MATLAB and the MTEX Toolbox for Texture Analysis

Nuclear Materials Research Group

Travis Skippon and Chris Cochrane

Department of Mechanical and Materials Engineering
Queen's University

Background

MATLAB is a numerical computing environment and programming language, by MathWorks It is designed to rapidly perform vector analysis and matrix algebra. Due to some specific design choices, it is a relatively straightforward platform for performing numerical calculations on large datasets.

MTEX is a free MATLAB toolbox for the analysis of crystallographic texture. It is primarily designed for analysis of EBSD data. The project is maintained by Dr. Ralf Hielscher, of the Technische Universitat Chemnitz. The documentation is quite extensive: https://mtex-toolbox.github.io/

Why do you care?

Many problems related to crystallographic texture, such as orientation analysis and grain statistics, are massively simplified by using the appropriate tool.
Very large datasets or transitions between reference frame or crystal structure can be handled by MATLAB and MTEX, allowing you to focus on the science instead of the calculation.
This presentation is meant to provide some background to help you get started, and give you an idea of what kinds of problems MTEX can help you solve.

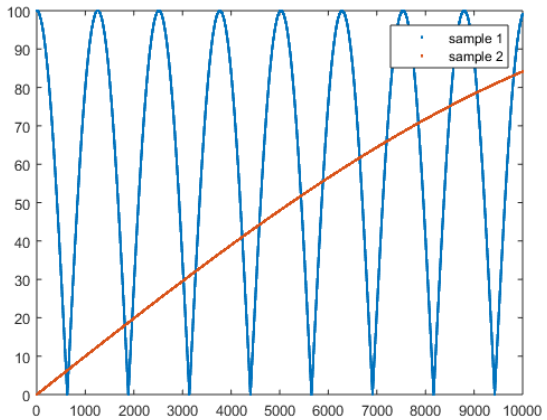
Basic Datatypes in MATLAB

```
%% Scalars:
x=2
x*3
```

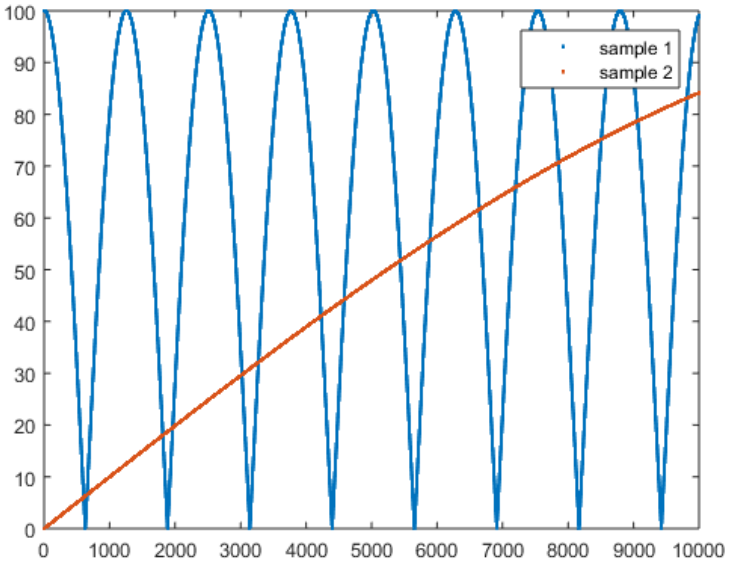
```
%% Vectors/matrices:
x=[1 2 3; 4 5 6; 7 8 9]
x(1,1) %returns 1st row, 1st column
x(1,1:3) %returns 1st row, columns 1 through 3
x(1,:) %returns the entire first row
x(1,3)= x(2,end) %assigns a value to position (1,3)
```

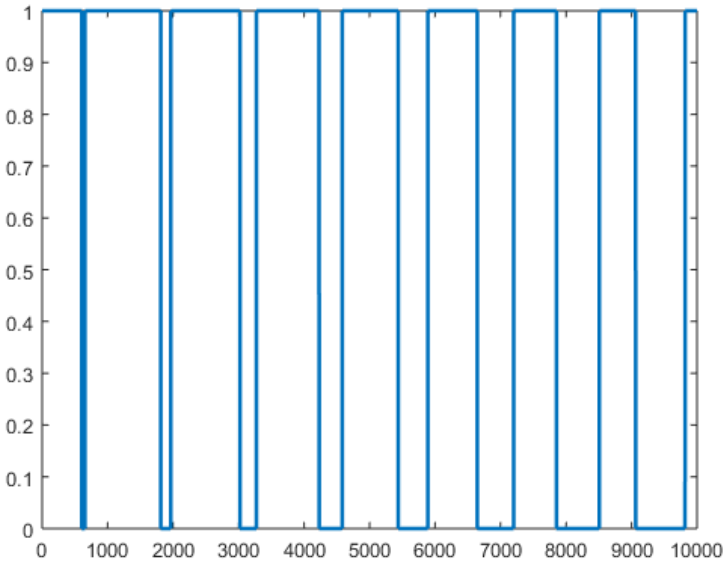
Conditional Indexing: Your New Best Friend

```
time=[1:10000];
sample1=abs(100*cos(time/400));
sample2=100*sin(time/10000);

figure(1)
plot(time,sample1,'.',time,sample2,'.')
legend({'sample 1' 'sample 2'})
```



Conditional Indexing





Conditional Indexing

What if we are only interested in the data where Sample 1 is greater than Sample 2?

With Conditional Indexing

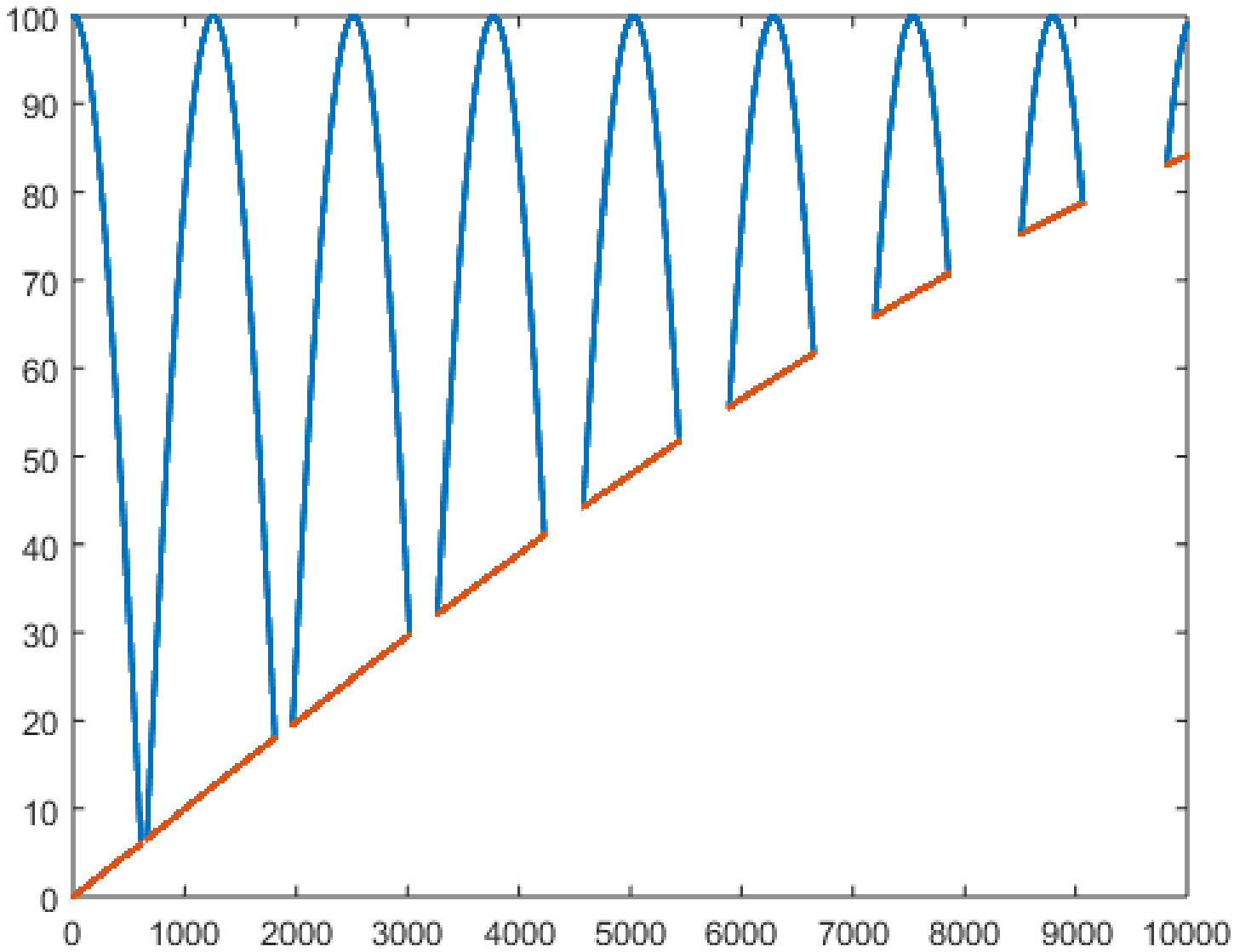
```
x=time(sample1>sample2);
y=sample1(sample1>sample2);
y2=sample2(sample1>sample2);
```

Run time: 0.00075 seconds

Without Cond. Ind.

```
j=1;
for i=1:10000
    if sample1(i)>sample2(i)
        xif(j)=time(i);
        yif(j)=sample1(i);
        yif2(j)=sample2(i);
        j=j+1;
    end
end
```

Run time: 0.03 seconds


100
90
80
70
60
50
40
30
20
10
0
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000

Structures and Properties

You can also create structures that contain many different types of data. This is useful in some situations and is used extensively by MTEX.
Here we set up a structure called Sample. Each sample has 4 properties: material, type, data, and rate.

```
sample.material='Zircaloy-2';
sample.type='tensile';
sample.data=0:0.2:5;
sample.rate=1e-4;
```

Structures and Properties

Like regular variables, structures can be made into arrays. Let's add a second entry to sample:

```
sample(2).material='Excel';
sample(2).type='compression';
sample(2).data=0:-0.2:-5;
sample(2).rate=1e-5;
```

Structures and Properties - Indexing

We can use conditional indexing on structures, too!
Let's say we wanted to find samples that had a rate of 1e-5

```
x=sample([sample.rate]==1e-5);
```

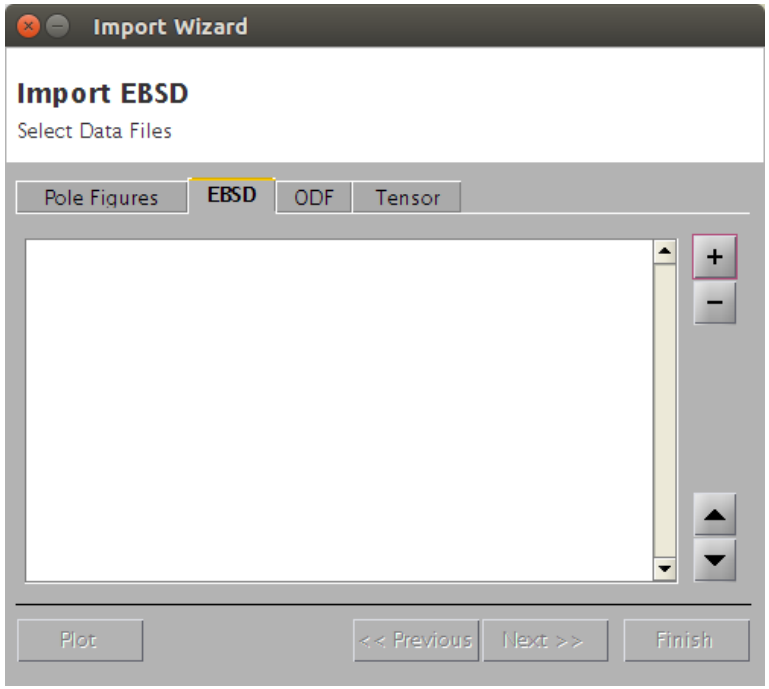
Note that we need to use square brackets to turn our structure output into something that can be directly compared to a value. This is just an odd Matlab quirk.

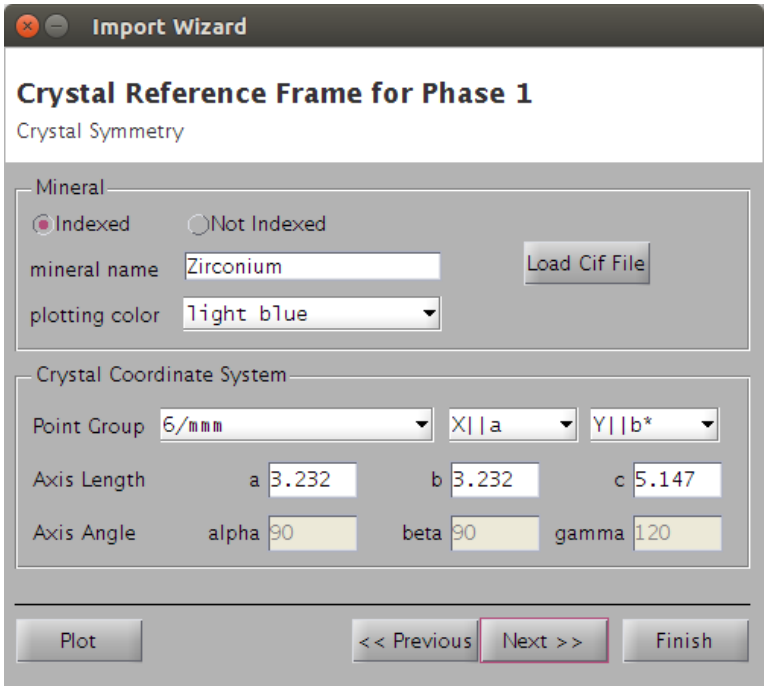
Working with EBSD Data

One of the most most frequent applications of MTEX is analysis of EBSD data.

```
> import_wizard('EBSD')

> ebsd = loadEBSD(fname,CS,'interface','ctf',...
  'convertEuler2SpatialReferenceFrame');
```

EBSD - Import Wizard





Ebsd Data is stored in a *structure* that has many *properties*. If you type the name of your ebsd structure, MATLAB will output a summary.

```
> ebsd
ebsd = EBSD (show methods, plot)

 Phase   Orientations          Mineral   Color   Symmetry   Crystal reference frame
     0     24905 (8%)      notIndexed
     1   213854 (69%)       Zirconium   light blue      6/mmm   X||a*, Y||b, Z||c
     2    71441 (23%)   ZirconiumBeta   light green       m-3m

 Properties: bands, bc, bs, error, mad, x, y
 Scan unit : um
```

Only some of the properties are listed. To see a list of all of them, type in "ebsd." (without quotes) to the command line and press the tab key. MATLAB will open a list of properties available. This works for most structure types in MATLAB.

EBSD Data - Indexing

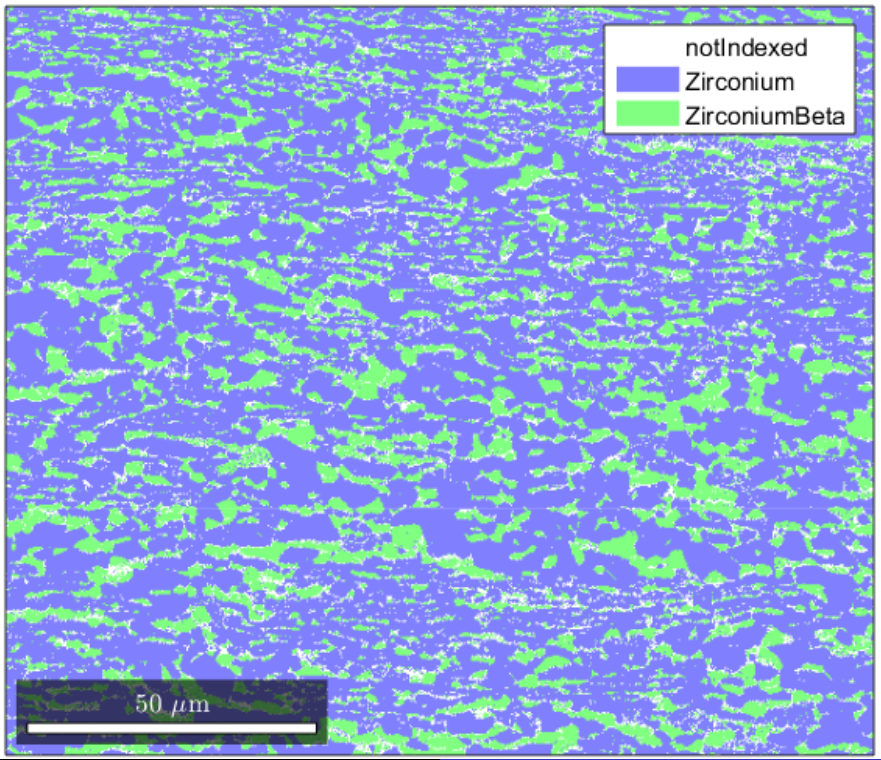
```
> ebsd('Zirconium')

ans = EBSD (show methods, plot)

 Phase    Orientations      Mineral          Color  Symmetry  Crystal reference frame
     1  213854 (100%)   Zirconium     light blue      6/mmm          X||a*, Y||b, Z||c

 Properties: bands, bc, bs, error, mad, x, y
 Scan unit : um
```

```
> ebsd(ebsd.phase==1)

ans = EBSD (show methods, plot)

 Phase    Orientations      Mineral          Color  Symmetry  Crystal reference frame
     1  213854 (100%)   Zirconium     light blue      6/mmm          X||a*, Y||b, Z||c

 Properties: bands, bc, bs, error, mad, x, y
 Scan unit : um
```

Plotting EBSD Data

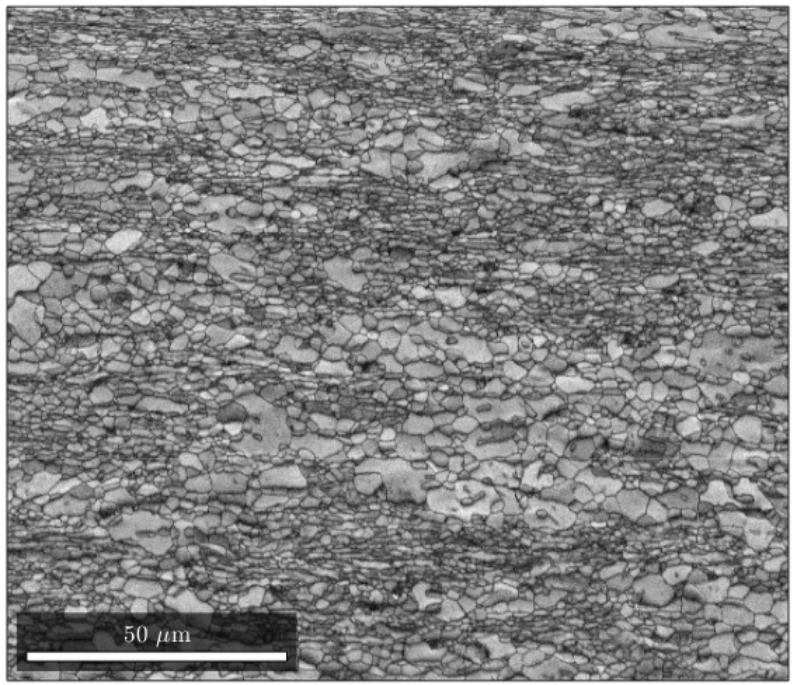
By default, MATLAB plots EBSD data by phase. If you have a single phase material, this means you won't see much.

```
> plot(ebsd)
```



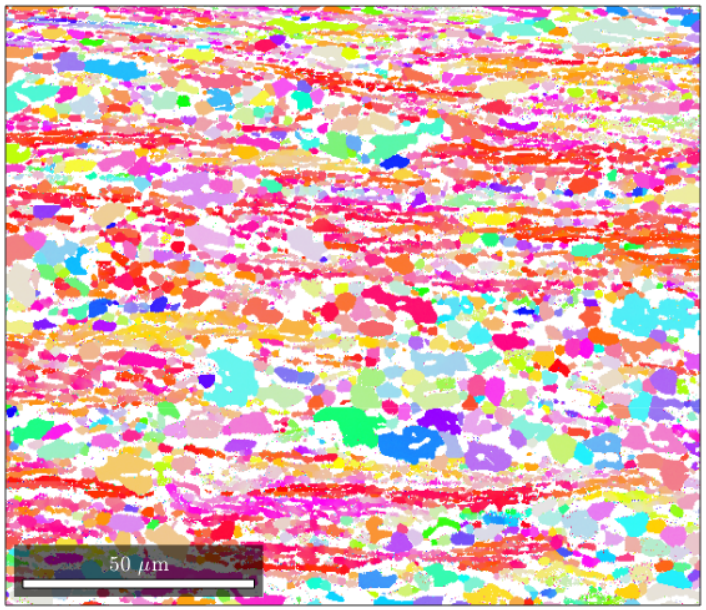
There are many other things we might be interested in plotting from EBSD data.
Here is how to plot Band Contrast.

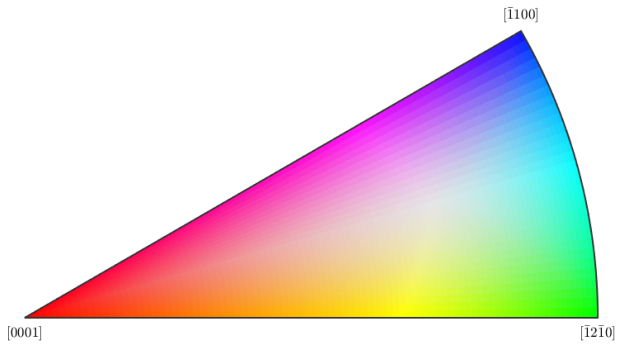
```
figure(1)
plot(ebsd,ebsd.bc)
colormap('gray');
```



We can plot orientations, but only for one phase at a time: Remember, you can specify a phase within brackets when calling the ebsd structure. Here is some code that will plot an inverse pole figure showing how the colors relate to orientation

```
figure(2)
plot(ebsd('Zirconium'),ebsd('Zirconium').orientations)
figure(3)
oM=ipdfHSVOrientationMapping(ebsd('Zirconium').orientations);
plot(oM)
```

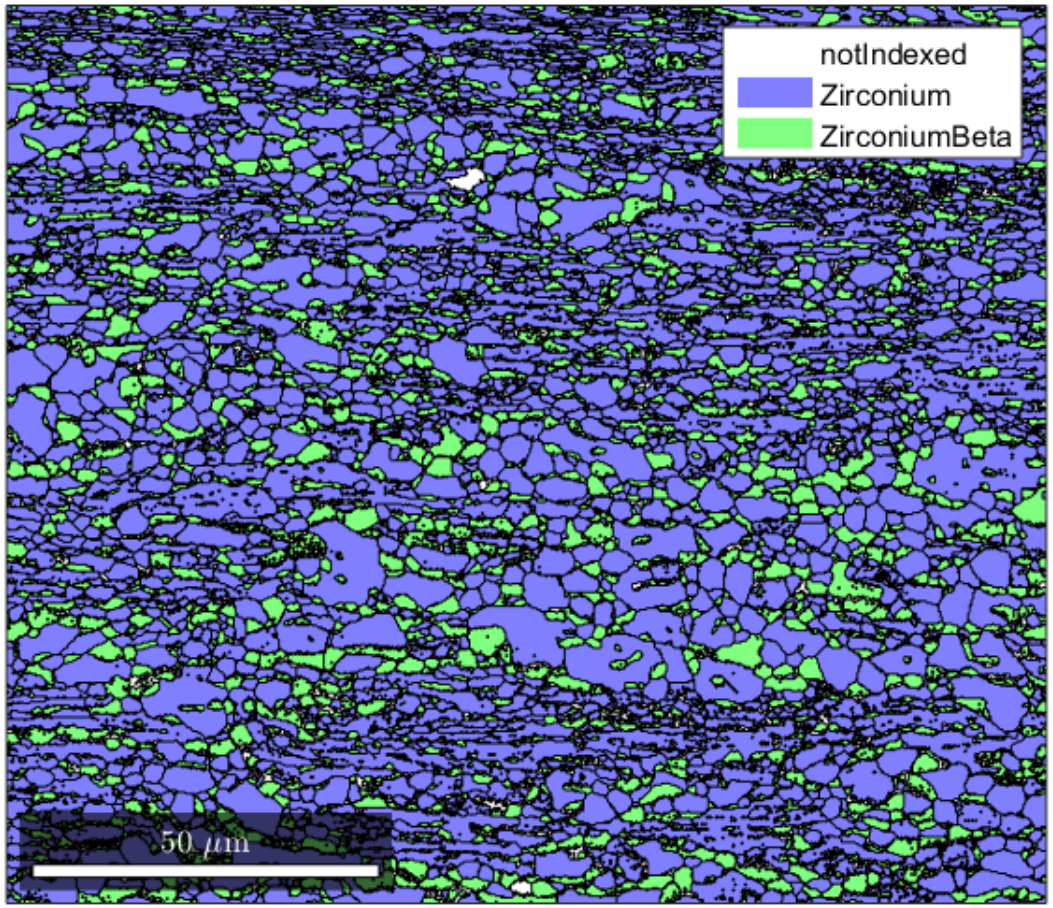




Grain Reconstruction

MTEX can "reconstruct" grains from ebsd data. calcGrains takes in ebsd and will define a grain boundary as any point where there is a change in orientation of 5 degrees or more. It outputs a list of grains, and gives the ebsd structure new properties called grainId, and mis2mean. GrainId tells which grain each pixel in the map belongs to, and mis2mean is the misorientation of each pixel to the mean orientation of its grain.

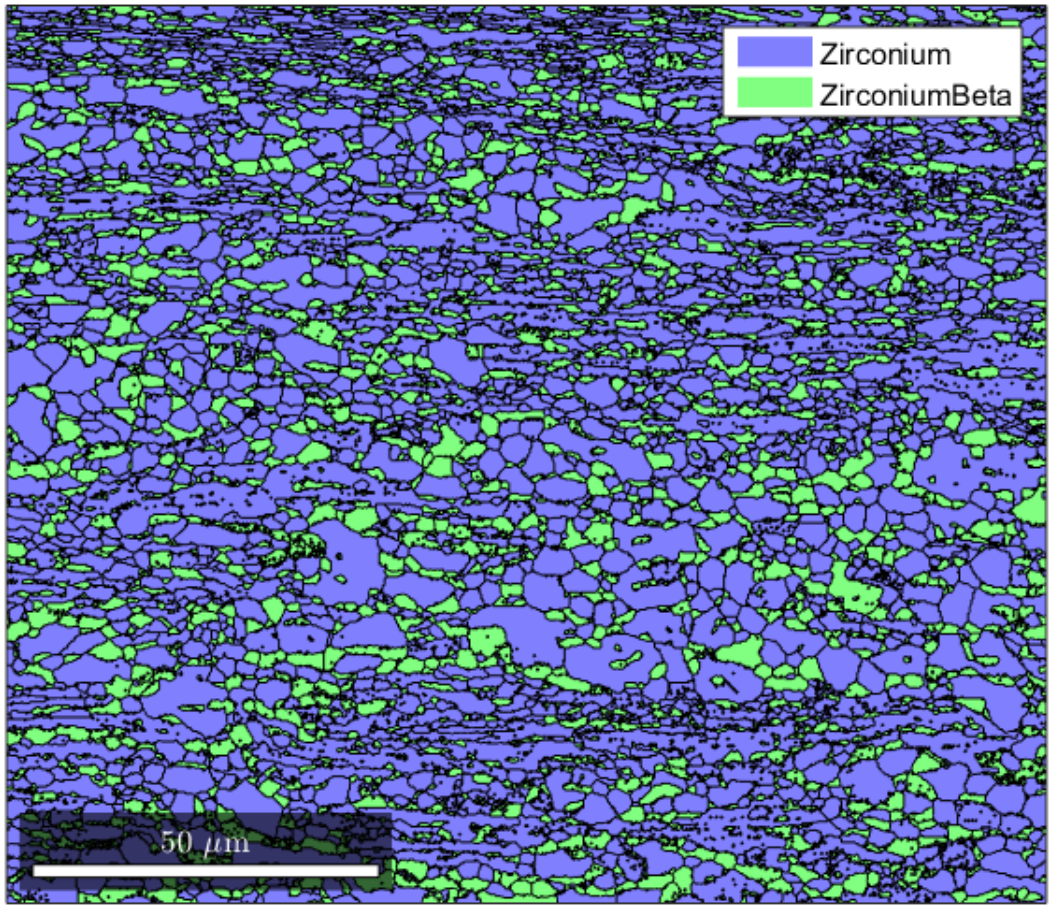
```
[grains, ebsd.grainId, ebsd.mis2mean] = ...
    calcGrains(ebsd, 'angle', 5*degree);
plot(grains)
```


notIndexed
Zirconium
ZirconiumBeta
50 μm

MTEX has treated non-indexed areas as their own grains. If you'd instead like MTEX to simply absorb non-indexed points into grains formed by indexed points, you can issue this command instead:

```
[grains, ebsd.grainId, ebsd.mis2mean] = ...
  calcGrains(ebsd('indexed'), 'angle', 5*degree);
```

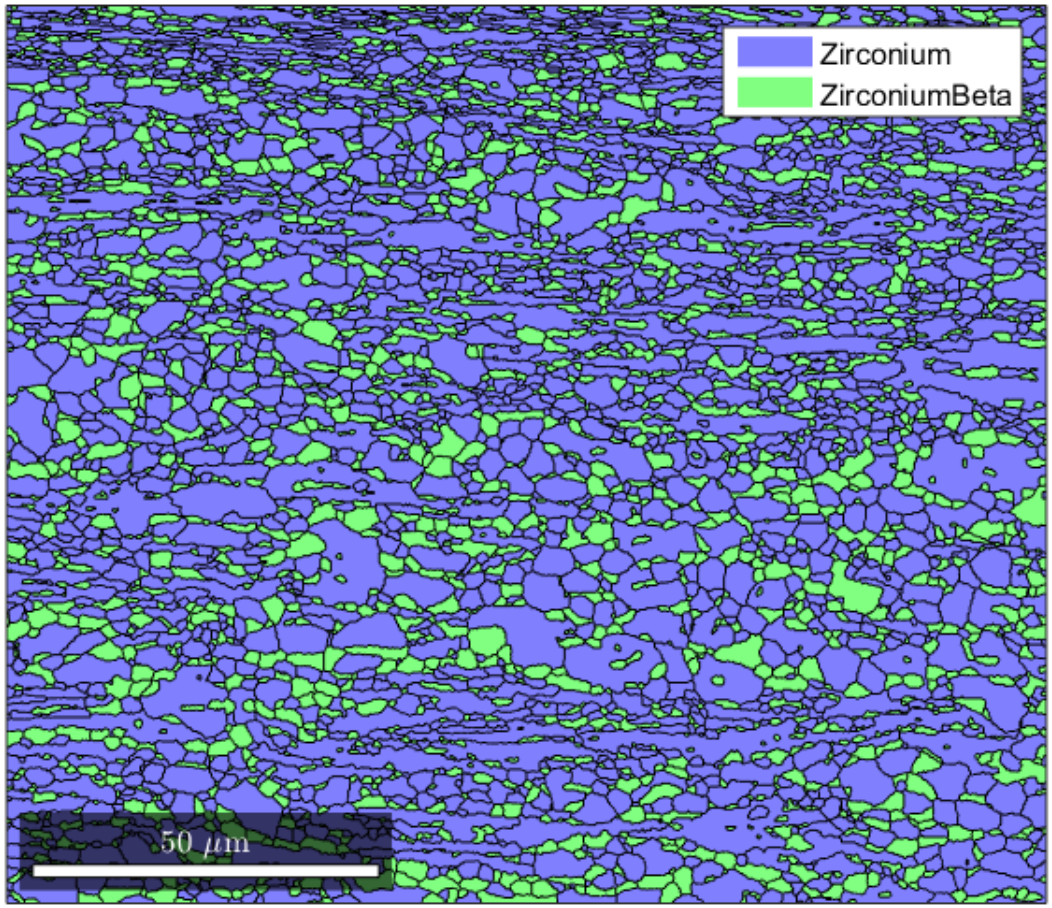
Instead of sending all of *ebsd* as an argument to calcGrains, we send only the indexed parts.


Zirconium
ZirconiumBeta
50 μm

Often you will see that there are a lot of reconstructed grains that are very tiny, perhaps even a single pixel. These are usually mis-indexed points

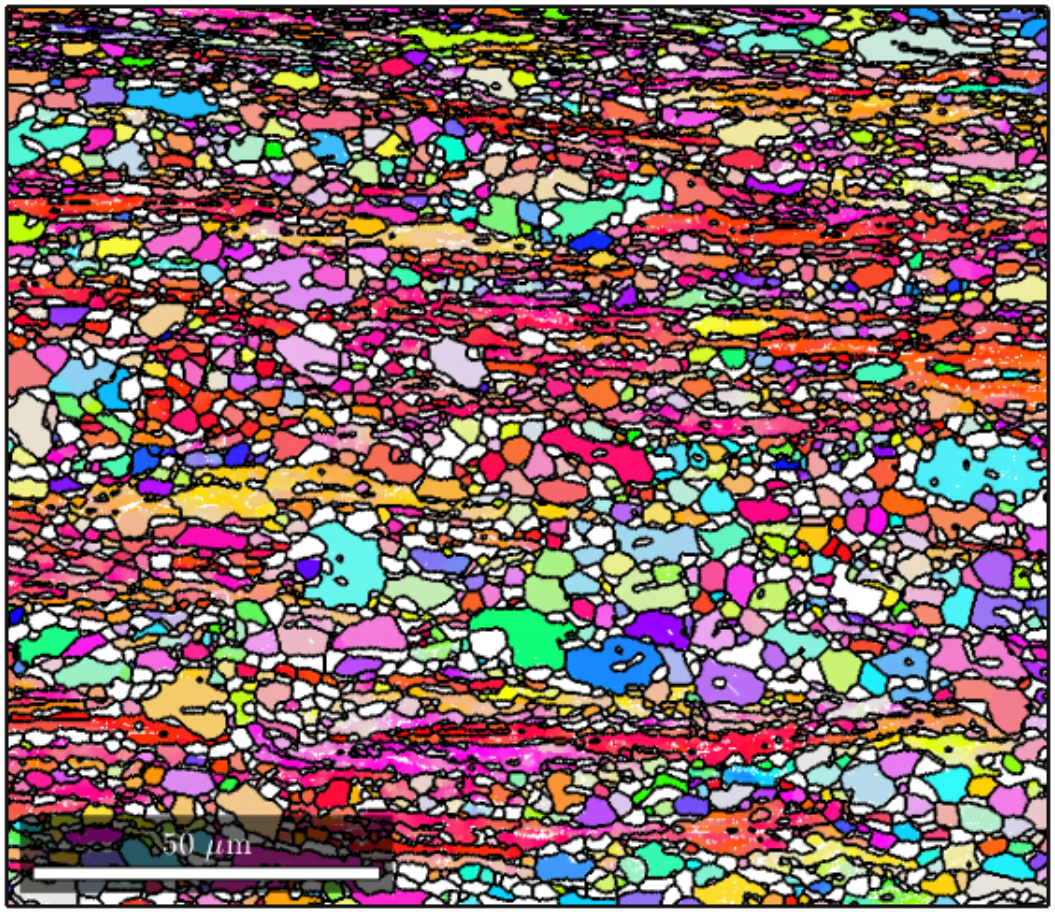
You can get rid of them and clean up your map a little by first running grain reconstruction, then executing the following command to remove any points that are in grains less than 3 pixels large:

```
[grains,ebsd.grainId,ebsd.mis2mean] = ...
    calcGrains(ebsd,'angle',5*degree);

ebsd(grains(grains.grainSize<3))=[];

[grains,ebsd.grainId,ebsd.mis2mean] = ...
    calcGrains(ebsd('indexed'),'angle',5*degree);
```


Zirconium
ZirconiumBeta
50 μm

It's often useful to combine the raw ebsd data with the reconstructed grain boundary data. Here we plot the orientation data and use MATLAB's **hold** feature to add grain boundaries to the figure:
Using the **hold** command, we plot a property of grains called boundary. This will outline the grain boundaries in black. We'll also set the linewidth to 1.5 since the default lines are a bit too thin

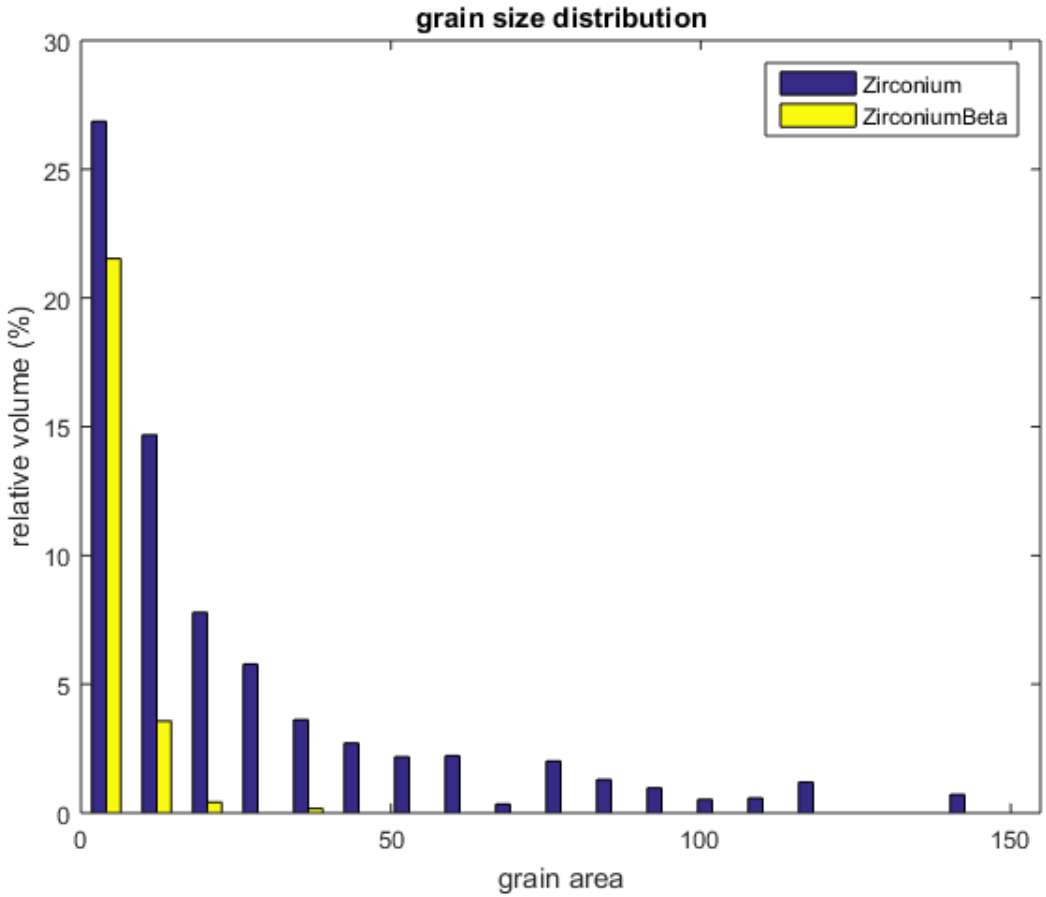
```
figure(1)
plot(ebsd('Zirconium'), ...
    ebsd('Zirconium').orientations);

hold on
plot(grains.boundary,'linewidth',1.5);
hold off
```


50 μm

Grain Statistics

In MTEX, you can get a histogram of grain sizes by sending your grains object to the function **hist()**. The number of bins can also be specified (default is 15).

```
hist(grains,20)
```

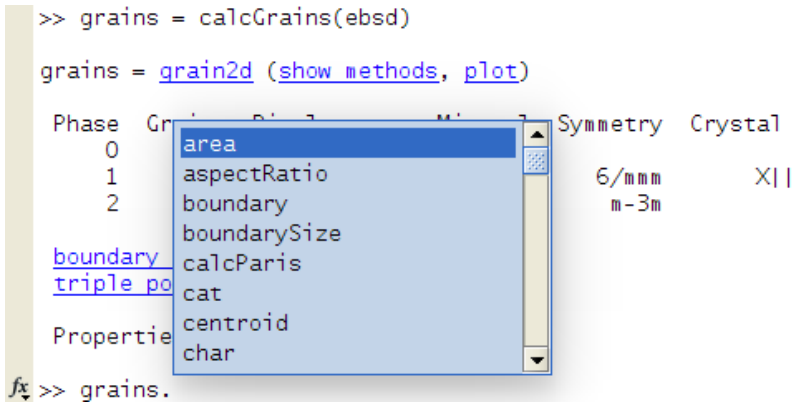

grain size distribution
Zirconium
ZirconiumBeta
relative volume (%)
grain area
0
5
10
15
20
25
30
0
50
100
150

You can also use standard MATLAB functions to get useful information about your grains Here we calculate the median grain size, as well as for each phase individually. (Note: grains.area returns the area in μm^2, and grains.grainSize returns the area in number of pixels.)

```
> median(grains.area)
ans =
    1.7709

> median(grains(grains.phase==1).area)
ans =
    2.5642

> median(grains(grains.phase==2).area)
ans =
    1.2681
```

Properties List Reminder

The grains object has many useful properties! Remember, you can see a list of the properties in grains by typing "grains." (without quotes) into the Matlab command line and pressing the tab key.



Plotting Grain Boundaries

Often the character of grain boundaries can tell us about how a material's grain structure formed. Let's say we want to plot grain boundaries with a certain angle, to compare high angle and low angle boundaries.
This gets a little tricky because it takes advantage of the fact that you can have multiple "layers" of properties in a single structure

First we'll define gB as the grain boundary information from grains

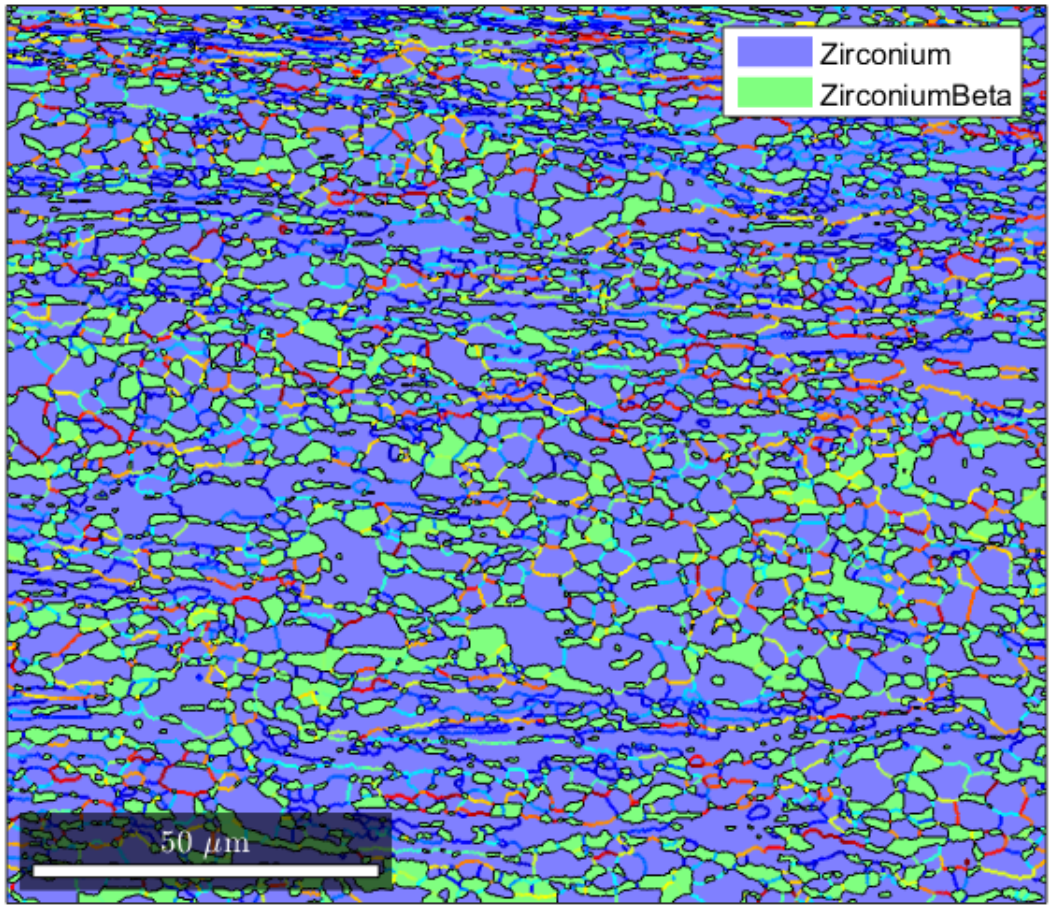
```
gB=grains.boundary;
```

Next we define gBAA as the alpha/alpha grain boundaries. gBBB will be the beta/beta grain boundaries.

```
gBAA=gB('Zirconium','Zirconium');
gBBB=gB('ZirconiumBeta','ZirconiumBeta');
plot(grains)
```

Using the hold command we plot GBAA and GBBB and color them by their misorientation angle. This will let us show which grain boundaries are high and which are low. We needed to do this for just the alpha/alpha and beta/beta boundaries because MTEX doesn't know how to meaningfully find misorientations between diffferent crystal structures

```
hold on
plot(gBAA,gBAA.misorientation.angle/degree, ...
  'linewidth',1.5)
plot(gBBB,gBBB.misorientation.angle/degree, ...
  'linewidth',1.5)
hold off
colormap('jet')
```


Zirconium
ZirconiumBeta
50 μm

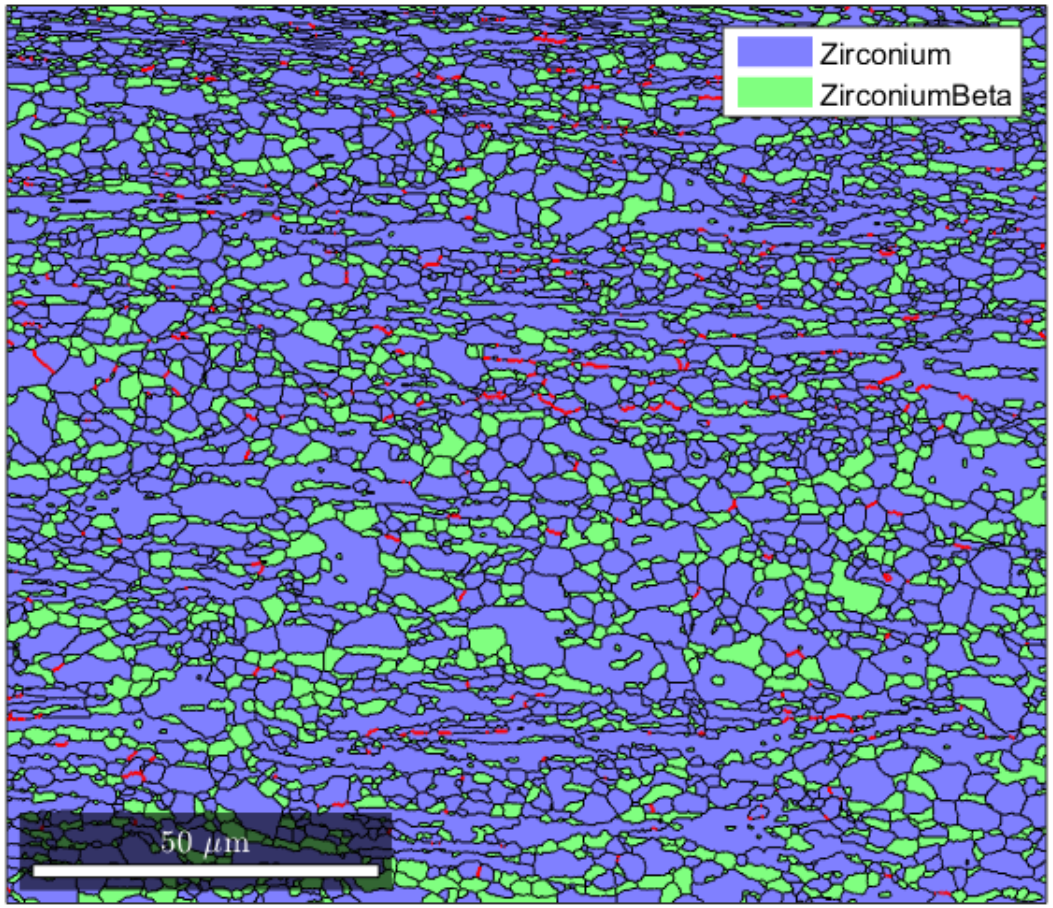
Plotting by Misorientation Axis

In addition to the misorientation angle of grain boundaries, it is often useful to know which crystallographic direction the misorientation occurs about.
We will define the grain boundaries as in the previous section. Now we define an axis that we are interested in. We define axisA as the $\{10\bar{1}0\}$ plane normal. The last argument is the crystal symmetry for the phase in question, which must be provided so MTEX knows what kind of structure your Miller variable goes with.

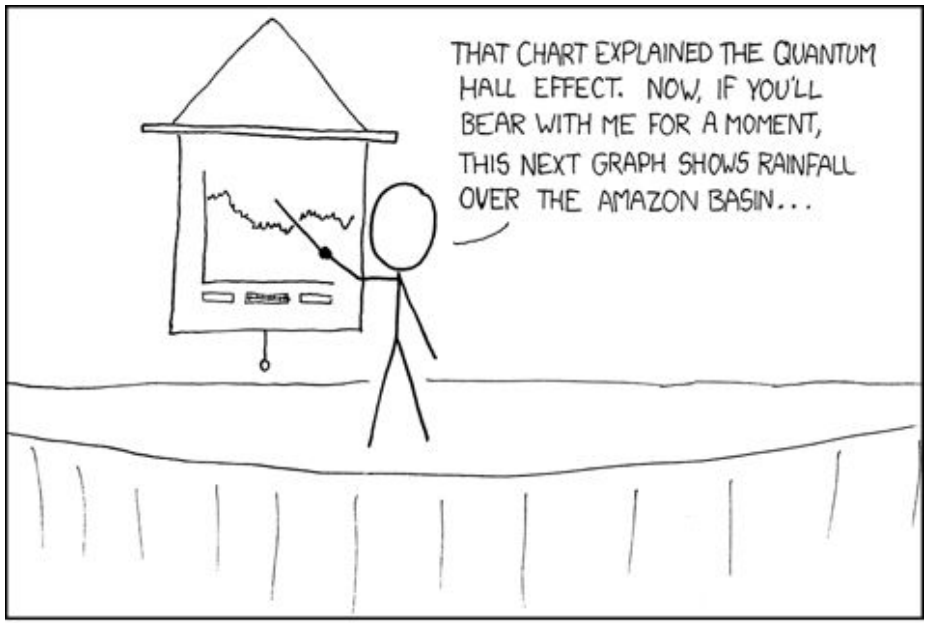
```
gB=grains.boundary;
gBAA=gB('Zirconium','Zirconium');
axisA=Miller(1,0,-1,0,ebsd('Zirconium').CS);
```

Now, we find which boundaries have a misorientation axis close to the ones we've defined. We define axisGBA as a conditional statement, so we can use it for conditional indexing. This will find all grain boundaries that have a misorientation axis within 5 degrees of the one we defined. Here we plot the grains, then use the hold command to add the special grain boundaries we're interested in. The alpha/alpha grains with their 10-10 misorientation axis

```
axisGBA = ...
  angle(gBAA.misorientation.axis,axisA)<5*degree;
plot(grains), hold on
plot(gBAA(axisGBA), 'linecolor','red', ...
  'linewidth',1.5);
hold off
```

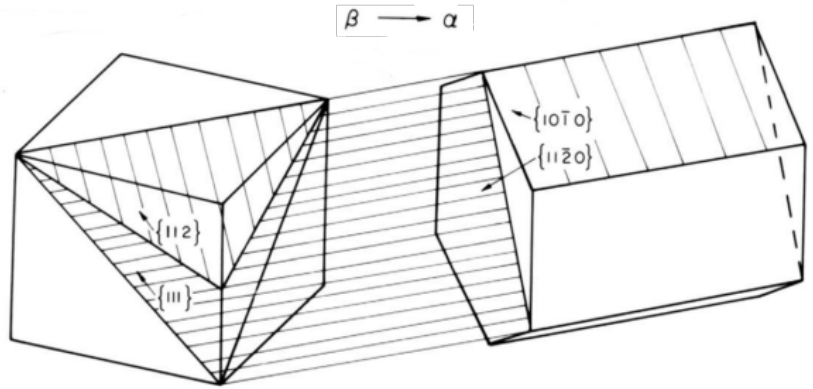

Zirconium
ZirconiumBeta
50 μm

Intermission?



Orientation Relationships

This section will demonstrate how to pick out very specific grain boundaries where both the misorientation axis and angle are defined. This will be used to look at the Burgers relationship between the alpha and beta phase, and to identify twin/parent combinations.



We start by defining 4 Miller directions, such that **a1**||**b1** and **a2**||**b2**. This will let us define the orientation relationship between alpha and beta phases

```
a1=Miller(0,0,0,2,ebsd('Zirconium').CS);
b1=Miller(1,1,0,ebsd('ZirconiumBeta').CS);
a2=Miller(1,1,-2,0,ebsd('Zirconium').CS);
b2=Miller(1,1,1,ebsd('ZirconiumBeta').CS);
```

Next, we create an **orientation map** from alpha to beta based on the sets of parallel directions we just defined.

```
ab=orientation('map',a1,b1,a2,b2, ...
  ebsd('Zirconium').CS,ebsd('ZirconiumBeta').CS);
```

We we now want to get just the grain boundaries that are alpha/beta boundaries.

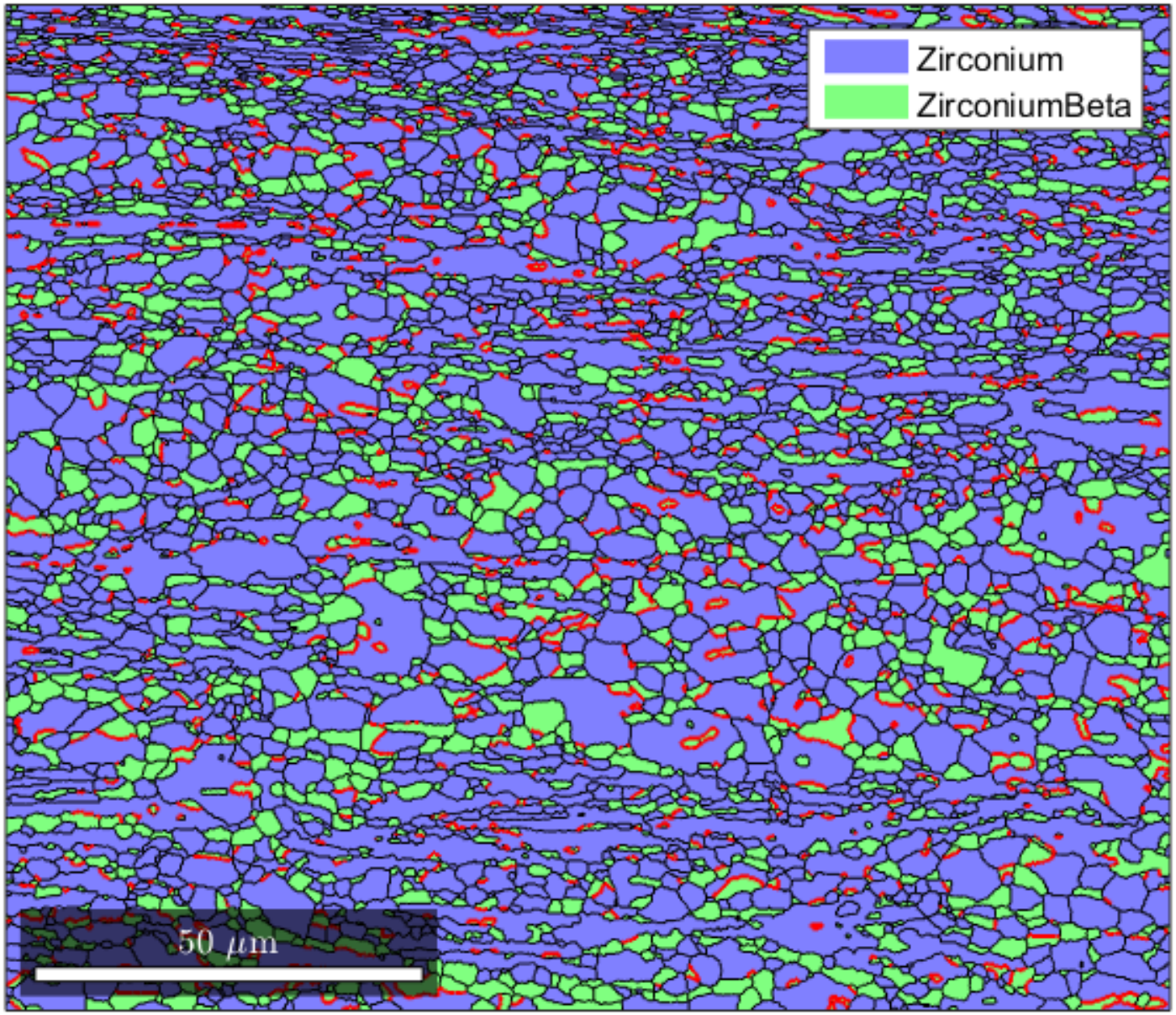
```
gBAB=grains.boundary('Zirconium','ZirconiumBeta');
```

Now we will take all alpha/beta boundaries that have a misorientation within 5 degrees of the relationship we defined.

```
isBurgers=angle(gBAB.misorientation,ab)<5*degree;
```

Finally, we plot the results using conditional indexing:

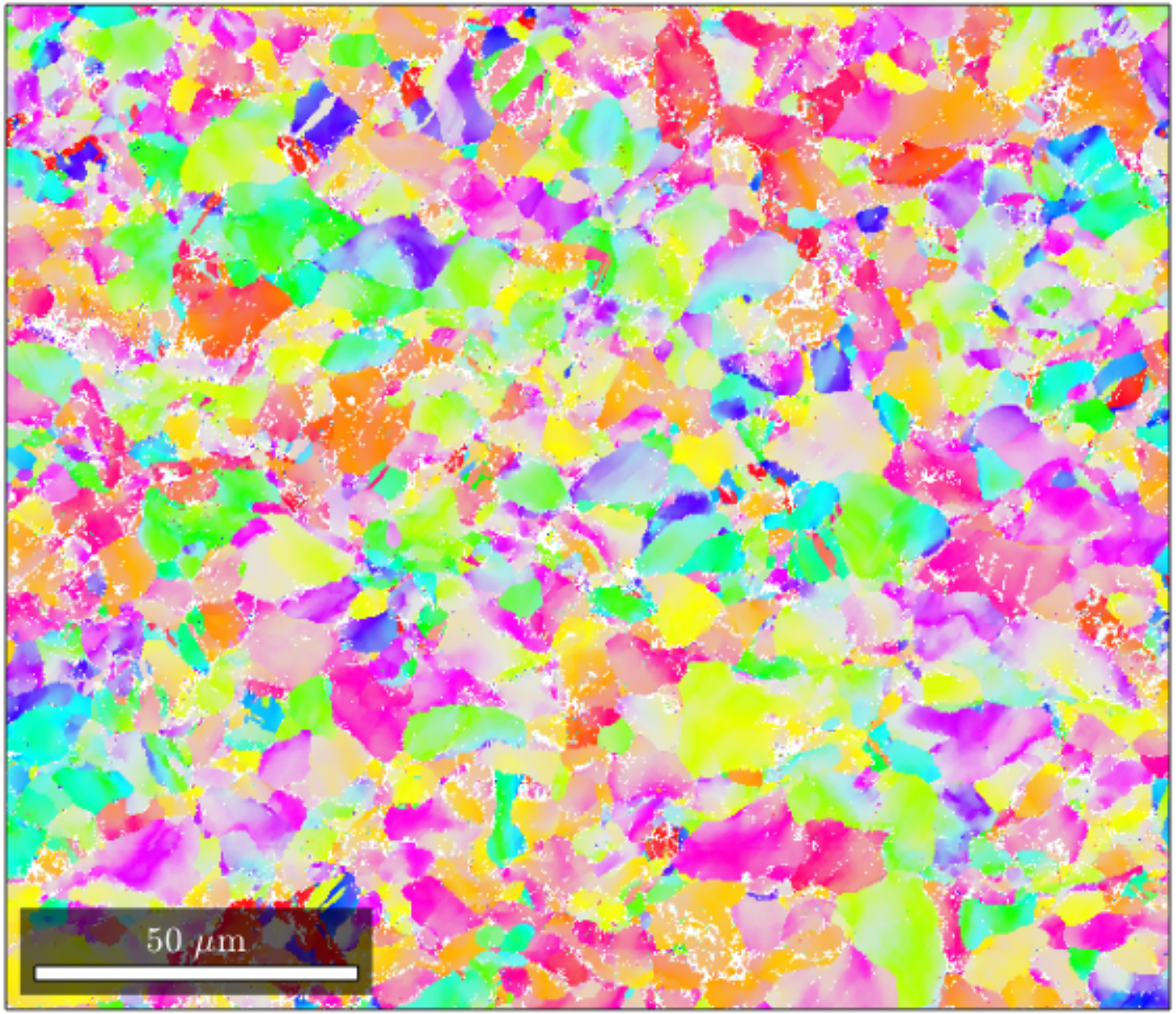
```
plot(grains), hold on
plot(gBAB(isBurgers),'linecolor','red', ...
  'linewidth',1.5);
hold off
```


Zirconium
ZirconiumBeta
50 μm

Twin Identification

In Zr and other materials, we sometimes see a lot of twins formed during deformation. It's really easy to pick out twins from EBSD data with MTEX. The MTEX help page on this is actually pretty good: http://mtex-toolbox.github.io/files/doc/TwinningAnalysis.html
First we'll import a dataset from a tensile sample that has been heavily deformed

```
import_increment3;
figure(1)
plot(ebsd('Zirconium'),ebsd('Zirconium').orientations);
```


50 μm

Now, run grain reconstruction. This map has some poorly indexed points, so we'll get rid of any grains smaller than 3 pixels. We are interested in Zr/Zr grain boundaries only.

```
[grains,ebsd.grainId,ebsd.mis2mean] = ...
  calcGrains(ebsd('indexed'),'angle',5*degree);

ebsd(grains(grains.grainSize<3))=[];
[grains,ebsd.grainId,ebsd.mis2mean] = ...
  calcGrains(ebsd('indexed'),'angle',5*degree);

bounds=grains.boundary;
boundsZr=bounds('Zirconium','Zirconium');
```

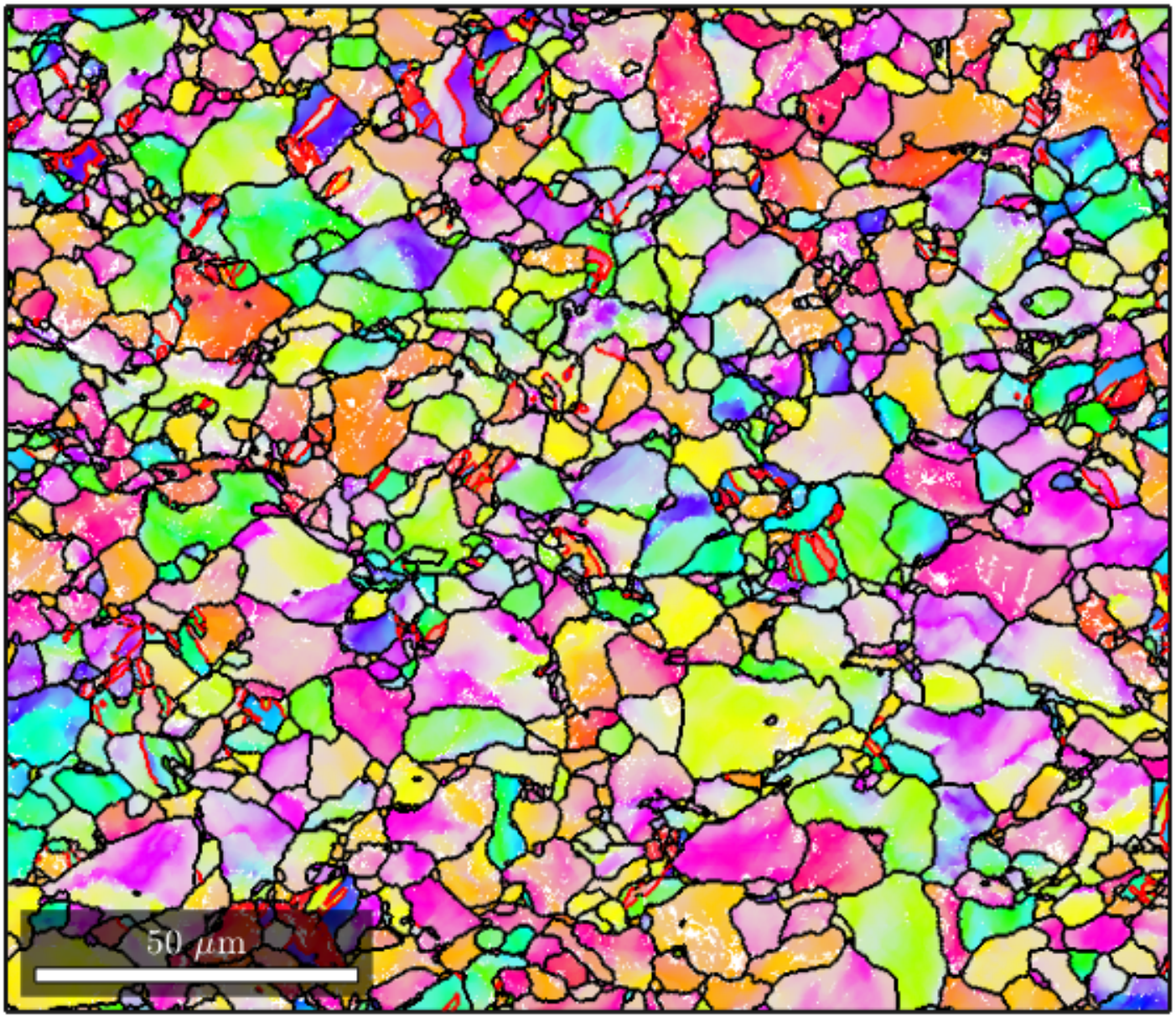
Now we define the twinning orientation relationship. In Zr the relationship is as defined below, where **t1**$\|$**t2** and **t3**$\|$**t4**.

```
t1=Miller(1,1,-2,0,ebsd('Zirconium').CS);
t2=Miller(2,-1,-1,0,ebsd('Zirconium').CS);
t3=Miller(-1,0,1,1,ebsd('Zirconium').CS);
t4=Miller(1,0,-1,1,ebsd('Zirconium').CS);

twinning = orientation('map',t1,t2,t3,t4)
```

This orientation relationship defines the $\{10\bar{1}2\}\langle\bar{1}011\rangle$ tensile twinning mode.

Now we check which grain boundaries are twins by setting a condition to check if the boundary's mistorientation is within 5 degrees of the orientation relationship, then use conditional indexing.

```
istwinning = ...
   angle(twinning,boundsZr.misorientation)<5*degree;
twinboundaries=boundsZr(istwinning);
figure(2)
plot(ebsd('Zirconium'), ...
   ebsd('Zirconium').orientations)
hold on
plot(grains.boundary,'linewidth',1.5);
plot(twinboundaries,'linewidth',1.5, ...
   'linecolor','red');
hold off
```

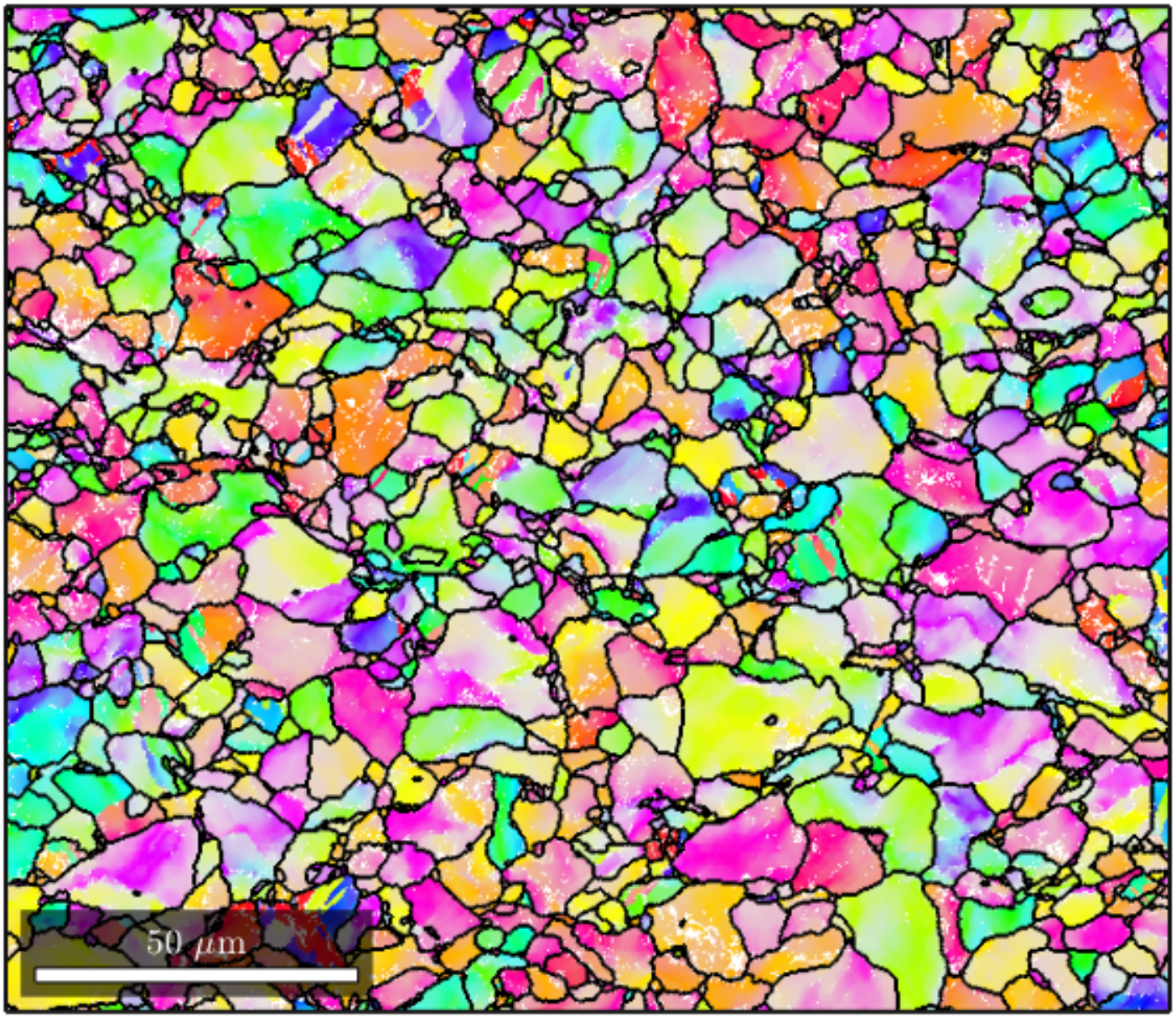

50 μm

There's a command to merge twins with their parents, so that they will be considered a single grain.

```
[mergedGrains, parentId] = ...
  merge(grains, twinboundaries);
```

This also lets you see what your untwinned grain size looks like:

```
> median(grains.area)
ans =
    2.3553

> median(mergedGrains.area)
ans =
    2.5126
```


50 μm

Finally, you can get the twin area fraction like this. First we get the grain id numbers of all the twins. The unique() function makes sure there's no duplicates.

```
twinId = unique(boundsZr(istwinning).grainId);
```

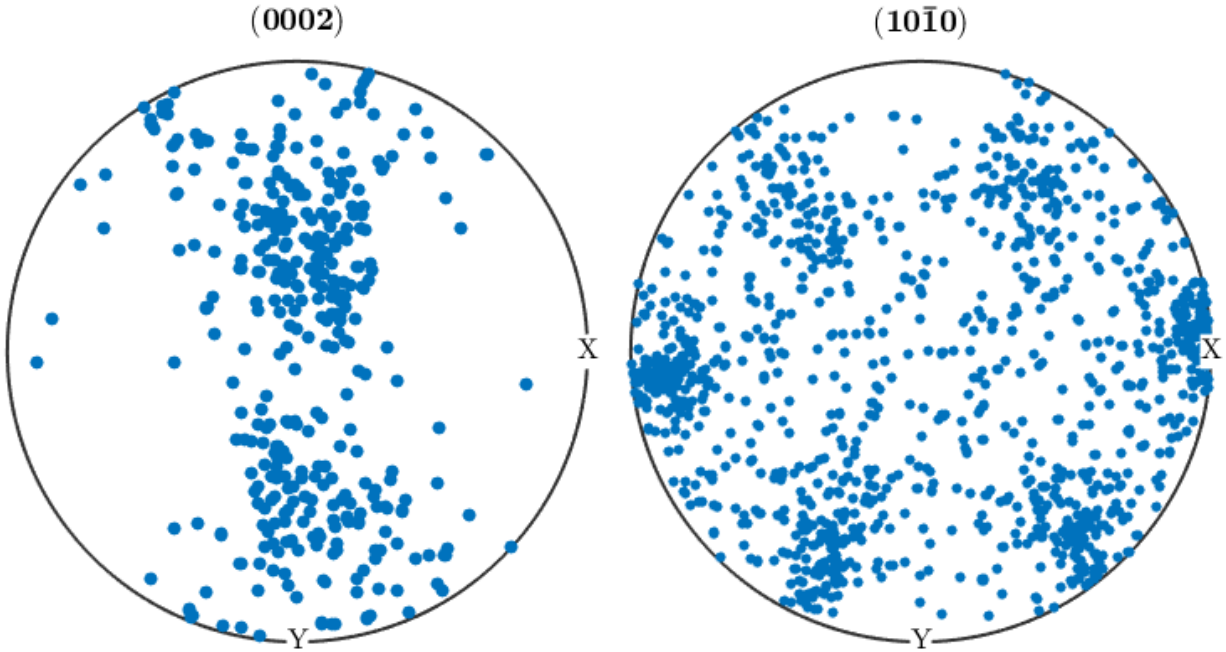
Next we get the sum of the areas all the twins, and divide them from the sum of the areas of the entire set of grains

```
> twinFraction = sum(area(grains(twinId))) / ...
  sum(area(grains)) * 100
twinFraction =
    14.7671
```

Pole Figures

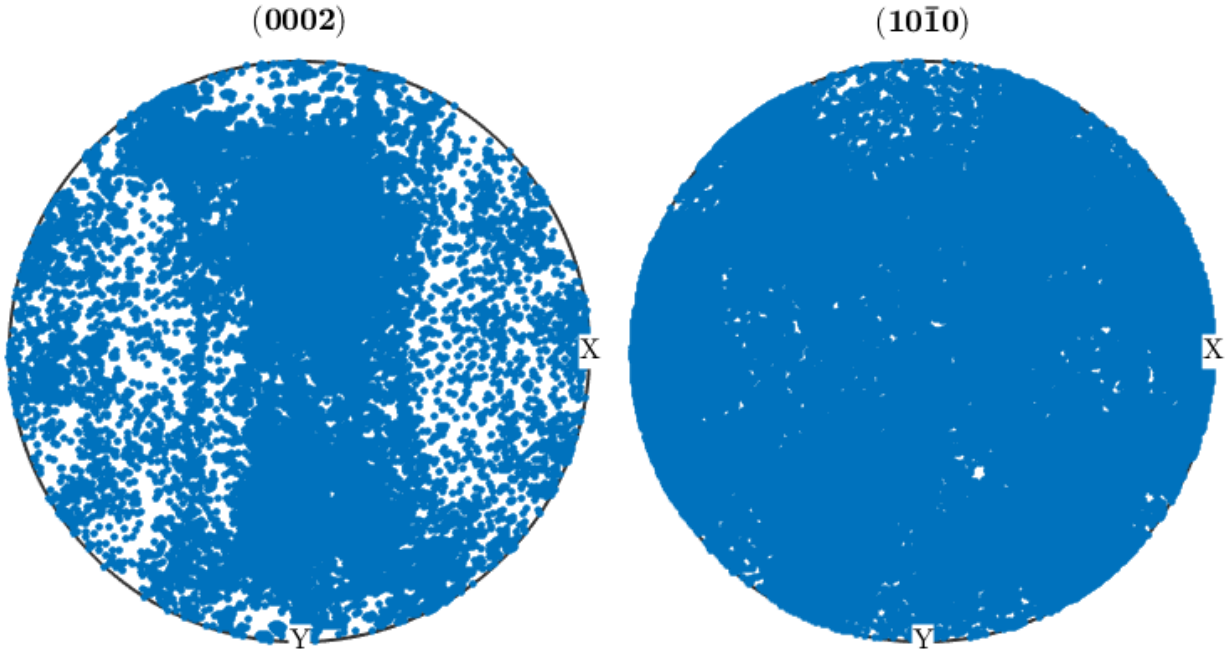
We can take the orientation data from an EBSD structure and use it to generate pole figures.
First, define the crystallographic directions you'd like to create pole figures for, then use the **plotPDF** command:

```
p1=Miller(0,0,0,2,ebsd('Zirconium').CS);
p2=Miller(1,0,-1,0,ebsd('Zirconium').CS);
plotPDF(ebsd('Zirconium').orientations,[p1 p2])
```


(0002)
X
Y
(10$\bar{1}$0)
X
Y

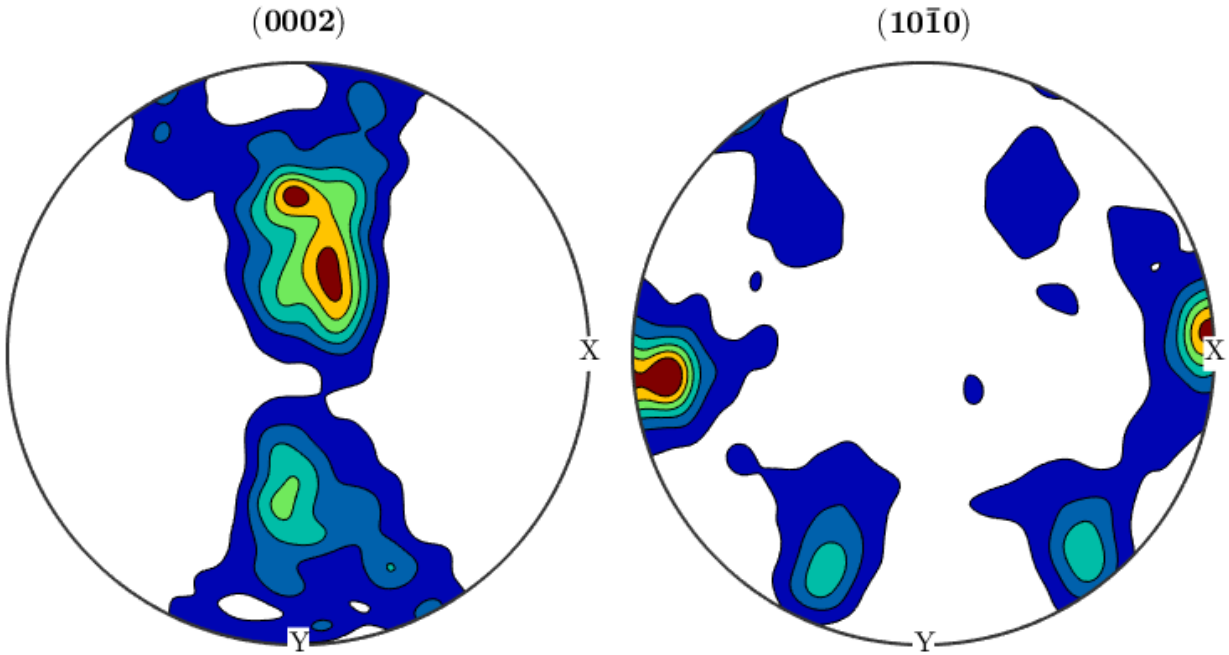
The plot we just made will randomly plot a certain number of points from the ebsd map onto the pole figure. If you want to plot all of the points, you can do so by setting the 'points' option to 'all'. It's also useful to change the marker size.

```
plotPDF(ebsd('Zirconium').orientations,[p1 p2], ...
  'points','all','markerSize',3)
```


(0002)
X
Y
(10$\bar{1}$0)
X
Y

All these points makes it difficult to see what's going on, and it's often more useful to display PF's as contour plots. The *contourf* option will plot the PF as a filled contour plot, and the option after it sets the scale of the contours, in this case from 0 to 6 mrd in steps of 1. If you prefer unfilled plots you can use contour instead of contourf.

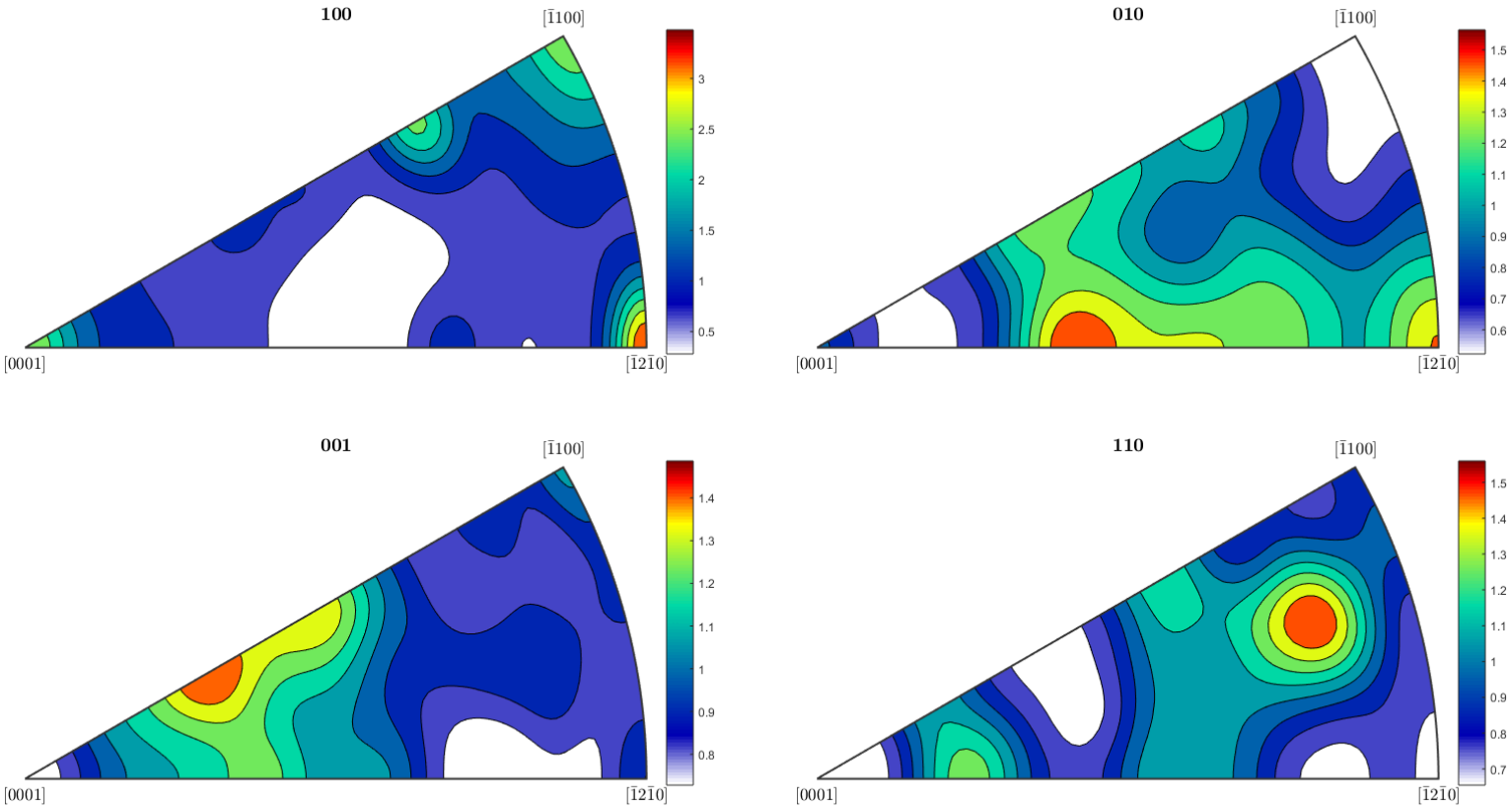
```
plotPDF(ebsd('Zirconium').orientations,[p1 p2], ...
  'contourf',0:1:6)
colorbar;
```


(0002)
X
Y
(10$\bar{1}$0)
X
Y

Inverse Pole Figures

If you want to plot an inverse pole figure to see which crystal orientations are aligned along a given sample direction, MTEX can do it quite easily. The following code will plot inverse pole figures for the x,y, and z directions of the ebsd map. Here we also show that you can use mean orientation data from grains instead of considering each individual pixel.

```
[grains,ebsd.grainId,ebsd.mis2mean] = ...
  calcGrains(ebsd('indexed'),'angle',5*degree);
figure(1)
plotIPDF(grains(grains.phase==1).meanOrientation, ...
  xvector,'contourf'), colorbar;
figure(2)
plotIPDF(grains(grains.phase==1).meanOrientation, ...
  yvector,'contourf'), colorbar;
figure(3)
plotIPDF(grains(grains.phase==1).meanOrientation, ...
  zvector,'contourf'), colorbar;
```


100
[$\bar{1}$100]
[0001]
[$\bar{1}2\bar{1}0$]
3
2.5
2
1.5
1
0.5
010
[$\bar{1}$100]
[0001]
[$\bar{1}2\bar{1}0$]
1.5
1.4
1.3
1.2
1.1
1
0.9
0.8
0.7
0.6
001
[$\bar{1}$100]
[0001]
[$\bar{1}2\bar{1}0$]
1.4
1.3
1.2
1.1
1
0.9
0.8
110
[$\bar{1}$100]
[0001]
[$\bar{1}2\bar{1}0$]
1.5
1.4
1.3
1.2
1.1
1
0.9
0.8
0.7

If you want to plot an IPF for an arbitrary direction you can do that as well. The following code will create an IPF for a direction halfway between the x and y directions (i.e. going diagonally across the map).

```
dir=vector3d(1,1,0);
figure(4)
plotIPDF(grains(grains.phase==1).meanOrientation, ...
    dir,'contourf'), colorbar;
```

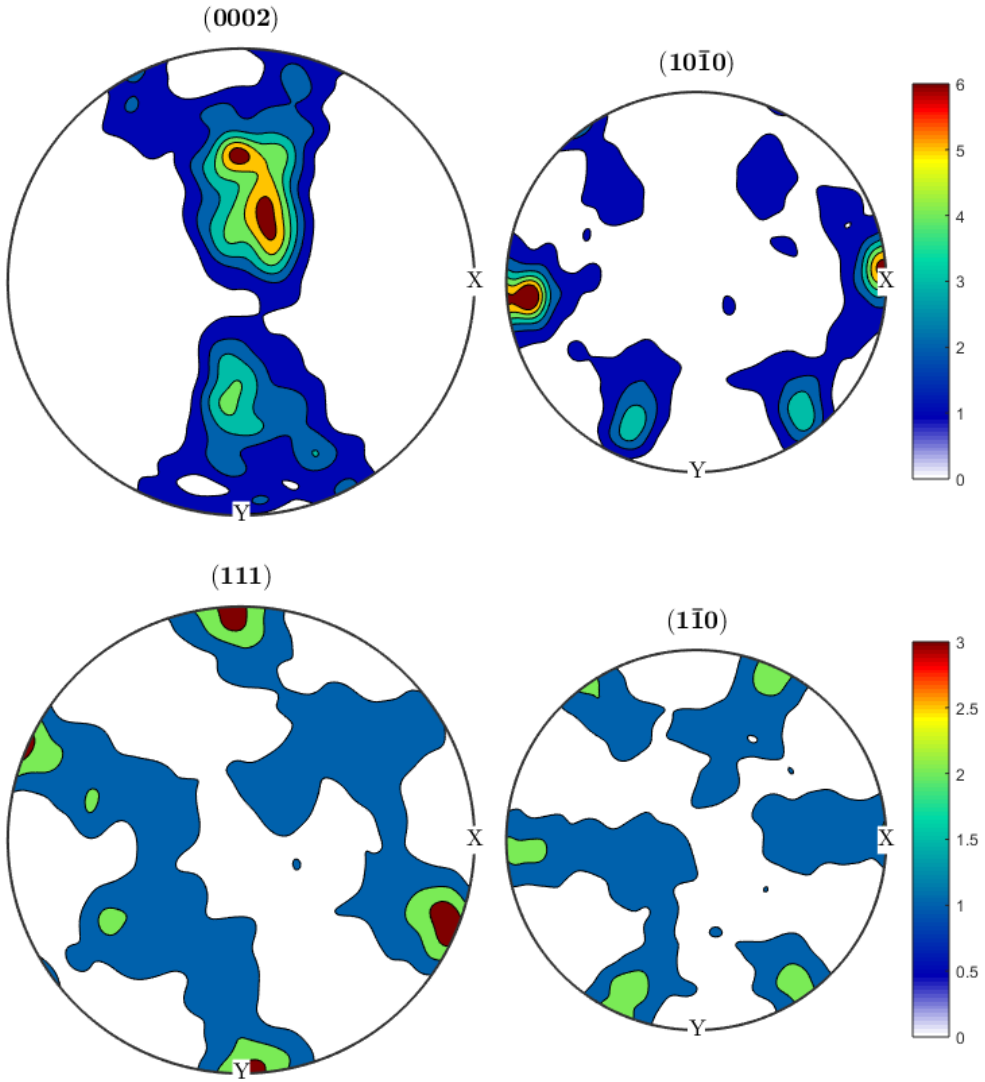
All of the figures here were done for the alpha phase, which is phase 1 in grains. To get figures for the beta phase, just change the (grains.phase==1) to (grains.phase==2).

An ODF contains full information about the distribution of orientations, instead of just information about a single crystal direction as in a pole figure. It can be calculated from other data-types, such as EBSD data, using the **calcODF** function.

```
alphaODF=calcODF(ebsd('Zirconium').orientations)
betaODF=calcODF(ebsd('ZirconiumBeta').orientations)
```

Let's take another look at the pole figures for our data, this time both alpha and beta phasess:

```
p1=Miller(0,0,0,2,ebsd('Zirconium').CS);
p2=Miller(1,0,-1,0,ebsd('Zirconium').CS);
p1beta=Miller(1,1,1,ebsd('ZirconiumBeta').CS);
p2beta=Miller(1,-1,0,ebsd('ZirconiumBeta').CS);

figure(1)
plotPDF(ebsd('Zirconium').orientations, ...
  [p1 p2],'contourf',0:1:6), colorbar;
figure(2)
plotPDF(ebsd('ZirconiumBeta').orientations, ...
  [p1beta p2beta],'contourf',0:1:3)
colorbar;
```


(0002)
(10$\bar{1}$0)
(111)
(1$\bar{1}$0)
X
Y
6
5
4
3
2
1
0
2.5
1.5
0.5

It seems like the pole figure is slightly tilted! This is pretty common since EBSD samples have to be aligned by hand in the SEM and might be off by a few degrees. We can use an MTEX function on the ODF to correct this. The **centerSpecimen** function takes in an ODF and attempts to find and align the sample symmetry. It returns a rotated ODF along with information about the rotation.

```
[cor_alphaODF, rotAlpha]=centerSpecimen(alphaODF);
[cor_betaODF, rotBeta]=centerSpecimen(betaODF);
```

We now have corrected ODF's for the alpha and beta phases, as well as the rotations applied to them. Since both are in the same sample, both rotations should be the same, so let's check:

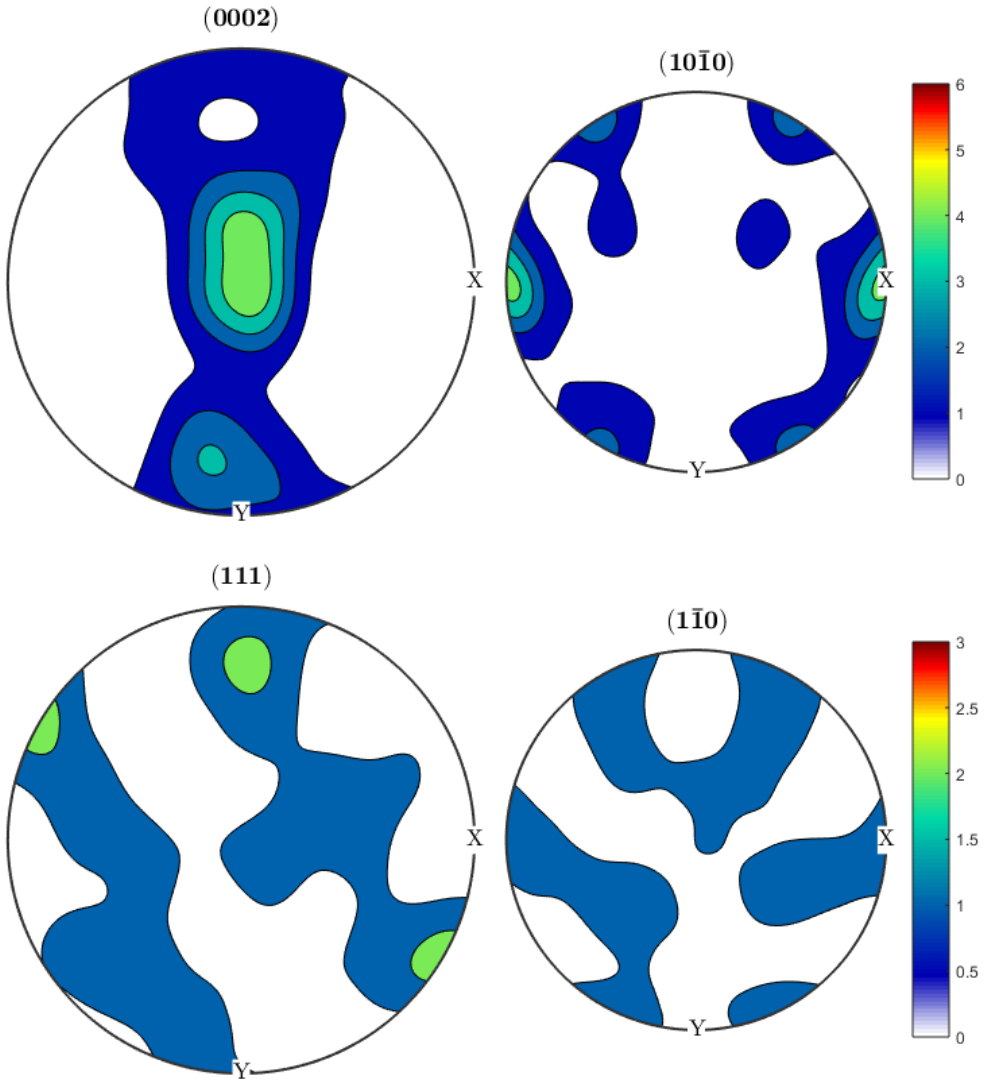
```
 > rotAlpha
rotAlpha = rotation (show methods, plot)
  size: 1 x 1

  Bunge Euler angles in degree
     phi1      Phi     phi2     Inv.
  195.759  24.3131  170.364        0
 > rotBeta
rotBeta = rotation (show methods, plot)
  size: 1 x 1

  Bunge Euler angles in degree
     phi1      Phi     phi2     Inv.
  273.467   16.099  90.3354        0
```

We can see from the euler angle output of the rotations that they don't look like they agree very well. Let's check how much they've been rotated by, and the axes about which they're being rotated.

```
> rotAlpha.angle/degree
ans =
   25.0610

> rotAlpha.axis
ans = vector3d (show methods, plot)
 size: 1 x 1
          x          y          z
  -0.946878  -0.213336   0.240645

> rotBeta.angle/degree
ans =
   16.5391
> rotBeta.axis
 ans = vector3d (show methods, plot)
 size: 1 x 1
          x          y          z
  0.0266052  -0.973203   0.228405
```

We can see that the alpha phase has been rotated by 25 degrees about an axis close to the x axis, while the beta phase is being rotated by 16 degrees, mostly about the y axis.
It doesn't make physical sense to apply two different corrections to the different phases, so we can use the rotate() function to rotate one of the ODF's by the correction applied to the other. We can assume that since the sample is mostly alpha, the statistics for that phase are better, so let's apply the alpha rotation to the beta phase and then plot the results.

```
figure(3)
plotPDF(cor_alphaODF,[p1 p2],'contourf',0:6)
colorbar;

figure(4)
cor_betaODF=rotate(betaODF,rotAlpha);
plotPDF(cor_betaODF,[p1beta p2beta],'contourf',0:3)
colorbar;
```


(0002)
X
Y
(10$\bar{1}$0)
X
Y
6
5
4
3
2
1
0
(111)
X
Y
(1$\bar{1}$0)
X
Y
3
2.5
2
1.5
1
0.5
0

Schmidt Factor Analysis

Say we want to calculate the Schmidt factor for various slip modes in the HCP structure. We need to know the direction of loading, and the orientation of our crystallite.
First, we define our loadVector, which is direction of loading. You can use "xvector", "yvector" and "zvector" to refer to the principle orientations of the plane. For EBSD, it will typically be xvector or yvector.

```
loadVector = zvector;
slipPlaneType=symmetrise(Miller(1,0,-1,0,CS,'hkl'));
slipDirecType=symmetrise(Miller(1,1,-2,0,CS,'uvw'));
```

Here we use **symmetrise** to create vectors with all the symmetrically equivalent planes and directions for prism slip.

Recall that the Schmidt factor is

$$m = \cos(\phi)\cos(\lambda)$$

Where ϕ is the angle between the loading axis and the slip direction, and λ is the angle between the loading axis and the slip plane.
The geometric definition of the dot product is:

$$a \cdot b = |a||b|\cos(\theta)$$

So if we simply normalize all of our vectors, the calculation of Schmidt factors simplifies to the calculation of dot products, which MATLAB can do very quickly.

Next, calculate the dot product of each slip plane and slip direction with the load vector. Here we have arbitrarily chosen to look at grain 6 in our ebsd data set.

```
o = grains(6).meanOrientation;
a = slipPlaneType;
N = length(slipDirecType);
slipPlane = a( ceil( (1:N*length(a))/N ) );
slipDirection = repmat(slipDirecType, ...
    length(slipPlaneType),1);
prismtau=dot(normalize(o*slipPlane),loadVector).* ...
   dot(normalize(o*slipDirection),loadVector);
```

The o*slipPlane returns the vector of the slipPlane normal in the sample reference frame, so that we can compare directly to the loadVector, which is also in this reference frame.

It's also important to remove slip systems that are impossible. The slip direction must be in the slip plane, i.e. orthogonal to the slip plane normal. Here is some code that uses some fancy conditional indexing to achieve this.

```
prismtau(abs(dot(vector3d(slipDirection), ...
    vector3d(slipPlane))) > 0) = [];
```

The code can then be adjusted to calculate values for basal or pyramidal slip modes. For example:

```
% Basal Slip
% (0001), [11-20]
slipPlaneType=symmetrise(Miller(0,0,0,1,CS,'hkl'));
slipDirecType=symmetrise(Miller(1,1,-2,0,CS,'uvw'));

% Pyramidal Slip
% (10-11), [11-23]
slipPlaneType=symmetrise(Miller(1,0,-1,1,CS,'hkl'));
slipDirecType=symmetrise(Miller(1,1,-2,3,CS,'uvw'));
```

Summary

MATLAB and the MTEX toolbox are very useful for analyzing and visualizing data related to crystallographic orientations and texture.

If you have a project involving crystallographic orientations, chances are that MTEX can make it far easier. Feel free to talk to us and we can help you out.

MATLAB and the MTEX Toolbox for Texture Analysis

Nuclear Materials Research Group

Travis Skippon and Chris Cochrane

Department of Mechanical and Materials Engineering
Queen's University