS_AA6016T4.txt and is stored in the /DATA/AA6016T4 sub-directory.

The abaqus-UMAT subroutine[b] implementing the NN-model of the yield function is abqUMAT_nnYS.for (the subroutines, abqUMAT_PolyN.for, implementing PolyN, and abqUMAT_Hill48.for, implementing Hill'48, are provided for comparison purposes).

Two types of simulations can be performed: uniaxial tensile tests and cup drawing (see PolyN GitHub-repo for further details). A sample simulation input file (abq_uniaxial_nnYS_AA6016T4.txt) is listed below.

```
## input file for abqSymmShell_mains.py

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

[a]In python, # indicates a comment: anything on the same line beyond # is discarded by the python interpreter.

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

```
dtMax:0.003
maxNInc:3000

## Hardening type: a string in ['Swift','Voce']
hLaw:Swift
## Elasticity parameters: (E,nuPoisson,muG)
eParam:70000.0,0.33,26200.0
## Hardening parameters: (A,B,C)
hParam: 478.200000000000,0.006800000000,0.289500000000

## for Hill'48 sims must provide additional details
## R-values required by Hill48: (r0,r45,r90) from experiments
rParam:0.526,0.253,0.601

## thickness(mm) of the metal sheet
hThick:1.0

## HRATIO is used to calculate the boundary displacement (see doc)
HRATIO:  10.0

##---------- 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)
##for example: with the default Hill48 of Abaqus, at test angle=0.0:
#sim: AA6022T4_0_HillAbaqus | False | False | 0.0

## or several tests (batch testing)
## for example, run tests with 'umatNNYS' and 'umatPoly'

#sim: nb13_NNdataFEA.txt | umatNNYS | True  | 0.0,15.0,30.0,45.0,60.0,75.0,90.0

sim: nb14_NNdataFEA.txt | umatNNYS | True  | 0.0,15.0,30.0,45.0,60.0,75.0,90.0

##sim: AA6016T4_TUAT_RV8zzz_deg8_FEdata.txt|umatPoly|True| 0.0
```

Most of the details of this file have been discussed in the previous `PolyN` repository. The main difference is on the format of the specification of a simulation:
`sim:  nb14_NNdataFEA.txt | umatNNYS | True | 0.0,15.0,30.0,45.0,60.0,75.0,90.0`
The above input line specifies that (from right to left):

- simulations of uniaxial tests (note that `uax` is set to `True`) are to be performed, at angles 0^o, 15^o, ..., 90^o from the rolling direction of the sheet
- the simulation uses `UMAT (indicated by the True setting)`
- the `UMAT` to be used is `umatNNYS` (which is a rename for `abqUMAT_nnYS.for`)
- the input file for the `UMAT` is `nb14_NNdataFEA.txt` (where the elastic-plastic and the NN-parameters are stored)

For a cup-drawing simulation the format of the simulation input file is similar. The main differences are in the type of input parameters (tools dimensions and friction) and in the specification of the simulation, which is shorter (since no test angles are required), e.g.:

`sim:  nb13_NNdataFEA.txt | umatNNYS | True`
indicating that (note that `uax` is set to `False` in this case):

- the simulation uses `UMAT (indicated by the True setting)`
- the `UMAT` to be used is `umatNNYS` (which is a rename for `abqUMAT_nnYS.for`)
- the input file for the `UMAT` is `nb13_NNdataFEA.txt` (where the elastic-plastic and the NN-parameters are stored)

Important: { Make sure the appropriate hardening law subroutine (Swift or Voce) is activated in `UMAT` (by renaming).

Finally, to run the `abaqus-jobs` specified in the simulation input file, open a command window and navigate to the root directory (which here is `abqNNys`) and execute at the prompt the following command:

```
abaqus cae noGUI=abqSymmShell_mains.py
```

Assuming all went well, simulation results are saved in the `/RESULTS/AA6016T4` directory. Post-processing of the above files can be performed by re-running the same driver script in `python` mode:

```
python  abqSymmShell_mains.py
```

The nature of the output files is described in the `PolyN_Usage.pdf` document in the `PolyN` repository.