Texture Analysis with MTEX inside Matlab

For 27-750
Texture, Microstructure and Anisotropy
A.D. Rollett

In-Class Questions

• What is “texture”? Texture quantifies any bias in crystal orientations with respect to a reference frame associated with the specimen shape, or equivalent. This is extended to grain boundary (GB) texture or preference for one type of GB over another. GB texture can be present even when the orientation texture is random.

• Why do we have to work with general 3D rotations in order to quantify texture? General 3D rotations are required because crystals have a 3D basis set for their crystal structure and three parameters are required to relate the (three) associated axes with the reference frame (however that is chosen).

• How does a rotation specified by a combination of axis and angle correspond to a set of three Euler angles (flippant answer: not very easily!)? The most straightforward correspondence is via the 3x3 rotation matrix; conversions between Euler angles and matrix can be found e.g. in Bunge’s book, similarly for axis+angle.

• How does texture apply to CdTe? The most relevant aspect is grain boundary texture, although some deposits also have a very strong orientation texture.
In-Class Questions: 2

• How does misorientation apply to CdTe (or any other polycrystalline material)? Misorientation is the term used to describe the difference in orientation, generally across a grain boundary (but between any pair of orientations in general).

• How is crystal symmetry taken into account in texture and grain boundary analysis? The symmetry of a crystal defines a point group; that point group must be used to calculate the minimum set of orientations or misorientations that are physically possible.

• What is the procedure that one can follow to use Matlab+MTex to construct an orientation distribution from pole figure data? This procedure is described in these lecture notes.

• What is the procedure that one can follow to use Matlab+MTex to construct an orientation distribution from EBSD data? As with pole figures, this is described in the notes. In general, it is reasonable to use the procedure provided by the MTEX package.

• What else can one obtain from a Matlab+MTex analysis? In principle, one can perform any analysis although some degree of exploration may be required to implement any given type of analysis.
Objectives

1. Introduce students to texture i.e. crystallographic preferred orientation, also known as fabric in geology.
2. Explain briefly two standard types of data, namely pole figures and orientation maps from e.g. electron back-scatter diffraction (EBSD).
3. Demonstrate how to use the MTEX package within Matlab to perform analyses on relevant data.
**Grain Definitions**

- Given an orientation map with continuously varying orientations from one point to the next, one imposes a grain structure by aggregating points with similar orientation. This requires choice of a threshold in misorientation, below which two adjacent points belong to the same grain. From percolation, the procedure is known as the “burn algorithm”.

**Examples of varying the misorientation threshold:**

3 degrees

15 degrees

Note that each color represents 1 grain
Variation in Boundary Properties

- Rhetorical question: do grain boundaries have variable properties? Answer: yes, even GB energy is highly variable.
- GB energy depends primarily on the two surfaces that are joined at the boundary. Low energy surfaces result in low energy GBs. In fcc metals, for example, \{111\} is the close packed (highest atomic density) surface and therefore lowest energy. GBs with \{111\} surface(s) are demonstrably low energy. Thus the coherent twin has the lowest energy of all, and \{111\} twist GBs are generally lower than most other GB types.
- GB energy correlates (positively) with excess free volume (per atom).
- Other properties also correlate with excess free volume such as diffusion.
- In grain growth, high energy GBs disappear faster than low energy GBs; therefore populations of GBs are anti-correlated with energy.
- GB mobility is poorly understood: some GBs exhibit anti-Arrhenius behavior (mobility decreases with increasing temperature).
Installation of MTex

• Find MTEX by searching on “mtex google code”; the author of MTEX, Ralf Hielscher, now hosts the code on his own website, [http://mtex-toolbox.github.io/](http://mtex-toolbox.github.io/). As of Jan. 19th 2016, this is still valid.

• MTEX has its own installation procedure. As detailed in the instructions found on-line, the steps include
  a) download the package (from the “downloads” page)
  b) set the Matlab “path” to the folder/directory where the MTex package is located (see next slide for screenshots); I put mine in /Users/Shared (on a Mac).
  c) in the Matlab command window, type “startup_mtex”.

• Then click on “Install MTEX for future sessions” (no longer required)

• Fortunately, this takes care of replacing any previous, older installations of MTex.

• The Matlab documentation now includes documentation on MTex.

• Caution: if you manually add the MTEX folder to your MATLAB path then there is a significant risk that MTEX will give errors (e.g. when you try to read in EBSD data). What you should do instead to get it to work is to put MTEX in a different folder and run startup_mtex from that directory.
Change Folder

Hold the mouse arrow over this icon and you will see “Browse for folder”

Or this: “Set Path”
What can MTex do?

• We will explore two things:
  a) analysis of pole figure data;
  b) analysis of EBSD data.

• Useful links:
  This one describes ways to plot individual orientations.
  http://merkel.zoneo.net/rdx/index.php?n=Texture.PlotIndividualOrientationsInMTex
  This one has useful hints (although be careful that many details have changed from v3 to v4):
Navigating the file structure

>> pwd
will tell you which directory you are in, probably “~/Documents/MATLAB”

>> cd /directory-of-your-choice
will place you in whatever folder/directory you like, i.e. where you have your data. Note, on a Mac, you can Copy a highlighted folder, Paste in Terminal, highlight and copy the full path that you see, and then Paste that after the “cd “ entry in the Matlab window.

You can then click on “Import pole figure data” or “Import EBSD data”, for example, to set up the script to read in your data. Or (next slide) you can type “import_wizard”.
Part 1: PF analysis

- Look in the “Mtex Toolbox” for “Short Pole Figure Analysis Tutorial”. Do not use this!
- Instead, type `import_wizard` or click on that in the list of links. Note, if you have just started MatLab (after installing MTEX in a previous session), you need to navigate (Change Folder) to the MTEX folder and type “`startup_mtex`” before doing anything else.
- A new, small window will open. It should be set to the “Pole Figures” tab by default (but if not, click on that tab). Click on the “+” and navigate to where you have “alr.epf” stored. You should see the window below.
Click on “Next >>” in the same window. You can enter the lattice parameter as 4.05 if you wish (although it should not make any difference because this is cubic). The wizard is quite clever: if you enter “Fe” or “Iron” as the mineral name, it will recognize this material and make the appropriate entries for you.
PF analysis – p3

• I prefer to set the plotting axes so that “x” points to the right (East), as for normal plots. For now re-set the “specimen symmetry” to “orthorhombic”.

There is now a control to enforce plotting with X to the right (east): plotx2east
PF analysis – p4

• After clicking “Next >>”, you should see this window.
PF analysis – p5

- After clicking “Next >>”, you should see this window (but with “orthorhombic” for the sample symmetry). Now click “Finish” to be done.
Plot experimental PFs

- `Plot(pf)` should show you discrete pole figures in which the color is red for high intensity and blue for low; “pf” was the default name of the PF data that you read in.
**ODF analysis**

- At this point you can run the ODF analysis:

  odf = calcODF(pf)

----- MTEX -- PDF to ODF inversion ---------------
Call c-routine
initialize solver
start iteration

6.2644E-02 6.0870E-02
Finished PDF-ODF inversion.

error: 6.0870E-02
alpha: 1.0518E+00 1.0678E+00 1.1113E+00
required time: 6s

odf = ODF (show methods, plot)

comment: ODF recalculated from /Users/rollett/Word/teaching/Micro14/MatLab_Casper/Al-PFs/alr.epf

crystal symmetry: aluminum (m-3m)
sample symmetry: orthorhombic

Radially symmetric portion:
  kernel: de la Vallee Poussin, hw = 5°
  center: 1232 orientations, resolution: 5°
  weight: 1
Recalculated PFs

- Type “plotPDF(odf,h,'antipodal’)” to plot the same set of pole figures but based on the ODF. Note that these are quadrant PFs because of the assumed orthorhombic sample symmetry.
**Adjust sample symmetry**

- Now go into the script that you had generated for the PF import, and change (by editing it) the sample symmetry from orthorhombic to triclinic.
Re-analyze

• Re-run the calcODF and plotpdf commands.
Inverse PFs

- To get a complete set of 3 inverse pole figures, type “plotIPDF(odf,[xvector,yvector,zvector],'antipodal’)”. The 3 sample directions (X, Y & Z) are specified by the built-in “xvector”, “yvector” and “zvector”. You may need to use Annotate to label the corners.
Sections through ODF

- plot(odf,'PHI2','sections') will give you plots of sections through the ODF. These go to 360° in phi1, because of the triclinic sample symmetry.
3D view

- "plot(odf,'PHI2','surf3')" – gives rotatable view.
>> textureindex(odf)
ans =
    10.6263
>> entropy(odf)
ans =
    -1.6371

• These values suggest a moderately strong texture.

• The section on “Characterizing ODFs” provides a few other techniques.
Misc., tricks

- Inside a script (*.m file), you can run a portion of it by positioning the cursor at the end of the line where you want to go to, and hit “Control-Enter” to run the script to that position.
Errors

%Error analysis:
%For a more quantitative description of the reconstruction quality one can use the function calcError to compute the fit between the reconstructed ODF and the measured pole figure intensities. The following measured are available:

   RP - error ;   L1 – error;   L2 – error

calcError(pf,odf,'RP',1)
ans =
   0.1540  0.1631  0.1319  0.1033  0.1163  0.1763  0.1734

%In order to recognize bad pole figure intensities it is often useful to plot difference pole figures between the normalized measured intensities and the recalculated ODF. This can be done with the command PlotDiff.
plotDiff(pf,odf)
**Volume fraction**

First we specify an texture component using “orientation”:

\[
\text{ori} = \text{orientation}('Euler',\phi_1, \Phi, \phi_2, \text{cs}, \text{ss})
\]
e.g.  \[
\text{center} = \text{orientation}('Euler',0, 55, 45, \text{CS}, \text{SS})
\]

The function 'volume' returns the ratio of an orientation that is close to an orientation (center) by a misorientation tolerance (radius) to the volume of the entire odf.

**Syntax:**

\[
v = \text{volume}(\text{odf}, \text{center}, \text{radius}, \text{<options>})
\]
Part 2: EBSD input

- Navigate with “cd” to wherever your data is; in this case, we will work with one or more of the CdTe datasets provided by Prof. Zaefferer’s group. You are recommended to place it in a folder/directory by itself so that you can store the images from running Matlab+MTEX. On the Macs, “Grab” is very handy for screen/window captures.

- Click on “Import EBSD data”, just as you did for the pole figure data and follow the steps to specify the material etc. **Make sure that you have CdTe.cif in the mtex-4.0.5/data/cif/ folder so that mtex knows about CdTe as a material.** It will very likely ask you about both the CdTe phase and about an unindexed phase. This latter is not a problem.

- This generates a dataset called “ebsd”. You can change this name in the script that was generated in the above step.

- Note that the mtex folder will have a version number (e.g. the 4.0.5 above) in the name but this version number may vary depending on when you downloaded the package.
EBSD: determine, plot grains

- We take input from: `monomodGrainSize+Twins.ang`
- Type "import_wizard" to get the interactive window for importing data.
- Click on the “EBSD” tab
- Navigate to the “monomodGrainSize+Twins.ang” file. The data will then be imported.
- Click “Next” several times to see the program recognize the phase(s) automatically until only “Finish” is available.
- You will then have a MatLab script in its own window. Save it under whatever name you prefer (you will see something like “mynname.m” appear, in whichever folder/directory you choose).
- Click on “Run” (green arrow, pointing right). This will execute the script and import the dataset as an entity called “ebsd”.
- Check the contents of the data by just typing the name “ebsd”. If you do not see the phase “Cadmium Telluride” appear in the list (just a blank list) then edit the script to remove “not indexed” as a phase. In other words,
  ```matlab
  % crystal symmetry
  CS = {...
      crystalSymmetry('432', [6.41 6.41 6.41], 'mineral', 'Cadmium Telluride', 'color', 'light blue'),...
  'not indexed');
  should become:
  % crystal symmetry
  CS = {...
      crystalSymmetry('432', [6.41 6.41 6.41], 'mineral', 'Cadmium Telluride', 'color', 'light blue'));
  ```
- Then repeat the importation by clicking on Run again, and again type “ebsd”. You should see this:
  ```matlab
  >> ebsd
  ebsd = EBSD (show methods, plot)
  Phase  Orientations         Mineral       Color      Symmetry      Crystal reference frame
        0  290700 (100%)  Cadmium Telluride  light blue  432
  Properties: ci, iq, sem_signal, x, y
  Scan unit : um
  ```
- This slightly odd procedure is required in order to account for the fact that MTEX gets confused if you tell it to look for both a real phase and non-indexed points when, in this particular dataset, every single point is indexed and belongs to the Cadmium Telluride phase. As of version 4.0.5, this problem appears to be fixed.
**EBSD: determine, plot grains**

- Type "grains_FMC = calcGrains(ebsd('Cadmium Telluride'),'FMC',… 3.5)" (takes a while to compute)
- Then compute inverse pole figure coloring with:
  \[ oM = ipdfHSVOrientationMapping(ebsd('Cadmium Telluride')) \]
- Then plot the ipdfHKL map with
  \[ \text{plot(ebsd('Cadmium Telluride'),oM.orientation2color(ebsd('Cadmium… Telluride').orientations))} \]
  and compute grain boundaries:
  \[ gB=grains_FMC.boundary… ('Cadmium Telluride') \]
  hold on
  \[ \text{plot(gB,'linewidth',2.)} \]
- This will provide a map with the GBs delineated (and “linewidth” controls the thickness).
First, do this:
plot(ebsd)
This should be “boring” i.e. it should show only 1 phase.

odf = calcODF(ebsd('Cadmium Telluride').orientations)
. OR.

odf = calcODF(ebsd('Cadmium Telluride').orientations,'halfwidth',5*degree)

The second should show a few lines of output with details about the calculation.
The third is the calculation of the ODF (in v4, using “psi” as the 3rd argument does not work, hence the numerical value).
**Plot ODF**

\[ h = [\text{Miller}(1,0,0,\text{CS}\{1\}), \text{Miller}(1,1,0,\text{CS}\{1\}), \ldots, \text{Miller}(1,1,1,\text{CS}\{1\})] \]

This defines a set of pole figure indices.

**CAUTION:** the above assumes you specified two phases in the input script, so check your input script as to which phase is the one that you want. For example, you may need to replace \"{1}\" by \"{2}\" if the phase of interest is the second one.

\[ \text{plotODF}(\text{odf},h,'\text{antipodal}') \]

This shows a set of sections based on the calculated ODF.
**ODF plot**

plotODF(odf) shows the same sections but in tableau style.
Pole figures

Try this first: `plotPDF(odf,h,'antipodal')`
This longer version should equalize the contour levels:
```matlab
plotPDF(odf,h,'antipodal','minmax', 'off', 'contour', 0:0.5:4)
colorbar
set(gcf,'PaperPositionMode','auto');
print(w1,'-dpng','filename');
```
This shows a set of PFs based on the calculated ODF. As you can see, the texture is nearly random. Make sure that MatLab is pointed to your local directory/folder with the data.
Discrete PFs

• This command will show discrete pole figure plots, where the argument “500” controls how many individual orientations are included in the plot:
  \[
  \text{plotPDF(ebsd('Cadmium… Telluride').orientations,h,'points',500,'antipodal')}
  \]
Inverse Pole Figures (IVPs)

plotlPDF(odf,[xvector,yvector,zvector],'antipodal','minmax', 'off',…
 'contour', 0:0.25:2)
>> hold on
>> colorbar
Note how the 001 inverse pole figure shows a maximum at the <111>
position. Annotate can be used to label the corners.
Grain Boundaries

To compute the grain boundaries:

\[ gB = \text{grains\_FMC('Cadmium Telluride').boundary} \]

Then to get a map of just the GBs:

\[ \text{plot}(gB) \]

To map GBs by misor angle [note that in v4.0.4, one has to add a double phase specification]:

\[ \text{plot}(gB('Cadmium Telluride', 'Cadmium Telluride'), gB('Cadmium Telluride', 'Cadmium Telluride').misorientation.angle/degree, 'linewidth', 1.5) \]

Note that you restrict the GBs examined to a particular phase by specifying the two phases in the command about GBs that you are using, just as you see here.
You can make a basic analysis of the grain boundaries in your map by making a histogram of the misorientation angles.

```
plotAngleDistribution(gB ('Cadmium Telluride', 'Cadmium Telluride'));
```

To export the figure directly, here’s a new trick. You can save the figure as a MatLab object:
```
w1 = figure(1);
```
Then you export it in, say, PNG format thus:
```
set(gcf, 'PaperPositionMode', 'auto')
print(w1, '-dpng', 'path_to_your_directory');
```
The clumsy thing here is the need for the complete pathname. This can be structured by saving the path to the relevant directory in a MatLab variable and then concatenating with an actual (local) filename.
To see gradients in orientation, one can calculate and then map thus:

```matlab
kamvalues = KAM(ebsdmonotwins,'threshold',5*degree);
plot(ebsd,ebsd.KAM./degree);
```
to control the color range
```matlab
setColorRange([0,1]) ;
```
And include a color bar (legend):
```matlab
colorbar
```
And then capture the figure thus:
```matlab
w2=figure(1);
```
this ensures that the framing of the figure is correct (tends to cut off large aspect ratio figures unless you do this):
```matlab
set(gcf,'PaperPositionMode','auto')
```
```matlab
print(w2,'-dpng',
'path_to_your_directory');
```
KAM Misorientation with GBs
Grain Boundary types

As an example, let’s emphasize sigma-3 boundaries; first we have to specify the boundary type as an axis-angle pair.

```matlab
rot = rotation('axis',vector3d(1,1,1),'angle',60*degree);
gB = grains_FMC.boundary('Cadmium Telluride','Cadmium Telluride')
ind = angle(gB.misorientation,rot)<10*degree;
```

Then to make the map:

```matlab
close all
plot(gB)
hold on
plot(gB(ind),'linewidth',1.5,…
'linecolor','r')
```

Plot shows general GBs in black and twin boundaries in red selected as $\Sigma 3$

[Works as of 5-Nov-14 in v 4.0.5]
Misc.

- These notes from the workshop in Belo Horizonte, Brasil, Sept. 2015.
- Look for a folder called workshops-BeloHorizonte-2015.1 with “15-brasil” inside, with “matlab” inside that.
- There are example scripts in the folder, each of which contains many procedures to do various things.
Extra for Grain Boundaries

• Note that you restrict the GBs examined to a particular phase by specifying the two phases in the command about GBs that you are using, just as you see here.

• The general Matlab command “struct” will show you what’s inside a variable/container.

• To smooth grain boundaries (and you can use a different name to avoid over-writing the previous, unsmoothed dataset):
  
  \[ \text{grains} = \text{smooth} (\text{grains}) \]

• To get the list of boundaries in the different phases:
  
  \[ \text{grains.boundary} \ [\text{I think } \ldots] \]

• (Note that “\text{grains}” is the name of your current set of grains and boundaries) The coordinates of the segments (F for segment) are in a list like this where the 1\text{st} column is the 1\text{st} vertex and 2\text{nd} column is the 2\text{nd} vertex (at the end vertex); the list is as long as the number of segments:
  
  \[ \text{grains.boundary.F} \]

• The actual coordinates of each vertex (V for vertex) is given by
  
  \[ \text{grains.boundary.V} \]

• If you type the following, you get the two IDs on either side of the boundary segment:
  
  \[ \text{grains.boundary.grainID} \]

• The index into the orientations of the adjacent pixels can be found thus:
  
  \[ \text{grains.boundary.edsdID} \]

• The list of point/pixel orientations:
  
  \[ \text{Edbsd.rotations} (\text{grains.boundary.edsbID}) \]
Homework

• The homework is simple to state but will require thought, in addition to processing the datasets with MTEX:

Four datasets are provided. Investigate the differences between them and report on what you (or your group) believe those differences are.
Summary

• The sequence provided up to this point illustrates how to read in and perform standard analysis on pole figures and EBSD maps.