Polycrystalline Finite Element Methods

Lukasz Kuna

Spring 2020

1 Mesh Generation

1.1 Neper

To generate the meshes used for polycrystalline meshes it is advised to use the Neper software package found at http://www.neper.info/.

Once installed a polycrystalline mesh can be made easily; a most basic example of tesselation generation is presented below:

```
neper -T -n 100 -id 1 -morpho gg
```

This generates a grain growth morphology cube of 100 grains. To mesh the generated blocks, the meshing module can be used:

```
neper -M n100-id1.tess
```

All of the options and settings for these modules in Neper are well documented in the manual which is included in the software package when installed.

1.1.1 Assigning Block IDs for MOOSE

As of the writing of this document, there is no way to assign block IDs using just Neper and MOOSE so this intermediate step has been added. It is slightly cubersome, but it works.

1. Import n100-id1.gmsh (Neper mesh module output) into GMSH (package can be downloaded at https://gmsh.info/)to change the format to one that can be read by Trelis (ABAQUS *.inp)
2. Import to Trelis/Cubit (https://csimsoft.com/trelis)
3. Add Block IDs to all volumes (Script Below)
4. Export mesh as Exodus

Adding the Block IDs to all the volumes can become tedious for meshes with more than five grains, so the following script can be used with Trelis/Cubit journal editor:

```
#!/usr/bin/env python2.5
import os, sys, math
# MAC Path
number_of_cell = int(input("Number of Blocks?"))
volume_id_list = cubit.get_entities( "volume" )
count = 0
for id in volume_id_list:
    count = count + 1
    cubit.cmd('block ' + str(count) + '  volume  ' + str(id))
```

2 Realistic Grain Orientation Generation

To generte realstic grain orientations the MATLAB toolbox MTEX can be utilized (found at https://mtex-toolbox.github.io/). A sample script used to generate cubic crystal structure orientations is posted below.

```
startup_mtex
prompt1 = 'How many random orientations are needed: ';
prompt2 = 'How many grains are there?  ';
n = input(prompt1);
grains = input(prompt2);
cs = crystalSymmetry('cubic');
% ss = specimenSymmetry('cubic');
x = n;
while x > 0
    ori = orientation.rand(grains,cs);
    filename = ['cubic_', num2str(grains), '_', num2str(x), '.txt']
    fname = fullfile(mtexDataPath, 'ODF', filename);
    export(ori, fname, 'Bunge')
    x = x-1;
end
```

3 Generating MOOSE Inputs

Lastly, the orientations generated by MTEX must be utilized in a MOOSE input file. This is done using the python script included within the polycrystalline methods directory (see *alumina_inputs_generator.py*). Note the following:

1. The location of the txt file containing the orientation data as output by MTEX is specified by line 16:

   ```
   for line in open("/Users/lukaszkuna/projects/ferret/trigonal_grains.txt"):
   ```

2. The input file generated by this script includes

   (a) Elasticity [ComputeElasticityTensor]

   (b) Piezoelectricity [ComputePiezostrictiveTensor]

   (c) Optics [ComputeIndicatrix]

   (d) Elasto-Optics [ComputeElastoopticTensor / ComputeDeltaIndicatrix]

   (e) Electro-Optics [ComputeElectroopticTensor / ComputeDeltaIndicatrixElectro]

3. Sidesets are generated from normals via MeshModifiers:

   ```
   [MeshModifiers]
       [./add_sidesets]
         type = SideSetsFromNormals
         normals = '1  0  0
                   -1  0  0
                    0  1  0
                    0 -1  0
                    0  0  1
                    0  0 -1'
         fixed_normal = true
         new_boundary = 'right left front back top bottom'
         variance = 0.5
       [../]
   []
   ```

4. With a slight modification this script can be used to rapidly generate many input files each with a unique set of orientations as generated by MTEX. This allows for the use of smaller meshes (grains < 200) with many quick calculations as opposed to many grain meshes (grains > 300).