Misorientations

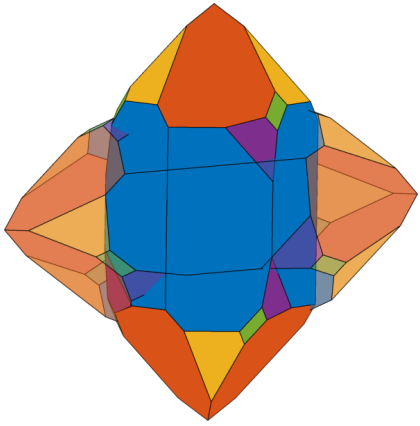
Ralf Hielscher

Faculty of Mathematics,
Chemnitz University of Technology, Germany

MTEX Workshop 2020

Misorientations

- describe the relative alignment of two crystal lattices with respect to each other
- if the lattices are of same phase it is the rotation that rotates one lattice onto the other.
- if the lattices of different phase it is the rotation between the corresponding orthonormal reference frames
- is the coordinate transform from lattice A coordinates into lattice B coordinates
- is subject to the symmetries of lattice A as well as lattice B

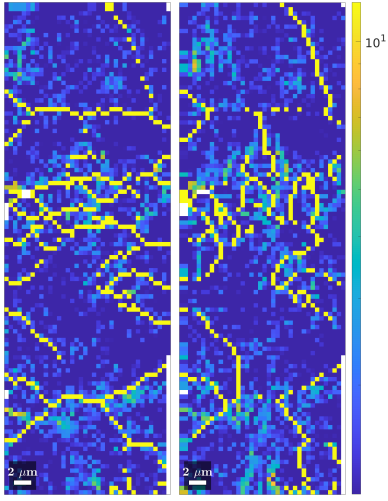


```
plot(cS)
hold on
plot(mori * cS)
```

Examples

neighboring pixels

```
ebsd.gradientX, ebsd.KAM
```




```
ori = ebsd.orientation;
angle(ori(:,1:end-1),ori(:,2:end))
```

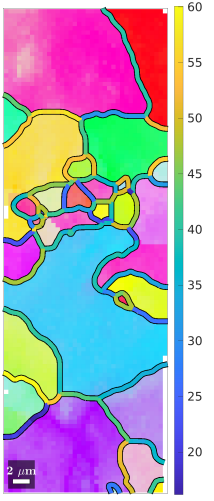
Examples

neighboring pixels

```
ebsd.gradientX, ebsd.KAM
```

grain boundary misorientations

```
grains.boundary.misorientation
```




```
gB    = grains.boundary
mori = gB.misorientation
plot(gB, mori.angle ./ degree)
```

Examples

neighboring pixels

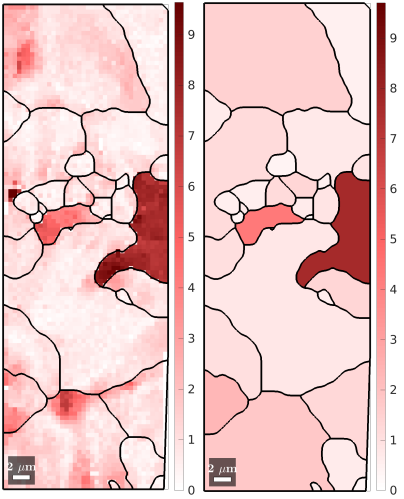
```
ebsd.gradientX, ebsd.KAM
```

grain boundary misorientations

```
grains.boundary.misorientation
```

pixel to grain mean orientation

```
grains.calcGROD, grains.GOS
```




```
plot(ebsd.mis2mean.angle./ degree)
plot(grains.GOS)
```

Examples

neighboring pixels

```
ebsd.gradientX, ebsd.KAM
```

grain boundary misorientations

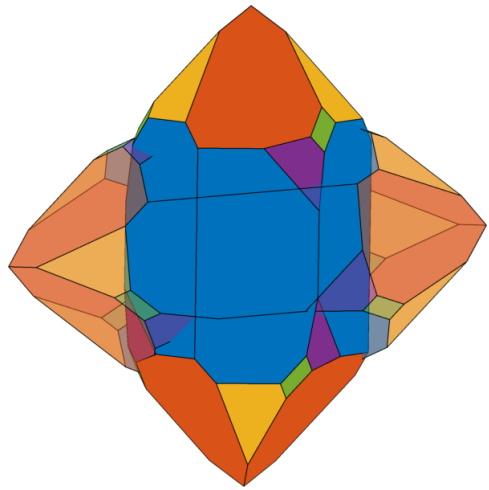
```
grains.boundary.misorientation
```

pixel to grain mean orientation

```
grains.calcGROD, grains.GOS
```

twinning misorientations

```
xi = Miller(1,0,-1,1,CS)
orientation.byAxisAngle(xi, pi)
```



Examples

neighboring pixels

```
ebsd.gradientX, ebsd.KAM
```

grain boundary misorientations

```
grains.boundary.misorientation
```

pixel to grain mean orientation

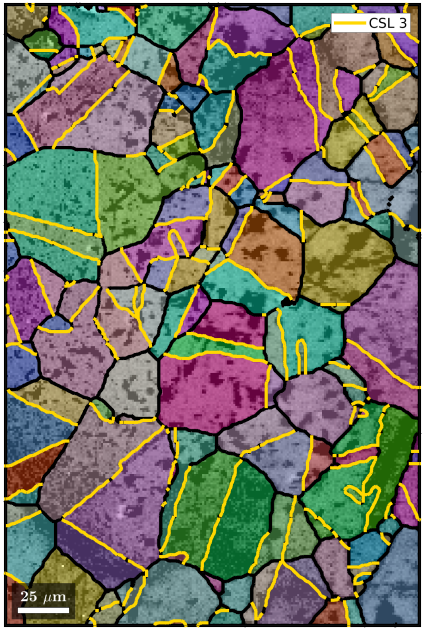
```
grains.calcGROD, grains.GOS
```

twinning misorientations

```
xi = Miller(1,0,-1,1,CS)
orientation.byAxisAngle(xi, pi)
```

common site lattice misorientations

```
mori = CSL(3, cs)
```



Examples

neighboring pixels

```
ebsd.gradientX, ebsd.KAM
```

grain boundary misorientations

```
grains.boundary.misorientation
```

pixel to grain mean orientation

```
grains.calcGROD, grains.GOS
```

twinning misorientations

```
xi = Miller(1,0,-1,1,CS)
orientation.byAxisAngle(xi, pi)
```

common site lattice misorientations

```
mori = CSL(3, cs)
```

parent to child orientation relationship

```
orientation.burger(csB, csA)
```

Coordinate Transforms

Remember, an orientation converts crystal into specimen coordinates

```
oBeta = orientation.byEuler(0,0,0,csBeta)
r = oBeta * Miller(1,1,0,csBeta)
```

```
oBeta = orientation
  crystal symmetry : Ti (BETA) (432)

  Bunge Euler angles in degree
  phi1  Phi phi2 Inv.
     0    0    0    0

r = vector3d
         x         y         z
  0.302024 0.302024         0
```

The inverse of an orientation converts specimen into crystal coordinates

```
oAlpha = orientation.byEuler(135, 90, 55, csAlpha)
inv(oAlpha) * r
```

Hence, **inv**(aAlpha) ∗ oBeta converts beta coordinates into alpha coordinates

```
inv(oAlpha) * oBeta * Miller(1,1,0,csBeta)
```

Coordinate Transforms

Remember, an orientation converts crystal into specimen coordinates

```
oBeta = orientation.byEuler(0,0,0,csBeta)
r = oBeta * Miller(1,1,0,csBeta)
```

The inverse of an orientation converts specimen into crystal coordinates

```
oAlpha = orientation.byEuler(135, 90, 55, csAlpha)
inv(oAlpha) * r
```

```
oAlpha = orientation
  crystal symmetry : Ti (alpha) (622, X||a*, Y||b, Z||c*)

  Bunge Euler angles in degree
  phi1  Phi phi2 Inv.
   135   90   55    0

ans = Miller
 mineral: Ti (alpha) (622, X||a*, Y||b, Z||c*)
  h      k      i      l
  0      0      0 1.9994
```

Hence, **inv**(aAlpha) $*$ oBeta converts beta coordinates into alpha coordinates

Coordinate Transforms

Remember, an orientation converts crystal into specimen coordinates

```
oBeta = orientation.byEuler(0,0,0,csBeta)
r = oBeta * Miller(1,1,0,csBeta)
```

The inverse of an orientation converts specimen into crystal coordinates

```
oAlpha = orientation.byEuler(135, 90, 55, csAlpha)
inv(oAlpha) * r
```

Hence, **inv**(aAlpha) $*$ oBeta converts beta coordinates into alpha coordinates

```
inv(oAlpha) *  oBeta * Miller(1,1,0,csBeta)
```

```
ans = Miller
 mineral: Ti (alpha) (622, X||a*, Y||b, Z||c*)
  h      k      i      l
  0      0      0 1.9994
```

Misorientations

The misorientation **beta2alpha** transforms Beta coordinates into Alpha coordinates

```
beta2alpha = inv(oAlpha) * oBeta
beta2alpha * Miller(1,1,0,csBeta)
```

```
beta2alpha = misorientation
  crystal symmetry : Ti (BETA) (432)
  crystal symmetry : Ti (alpha) (622, X||a*, Y||b, Z||c*)

  Bunge Euler angles in degree
  phi1  Phi phi2 Inv.
   125   90   45    0

ans = Miller
 mineral: Ti (alpha) (622, X||a*, Y||b, Z||c*)
  h      k      i      l
  0      0      0 1.9994
```

its inverse transforms Alpha coordinates into Beta coordinates

```
inv(beta2alpha) * Miller(0,0,0,1,csAlpha)
```

beta2alpha is close to the Burger orientation relationship between the hexagonal alpha and

Misorientations

The misorientation **beta2alpha** transforms Beta coordinates into Alpha coordinates

```
beta2alpha = inv(oAlpha) * oBeta
beta2alpha * Miller(1,1,0,csBeta)
```

its inverse transforms Alpha coordinates into Beta coordinates

```
inv(beta2alpha) * Miller(0,0,0,1,csAlpha)
```

```
a2b = misorientation
  crystal symmetry : Ti (BETA) (432)
  crystal symmetry : Ti (alpha) (622, X||a*, Y||b, Z||c*)

  Bunge Euler angles in degree
  phi1  Phi phi2 Inv.
   125   90   45    0

ans = Miller
 mineral: Ti (BETA) (432)
       h       k       l
  0.5002  0.5002       0
```

beta2alpha is close to the Burger orientation relationship between the hexagonal alpha and

Misorientations

The misorientation **beta2alpha** transforms Beta coordinates into Alpha coordinates

```
beta2alpha = inv(oAlpha) * oBeta
beta2alpha * Miller(1,1,0,csBeta)
```

its inverse transforms Alpha coordinates into Beta coordinates

```
inv(beta2alpha) * Miller(0,0,0,1,csAlpha)
```

`beta2alpha` is close to the Burger orientation relationship between the hexagonal alpha and the cubic beta phase.

```
round2Miller(beta2alpha)
```

```
(110) || (0001)    [1-11] || [-12-10]    error: 0.26
```

define the Burger orientation relationship exactly

```
beta2alpha = orientation.map(...
  Miller(1,1,0,csBeta,'hkl'),Miller(0,0,0,1,csAlpha,'hkl'),...
  Miller(-1,1,-1,csBeta,'uvw'),Miller(2,-1,-1,0,csAlpha,'uvw'))
```

Misorientations

The misorientation **beta2alpha** transforms Beta coordinates into Alpha coordinates

```
beta2alpha = inv(oAlpha) * oBeta
beta2alpha * Miller(1,1,0,csBeta)
```

its inverse transforms Alpha coordinates into Beta coordinates

```
inv(beta2alpha) * Miller(0,0,0,1,csAlpha)
```

`beta2alpha` is close to the Burger orientation relationship between the hexagonal alpha and the cubic beta phase.

```
round2Miller(beta2alpha)
```

```
(110) || (0001)    [1-11] || [-12-10]    error: 0.26
```

define the Burger orientation relationship exactly

```
beta2alpha = orientation.map(...
  Miller(1,1,0,csBeta,'hkl'),Miller(0,0,0,1,csAlpha,'hkl'),...
  Miller(-1,1,-1,csBeta,'uvw'),Miller(2,-1,-1,0,csAlpha,'uvw'))
```

Symmetrically Equivalent Misorientations

- `sA` - symmetry operation from `csAlpha`
- `sB` - symmetry operation from `csBeta`
- `beta2alpha` - misorientation from beta to alpha
- symmetrically equivalent misorientations:
  `sA * beta2alpha * sB`
- number of symmetric misorientations:
  - 2 Laue groups: `|csAlpha| * |csBeta| / 2`
  - otherwise: `|csAlpha| * |csBeta|`

```
beta2alpha.symmetrise
```

```
ans = misorientation
  size: 288 x 1
  crystal symmetry : Ti (BETA) (432)
  crystal symmetry : Ti (alpha) (622, X||a*, Y||b, Z||c*)

  show Euler angles
```

Grain Exchange Symmetry

- consider misorientations `inv(ori1)*ori2` between orientations of the same phase
- sometimes no ordering `ori1`, `ori2` is given e.g. boundary misorientations
- $\rightarrow$ we should not distinguish between `inv(ori1)*ori2` vs `inv(ori2)*ori1`
- this additional symmetry is called grain exchange symmetry
- in MTEX it is denoted by the option `'antipodal'`

```
mori = grains.boundary.misorientation
mori.antipodal
```

```
ans = misorientation
  size: 8931 x 1
  crystal symmetry : iron (m-3m)
  crystal symmetry : iron (m-3m)
  antipodal:         true
```

Misorientation Angle and Axis

misorientation

```
mori = inv(ori1)*ori2
```

the misorientation angle is the smallest rotational angle of **mori.symmetrise**

```
angle(ori1,ori2)
angle(mori)
```

the rotational angle ignoring symmetry

```
angle(mori,'noSymmetry')
```

the disorientation is the misorientation with

- minimum angle
- axis in the fundamental sector

```
mori.project2FundamentalRegion
```

the rotational axis of `mori` refers to the disorientation

```
axis(mori,csAlpha)
axis(mori,csBeta)
```

the axis in specimen coordinates

```
axis(ori1,ori2)
```

ignore symmetry

```
axis(mori,'noSymmetry')
```

Misorientation Angle and Axis

misorientation

```
mori = inv(ori1)*ori2
```

the misorientation angle is the smallest rotational angle of **mori.symmetrise**

```
angle(ori1,ori2)
angle(mori)
```

the rotational angle ignoring symmetry

```
angle(mori,'noSymmetry')
```

the disorientation is the misorientation with

- minimum angle
- axis in the fundamental sector

```
mori.project2FundamentalRegion
```

the rotational axis of `mori` refers to the disorientation

```
axis(mori,csAlpha)
axis(mori,csBeta)
```

the axis in specimen coordinates

```
axis(ori1,ori2)
```

ignore symmetry

```
axis(mori,'noSymmetry')
```

Misorientation Angle and Axis

misorientation

```
mori = inv(ori1)*ori2
```

the misorientation angle is the smallest rotational angle of **mori.symmetrise**

```
angle(ori1,ori2)
angle(mori)
```

the rotational angle ignoring symmetry

```
angle(mori,'noSymmetry')
```

the disorientation is the misorientation with

- minimum angle
- axis in the fundamental sector

```
mori.project2FundamentalRegion
```

the rotational axis of `mori` refers to the disorientation

```
axis(mori,csAlpha)
axis(mori,csBeta)
```

the axis in specimen coordinates

```
axis(ori1,ori2)
```

ignore symmetry

```
axis(mori,'noSymmetry')
```

Misorientation Angle and Axis

misorientation

```
mori = inv(ori1)*ori2
```

the misorientation angle is the smallest rotational angle of **mori.symmetrise**

```
angle(ori1,ori2)
angle(mori)
```

the rotational angle ignoring symmetry

```
angle(mori,'noSymmetry')
```

the disorientation is the misorientation with

- minimum angle
- axis in the fundamental sector

```
mori.project2FundamentalRegion
```

the rotational axis of `mori` refers to the disorientation

```
axis(mori,csAlpha)
axis(mori,csBeta)
```

the axis in specimen coordinates

```
axis(ori1,ori2)
```

ignore symmetry

```
axis(mori,'noSymmetry')
```

Misorientation Angle and Axis

misorientation

```
mori = inv(ori1)*ori2
```

the misorientation angle is the smallest rotational angle of **mori.symmetrise**

```
angle(ori1,ori2)
angle(mori)
```

the rotational angle ignoring symmetry

```
angle(mori,'noSymmetry')
```

the disorientation is the misorientation with

- minimum angle
- axis in the fundamental sector

```
mori.project2FundamentalRegion
```

the rotational axis of `mori` refers to the disorientation

```
axis(mori,csAlpha)
axis(mori,csBeta)
```

```
 mineral: Ti (alpha) (622)
        h        k        i        l
  -1.5812   2.9291  -1.348   0.5158

ans = Miller (show methods, plot)
 mineral: Ti (BETA) (432)
        h        k        l
  -0.1511   3.2874   0.3649
```

the axis in specimen coordinates

```
axis(ori1,ori2)
```

ignore symmetry

```
axis(mori,'noSymmetry')
```

Misorientation Angle and Axis

misorientation

```
mori = inv(ori1)*ori2
```

the misorientation angle is the smallest rotational angle of **mori.symmetrise**

```
angle(ori1,ori2)
angle(mori)
```

the rotational angle ignoring symmetry

```
angle(mori,'noSymmetry')
```

the disorientation is the misorientation with

- minimum angle
- axis in the fundamental sector

```
mori.project2FundamentalRegion
```

the rotational axis of `mori` refers to the disorientation

```
axis(mori,csAlpha)
axis(mori,csBeta)
```

the axis in specimen coordinates

```
axis(ori1,ori2)
```

```
ans = vector3d (show methods, plot)
 size: 1 x 1
           x          y          z
  0.0495585   0.992437 -0.112305
```

ignore symmetry

```
axis(mori,'noSymmetry')
```

Misorientation Angle and Axis

misorientation

```
mori = inv(ori1)*ori2
```

the misorientation angle is the smallest rotational angle of **mori.symmetrise**

```
angle(ori1,ori2)
angle(mori)
```

the rotational angle ignoring symmetry

```
angle(mori,'noSymmetry')
```

the disorientation is the misorientation with

- minimum angle
- axis in the fundamental sector

```
mori.project2FundamentalRegion
```

the rotational axis of `mori` refers to the disorientation

```
axis(mori,csAlpha)
axis(mori,csBeta)
```

the axis in specimen coordinates

```
axis(ori1,ori2)
```

ignore symmetry

```
axis(mori,'noSymmetry')
```

Axis Angle Coloring

the standard color key

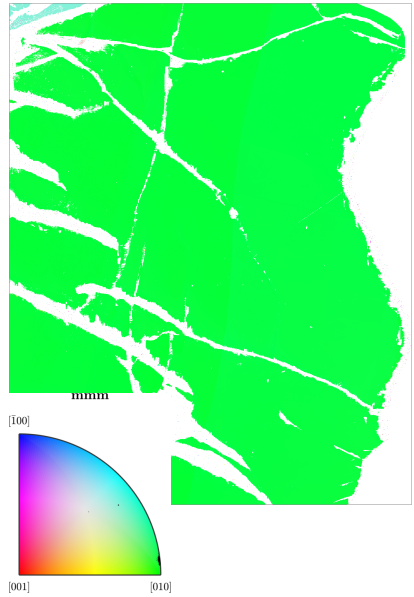
```
ori = ebsd('Fo').orientations
key = ipfHSVKey(ori.CS)
color = key.orientation2color(ori)
```

sharp standard color key

```
key.inversePoleFigureDirection =
        mean(ori)* key.whiteCenter
key.maxAngle = 4*degree
```

sharp axis angle key

```
key =  axisAngleKey
key.oriRef = mean(ori)
key.maxAngle = 4*degree
```



Axis Angle Coloring

the standard color key

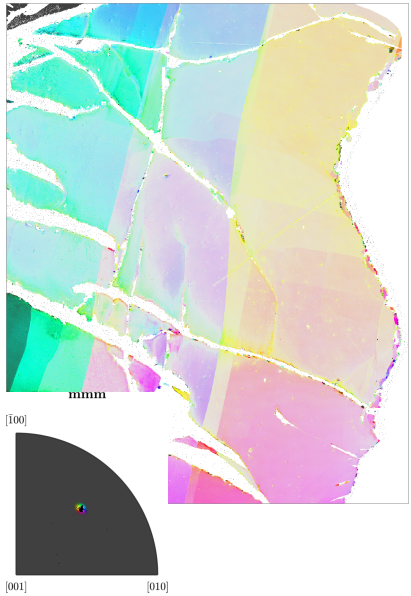
```
ori = ebsd('Fo').orientations
key = ipfHSVKey(ori.CS)
color = key.orientation2color(ori)
```

sharp standard color key

```
key.inversePoleFigureDirection =
        mean(ori)* key.whiteCenter
key.maxAngle = 4*degree
```

sharp axis angle key

```
key =  axisAngleKey
key.oriRef = mean(ori)
key.maxAngle = 4*degree
```



Axis Angle Coloring

the standard color key

```
ori = ebsd('Fo').orientations
key = ipfHSVKey(ori.CS)
color = key.orientation2color(ori)
```

sharp standard color key

```
key.inversePoleFigureDirection =
      mean(ori)* key.whiteCenter
key.maxAngle = 4*degree
```

sharp axis angle key

```
key =  axisAngleKey
key.oriRef = mean(ori)
key.maxAngle = 4*degree
```

Grain Reference Orientation Deviation and mis2mean

- `ori` - orientation at a certain pixel
- `oriG_g` - grain mean orientation
- `S` - symmetry to minimize the angle of

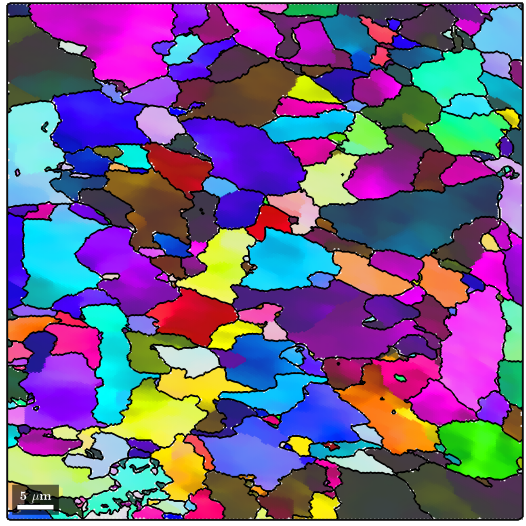
```
GROD = S * inv(ori_g) * ori

grod = calcGROD(ebsd, grains)

GOS = grainMean(ebsd, grod.angle)

axC = grod.axis

axS = ebsd.orientations .* ...
  grod.axis('noSymmetry')
```



Grain Reference Orientation Deviation and mis2mean

- `ori` - orientation at a certain pixel
- `oriG_g` - grain mean orientation
- `S` - symmetry to minimize the angle of

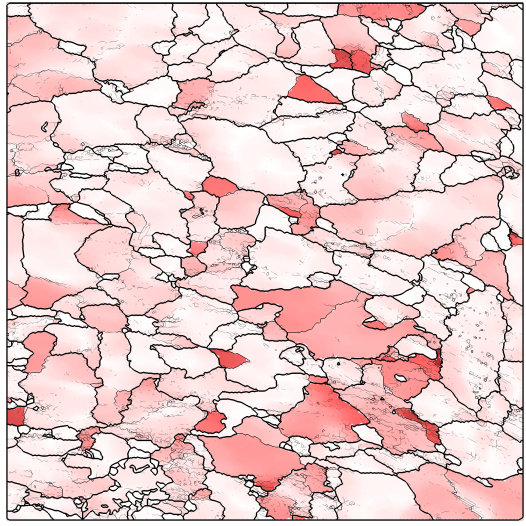
```
GROD = S * inv(ori_g) * ori

grod = calcGROD(ebsd, grains)

GOS = grainMean(ebsd, grod.angle)

axC = grod.axis

axS = ebsd.orientations .* ...
  grod.axis('noSymmetry')
```



Grain Reference Orientation Deviation and mis2mean

- `ori` - orientation at a certain pixel
- `oriG_g` - grain mean orientation
- `S` - symmetry to minimize the angle of

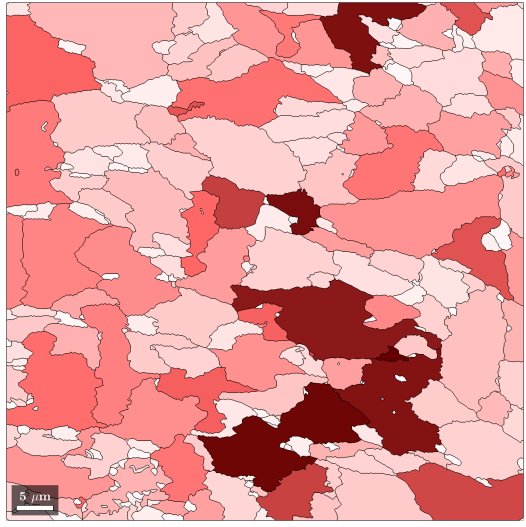
```
GROD = S * inv(ori_g) * ori
```

```
grod = calcGROD(ebsd, grains)
```

```
GOS = grainMean(ebsd, grod.angle)
```

```
axC = grod.axis
```

```
axS = ebsd.orientations .* ...
  grod.axis('noSymmetry')
```



Grain Reference Orientation Deviation and mis2mean

- `ori` - orientation at a certain pixel
- `oriG_g` - grain mean orientation
- `S` - symmetry to minimize the angle of

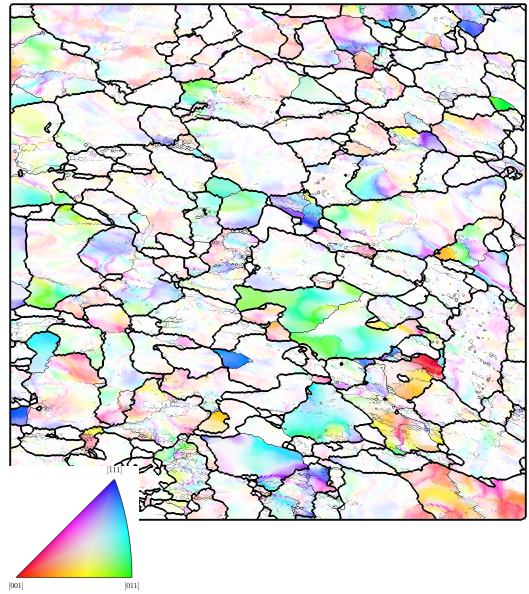
```
GROD = S * inv(ori_g) * ori
```

```
grod = calcGROD(ebsd, grains)
```

```
GOS = grainMean(ebsd, grod.angle)
```

```
axC = grod.axis
```

```
axS = ebsd.orientations .* ...
  grod.axis('noSymmetry')
```



Grain Reference Orientation Deviation and mis2mean

- `ori` - orientation at a certain pixel
- `oriG_g` - grain mean orientation
- `S` - symmetry to minimize the angle of

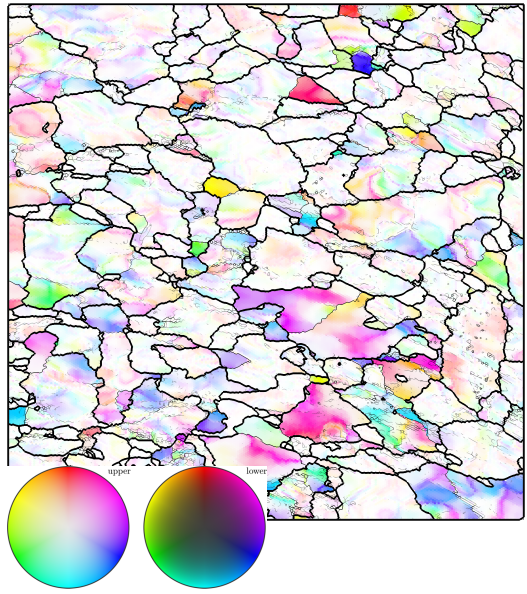
```
GROD = S * inv(ori_g) * ori
```

```
grod = calcGROD(ebsd, grains)
```

```
GOS = grainMean(ebsd, grod.angle)
```

```
axC = grod.axis
```

```
axS = ebsd.orientations .* ...
  grod.axis('noSymmetry')
```



Misorientation Distribution

distribution of boundary misorientation angles

```
mori = grains.boundary.misorientation
histogram(mori.angle./degree)
plotAngleDistribution(mori)
```

uniform misorientation distribution function

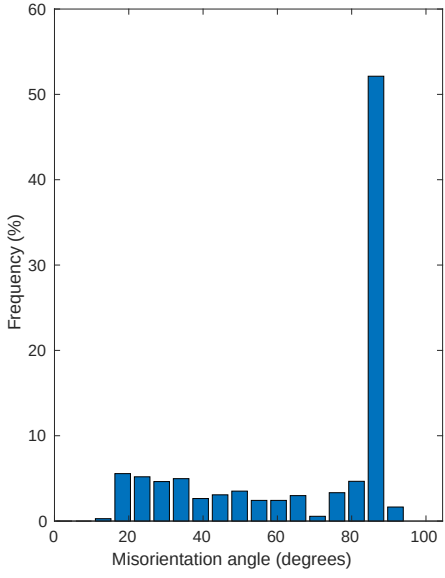
```
plotAngleDistribution(mori.CS)
```

uncorrelated misorientation distribution function

```
odf = calcODF(ebsd.orientations)
umdf = calcMDF(odf)
plotAngleDistribution(umdf)
```

distribution of boundary misorientations

```
bmdf = calcDensity(mori)
plotAngleDistribution(bmdf)
```



Misorientation Distribution

distribution of boundary misorientation angles

```
mori = grains.boundary.misorientation
histogram(mori.angle./degree)
plotAngleDistribution(mori)
```

uniform misorientation distribution function

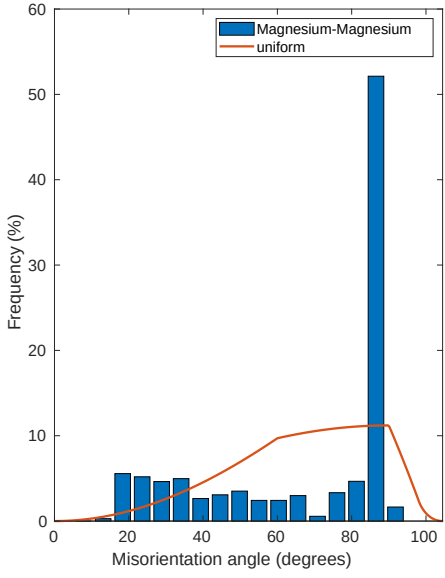
```
plotAngleDistribution(mori.CS)
```

uncorrelated misorientation distribution function

```
odf = calcODF(ebsd.orientations)
umdf = calcMDF(odf)
plotAngleDistribution(umdf)
```

distribution of boundary misorientations

```
bmdf = calcDensity(mori)
plotAngleDistribution(bmdf)
```



Misorientation Distribution

distribution of boundary misorientation angles

```
mori = grains.boundary.misorientation
histogram(mori.angle./degree)
plotAngleDistribution(mori)
```

uniform misorientation distribution function

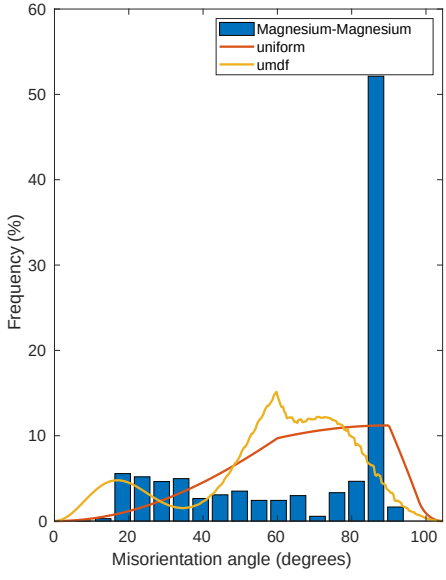
```
plotAngleDistribution(mori.CS)
```

uncorrelated misorientation distribution function

```
odf = calcODF(ebsd.orientations)
umdf = calcMDF(odf)
plotAngleDistribution(umdf)
```

distribution of boundary misorientations

```
bmdf = calcDensity(mori)
plotAngleDistribution(bmdf)
```



Misorientation Distribution

distribution of boundary misorientation angles

```
mori = grains.boundary.misorientation
histogram(mori.angle./degree)
plotAngleDistribution(mori)
```

uniform misorientation distribution function

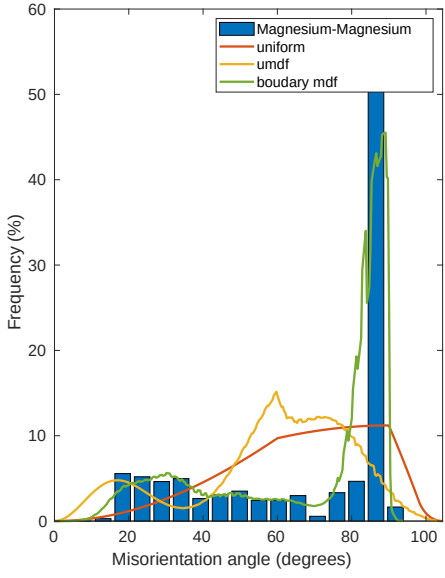
```
plotAngleDistribution(mori.CS)
```

uncorrelated misorientation distribution function

```
odf = calcODF(ebsd.orientations)
umdf = calcMDF(odf)
plotAngleDistribution(umdf)
```

distribution of boundary misorientations

```
bmdf = calcDensity(mori)
plotAngleDistribution(bmdf)
```



Misorientation Axis Distribution

distribution of boundary misorientation angles

```
plotAxisDistribution(mori)
```

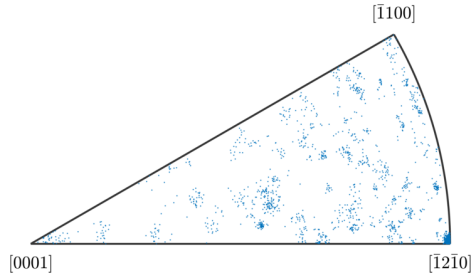
```
plotAxisDistribution(mori,'contourf')
```

uniform misorientation distribution function

```
plotAxisDistribution(mori.CS)
```

uncorrelated misorientation distribution function
texture dependent

```
plotAxisDistribution(umdf)
```



Misorientation Axis Distribution

distribution of boundary misorientation angles

```
plotAxisDistribution(mori)
```

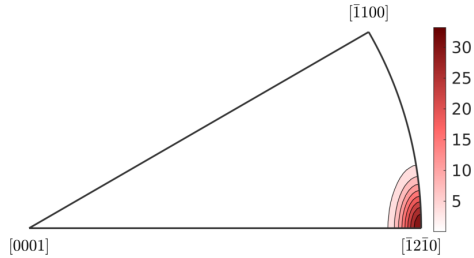
```
plotAxisDistribution(mori,'contourf')
```

uniform misorientation distribution function

```
plotAxisDistribution(mori.CS)
```

uncorrelated misorientation distribution function
texture dependent

```
plotAxisDistribution(umdf)
```

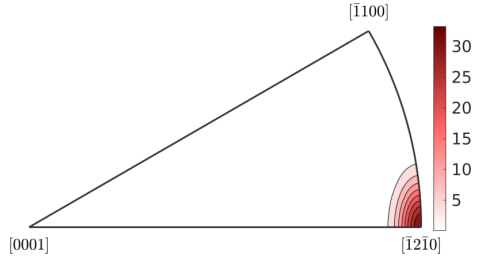


Misorientation Axis Distribution

distribution of boundary misorientation angles

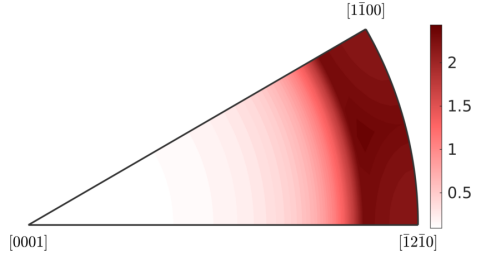
```
plotAxisDistribution(mori)
```

```
plotAxisDistribution(mori,'contourf')
```



uniform misorientation distribution function

```
plotAxisDistribution(mori.CS)
```



uncorrelated misorientation distribution function

texture dependent

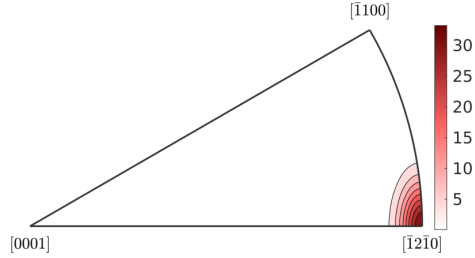
```
plotAxisDistribution(umdf)
```

Misorientation Axis Distribution

distribution of boundary misorientation angles

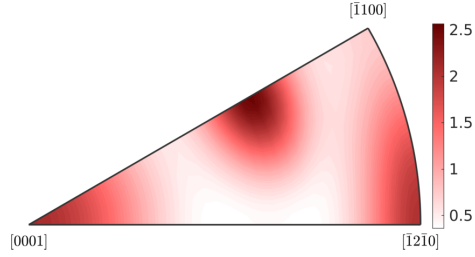
```
plotAxisDistribution(mori)
```

```
plotAxisDistribution(mori,'contourf')
```



uniform misorientation distribution function

```
plotAxisDistribution(mori.CS)
```



uncorrelated misorientation distribution function
texture dependent

```
plotAxisDistribution(umdf)
```

Phase Transition

The Burger OR translates beta into alpha coordinates

```
b2a = orientation.burger(csBeta,csAlpha)
b2a * Miller(0,0,1,csBeta)
```

consider an alpha orientation

```
oriAlpha = orientation.Id(csAlpha)
oriAlpha * b2a * Miller(0,0,1,csBeta)
```

Hence, we can compute a beta orientation out of

```
oriBeta = oriAlpha * b2a
```

conversely, we obtain an alpha orientation by

```
oriAlpha = oriBeta * inv(b2a)
```

Phase Transition

The Burger OR translates beta into alpha coordinates

```
b2a = orientation.burger(csBeta,csAlpha)
b2a * Miller(0,0,1,csBeta)
```

consider an alpha orientation

```
oriAlpha = orientation.Id(csAlpha)
oriAlpha * b2a * Miller(0,0,1,csBeta)
```

Hence, we can compute a beta orientation out of

```
oriBeta = oriAlpha * b2a
```

conversely, we obtain an alpha orientation by

```
oriAlpha = oriBeta * inv(b2a)
```

Phase Transition

The Burger OR translates beta into alpha coordinates

```
b2a = orientation.burger(csBeta,csAlpha)
b2a * Miller(0,0,1,csBeta)
```

consider an alpha orientation

```
oriAlpha = orientation.Id(csAlpha)
oriAlpha * b2a * Miller(0,0,1,csBeta)
```

Hence, we can compute a beta orientation out of

```
oriBeta = oriAlpha * b2a
```

conversely, we obtain an alpha orientation by

```
oriAlpha = oriBeta * inv(b2a)
```

Phase Transition

The Burger OR translates beta into alpha coordinates

```
b2a = orientation.burger(csBeta,csAlpha)
b2a * Miller(0,0,1,csBeta)
```

consider an alpha orientation

```
oriAlpha = orientation.Id(csAlpha)
oriAlpha * b2a * Miller(0,0,1,csBeta)
```

Hence, we can compute a beta orientation out of

```
oriBeta = oriAlpha * b2a
```

conversely, we obtain an alpha orientation by

```
oriAlpha = oriBeta * inv(b2a)
```

Variants

Due to symmetry the Burger OR translates a single beta orientation into several alpha orientations

```
oriAlpha = oriBeta * inv(b2a.symmetrise)
oriAlpha = oriBeta.symmetrise * inv(b2a)
```

Some of these alpha orientations at not symmetrically equivalent. The are called variants of `oriBeta`

```
alphaV = variants(b2a, oriBeta)
```

Lets compare alpha and beta orientations

```
plotPDF(oriBeta, Miller(1,1,0,csBeta))
hAlpha = symmetrise(Miller(0,0,0,1,csAlpha)
plotPDF(alphaV * hAlpha)
```

Variants

Due to symmetry the Burger OR translates a single beta orientation into several alpha orientations

```
oriAlpha = oriBeta * inv(b2a.symmetrise)
oriAlpha = oriBeta.symmetrise * inv(b2a)
```

Some of these alpha orientations at not symmetrically equivalent. The are called variants of `oriBeta`

```
alphaV = variants(b2a, oriBeta)
```

Lets compare alpha and beta orientations

```
plotPDF(oriBeta, Miller(1,1,0,csBeta))
hAlpha = symmetrise(Miller(0,0,0,1,csAlpha)
plotPDF(alphaV * hAlpha)
```

Variants

Due to symmetry the Burger OR translates a single beta orientation into several alpha orientations

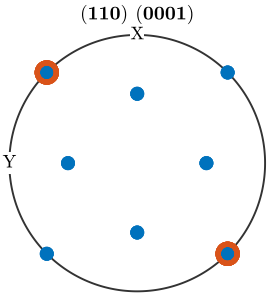
```
oriAlpha = oriBeta * inv(b2a.symmetrise)
oriAlpha = oriBeta.symmetrise * inv(b2a)
```

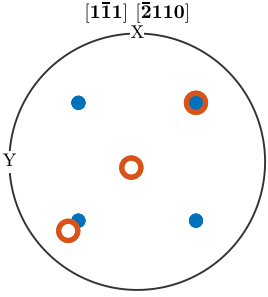
Some of these alpha orientations at not symmetrically equivalent. The are called variants of `oriBeta`

```
alphaV = variants(b2a, oriBeta)
```

Lets compare alpha and beta orientations

```
plotPDF(oriBeta, Miller(1,1,0,csBeta))
hAlpha = symmetrise(Miller(0,0,0,1,csAlpha)
plotPDF(alphaV * hAlpha)
```





Variants

Due to symmetry the Burger OR translates a single beta orientation into several alpha orientations

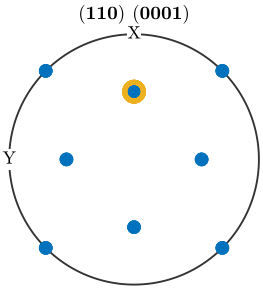
```
oriAlpha = oriBeta * inv(b2a.symmetrise)
oriAlpha = oriBeta.symmetrise * inv(b2a)
```

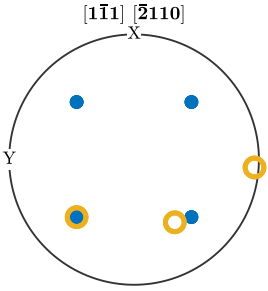
Some of these alpha orientations at not symmetrically equivalent. The are called variants of `oriBeta`

```
alphaV = variants(b2a, oriBeta)
```

Lets compare alpha and beta orientations

```
plotPDF(oriBeta, Miller(1,1,0,csBeta))
hAlpha = symmetrise(Miller(0,0,0,1,csAlpha)
plotPDF(alphaV * hAlpha)
```





Variants

Due to symmetry the Burger OR translates a single beta orientation into several alpha orientations

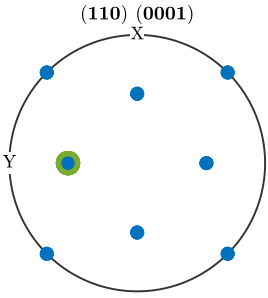
```
oriAlpha = oriBeta * inv(b2a.symmetrise)
oriAlpha = oriBeta.symmetrise * inv(b2a)
```

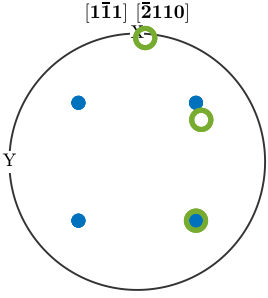
Some of these alpha orientations at not symmetrically equivalent. The are called variants of `oriBeta`

```
alphaV = variants(b2a, oriBeta)
```

Lets compare alpha and beta orientations

```
plotPDF(oriBeta, Miller(1,1,0,csBeta))
hAlpha = symmetrise(Miller(0,0,0,1,csAlpha)
plotPDF(alphaV * hAlpha)
```





Variants

Due to symmetry the Burger OR translates a single beta orientation into several alpha orientations

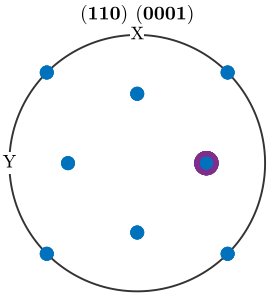
```
oriAlpha = oriBeta * inv(b2a.symmetrise)
oriAlpha = oriBeta.symmetrise * inv(b2a)
```

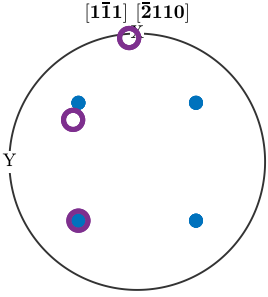
Some of these alpha orientations at not symmetrically equivalent. The are called variants of `oriBeta`

```
alphaV = variants(b2a, oriBeta)
```

Lets compare alpha and beta orientations

```
plotPDF(oriBeta, Miller(1,1,0,csBeta))
hAlpha = symmetrise(Miller(0,0,0,1,csAlpha)
plotPDF(alphaV * hAlpha)
```





Variants

Due to symmetry the Burger OR translates a single beta orientation into several alpha orientations

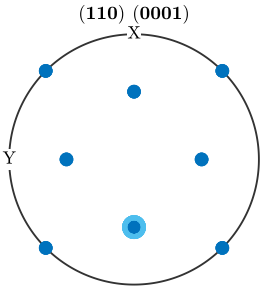
```
oriAlpha = oriBeta * inv(b2a.symmetrise)
oriAlpha = oriBeta.symmetrise * inv(b2a)
```

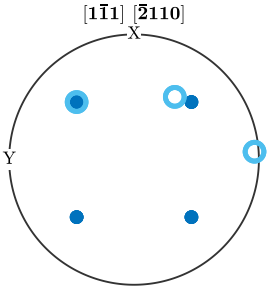
Some of these alpha orientations at not symmetrically equivalent. The are called variants of `oriBeta`

```
alphaV = variants(b2a, oriBeta)
```

Lets compare alpha and beta orientations

```
plotPDF(oriBeta, Miller(1,1,0,csBeta))
hAlpha = symmetrise(Miller(0,0,0,1,csAlpha)
plotPDF(alphaV * hAlpha)
```





Variants

Due to symmetry the Burger OR translates a single beta orientation into several alpha orientations

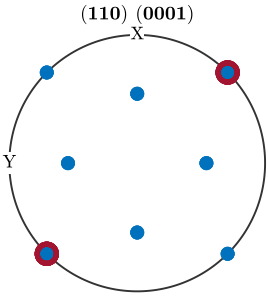
```
oriAlpha = oriBeta * inv(b2a.symmetrise)
oriAlpha = oriBeta.symmetrise * inv(b2a)
```

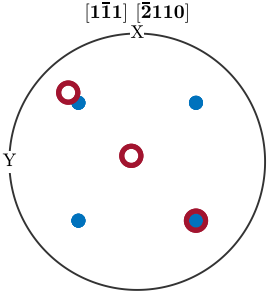
Some of these alpha orientations at not symmetrically equivalent. The are called variants of `oriBeta`

```
alphaV = variants(b2a, oriBeta)
```

Lets compare alpha and beta orientations

```
plotPDF(oriBeta, Miller(1,1,0,csBeta))
hAlpha = symmetrise(Miller(0,0,0,1,csAlpha)
plotPDF(alphaV * hAlpha)
```





Variants

Due to symmetry the Burger OR translates a single beta orientation into several alpha orientations

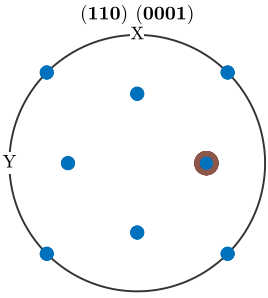
```
oriAlpha = oriBeta * inv(b2a.symmetrise)
oriAlpha = oriBeta.symmetrise * inv(b2a)
```

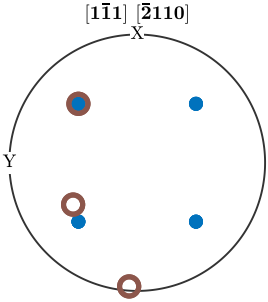
Some of these alpha orientations at not symmetrically equivalent. The are called variants of `oriBeta`

```
alphaV = variants(b2a, oriBeta)
```

Lets compare alpha and beta orientations

```
plotPDF(oriBeta, Miller(1,1,0,csBeta))
hAlpha = symmetrise(Miller(0,0,0,1,csAlpha)
plotPDF(alphaV * hAlpha)
```





Variants

Due to symmetry the Burger OR translates a single beta orientation into several alpha orientations

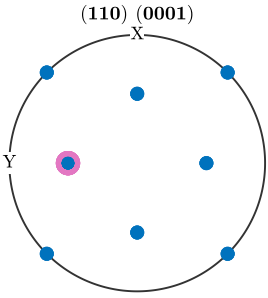
```
oriAlpha = oriBeta * inv(b2a.symmetrise)
oriAlpha = oriBeta.symmetrise * inv(b2a)
```

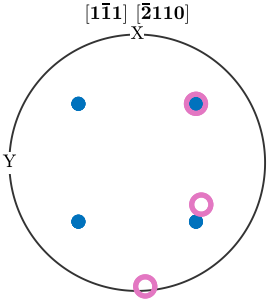
Some of these alpha orientations at not symmetrically equivalent. The are called variants of `oriBeta`

```
alphaV = variants(b2a, oriBeta)
```

Lets compare alpha and beta orientations

```
plotPDF(oriBeta, Miller(1,1,0,csBeta))
hAlpha = symmetrise(Miller(0,0,0,1,csAlpha)
plotPDF(alphaV * hAlpha)
```





Variants

Due to symmetry the Burger OR translates a single beta orientation into several alpha orientations

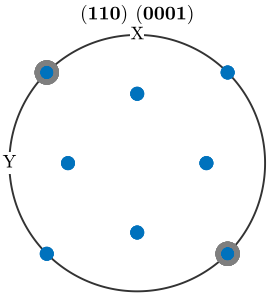
```
oriAlpha = oriBeta * inv(b2a.symmetrise)
oriAlpha = oriBeta.symmetrise * inv(b2a)
```

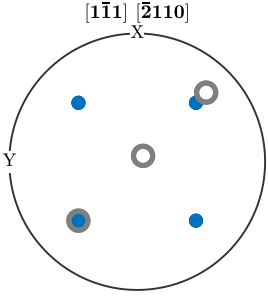
Some of these alpha orientations at not symmetrically equivalent. The are called variants of `oriBeta`

```
alphaV = variants(b2a, oriBeta)
```

Lets compare alpha and beta orientations

```
plotPDF(oriBeta, Miller(1,1,0,csBeta))
hAlpha = symmetrise(Miller(0,0,0,1,csAlpha)
plotPDF(alphaV * hAlpha)
```





Variants

Due to symmetry the Burger OR translates a single beta orientation into several alpha orientations

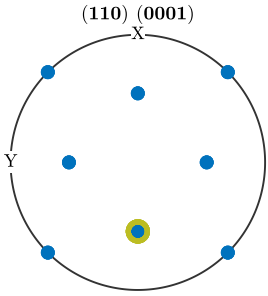
```
oriAlpha = oriBeta * inv(b2a.symmetrise)
oriAlpha = oriBeta.symmetrise * inv(b2a)
```

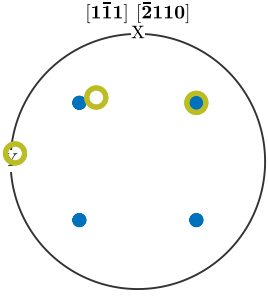
Some of these alpha orientations at not symmetrically equivalent. The are called variants of `oriBeta`

```
alphaV = variants(b2a, oriBeta)
```

Lets compare alpha and beta orientations

```
plotPDF(oriBeta, Miller(1,1,0,csBeta))
hAlpha = symmetrise(Miller(0,0,0,1,csAlpha)
plotPDF(alphaV * hAlpha)
```





Variants

Due to symmetry the Burger OR translates a single beta orientation into several alpha orientations

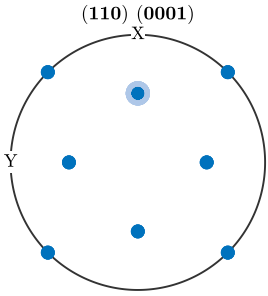
```
oriAlpha = oriBeta * inv(b2a.symmetrise)
oriAlpha = oriBeta.symmetrise * inv(b2a)
```

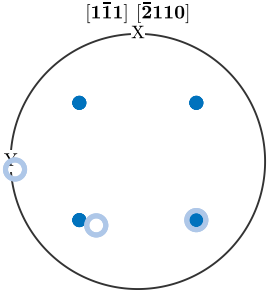
Some of these alpha orientations at not symmetrically equivalent. The are called variants of `oriBeta`

```
alphaV = variants(b2a, oriBeta)
```

Lets compare alpha and beta orientations

```
plotPDF(oriBeta, Miller(1,1,0,csBeta))
hAlpha = symmetrise(Miller(0,0,0,1,csAlpha)
plotPDF(alphaV * hAlpha)
```





Variants

Due to symmetry the Burger OR translates a single beta orientation into several alpha orientations

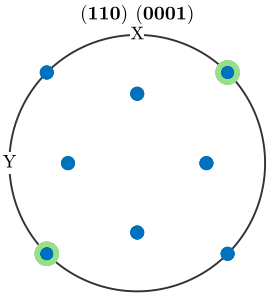
```
oriAlpha = oriBeta * inv(b2a.symmetrise)
oriAlpha = oriBeta.symmetrise * inv(b2a)
```

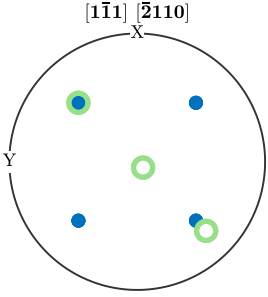
Some of these alpha orientations at not symmetrically equivalent. The are called variants of `oriBeta`

```
alphaV = variants(b2a, oriBeta)
```

Lets compare alpha and beta orientations

```
plotPDF(oriBeta, Miller(1,1,0,csBeta))
hAlpha = symmetrise(Miller(0,0,0,1,csAlpha)
plotPDF(alphaV * hAlpha)
```





Variants

Due to symmetry the Burger OR translates a single beta orientation into several alpha orientations

```
oriAlpha = oriBeta * inv(b2a.symmetrise)
oriAlpha = oriBeta.symmetrise * inv(b2a)
```

Some of these alpha orientations at not symmetrically equivalent. The are called variants of `oriBeta`

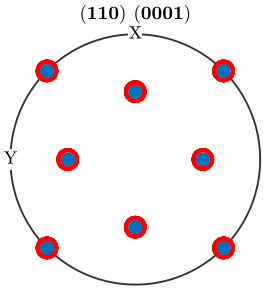
```
alphaV = variants(b2a, oriBeta)
```

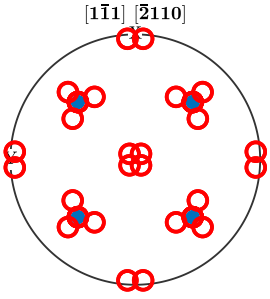
Lets compare alpha and beta orientations

```
plotPDF(oriBeta, Miller(1,1,0,csBeta))
hAlpha = symmetrise(Miller(0,0,0,1,csAlpha)
plotPDF(alphaV * hAlpha)
```

determine which variant we have

```
id = calcVariantId(oriBeta, oriAlpha, b2a)
```





Child to Child Misorientations

assume two child variants

- `oriA1 = oriB * SymB1 * inv(b2a)`
- `oriA2 = oriB * SymB2 * inv(b2a)`

then its misorientation is

$$\begin{aligned} &\mathtt{inv(oriA1)} * \mathtt{oriA2} \\ &= \mathtt{b2a} * \mathtt{symB1} * \mathtt{inv(oriB)} * \mathtt{oriB} * \mathtt{symB2} * \mathtt{inv(b2a)} \\ &= \mathtt{b2a} * \mathtt{symB} * \mathtt{inv(b2a)} \end{aligned}$$

list of all possibe child to child misorientations

```
c2c = b2a.variants * b2a
```

compute fit and parent orientation

Child to Child Misorientations

assume two child variants

- `oriA1 = oriB * SymB1 * inv(b2a)`
- `oriA2 = oriB * SymB2 * inv(b2a)`

then its misorientation is

$$\begin{aligned} &\texttt{inv(oriA1)} * \texttt{oriA2} \\ &= \texttt{b2a} * \texttt{symB1} * \texttt{inv(oriB)} * \texttt{oriB} * \texttt{symB2} * \texttt{inv(b2a)} \\ &= \texttt{b2a} * \texttt{symB} * \texttt{inv(b2a)} \end{aligned}$$

list of all possibe child to child misorientations

```
c2c = b2a.variants * b2a
```

compute fit and parent orientation

Child to Child Misorientations

assume two child variants

- `oriA1 = oriB * SymB1 * inv(b2a)`
- `oriA2 = oriB * SymB2 * inv(b2a)`

then its misorientation is

$$\begin{aligned} &\texttt{inv(oriA1)} * \texttt{oriA2} \\ &= \texttt{b2a} * \texttt{symB1} * \texttt{inv(oriB)} * \texttt{oriB} * \texttt{symB2} * \texttt{inv(b2a)} \\ &= \texttt{b2a} * \texttt{symB} * \texttt{inv(b2a)} \end{aligned}$$

list of all possibe child to child misorientations

```
c2c = b2a.variants * b2a
```

compute fit and parent orientation

Child to Child Misorientations

assume two child variants

- `oriA1 = oriB * SymB1 * inv(b2a)`
- `oriA2 = oriB * SymB2 * inv(b2a)`

then its misorientation is

$$\begin{aligned} &\texttt{inv(oriA1)} * \texttt{oriA2} \\ &= \texttt{b2a} * \texttt{symB1} * \texttt{inv(oriB)} * \texttt{oriB} * \texttt{symB2} * \texttt{inv(b2a)} \\ &= \texttt{b2a} * \texttt{symB} * \texttt{inv(b2a)} \end{aligned}$$

list of all possibe child to child misorientations

```
c2c = b2a.variants * b2a
```

compute fit and parent orientation

Child to Child Misorientations

assume two child variants

- `oriA1 = oriB * SymB1 * inv(b2a)`
- `oriA2 = oriB * SymB2 * inv(b2a)`

then its misorientation is

$$\begin{aligned} &\mathtt{inv(oriA1)} * \mathtt{oriA2} \\ &= \mathtt{b2a} * \mathtt{symB1} * \mathtt{inv(oriB)} * \mathtt{oriB} * \mathtt{symB2} * \mathtt{inv(b2a)} \\ &= \mathtt{b2a} * \mathtt{symB} * \mathtt{inv(b2a)} \end{aligned}$$

list of all possibe child to child misorientations

```
c2c = b2a.variants * b2a
```

compute fit and parent orientation

```
[oriB, fit] = calcParent([oriA1,oriA2], b2a)
```

Child to Child Misorientations

assume two child variants

- `oriA1 = oriB * SymB1 * inv(b2a)`
- `oriA2 = oriB * SymB2 * inv(b2a)`

then its misorientation is

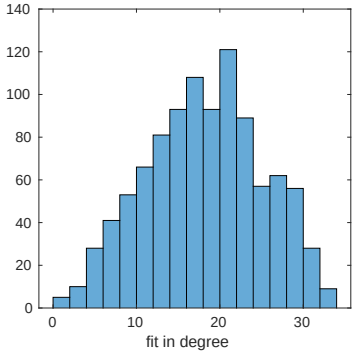
$$\begin{aligned} &\texttt{inv(oriA1)} * \texttt{oriA2} \\ &= \texttt{b2a} * \texttt{symB1} * \texttt{inv(oriB)} * \texttt{oriB} * \texttt{symB2} * \texttt{inv(b2a)} \\ &= \texttt{b2a} * \texttt{symB} * \texttt{inv(b2a)} \end{aligned}$$

list of all possibe child to child misorientations

```
c2c = b2a.variants * b2a
```

compute fit and parent orientation

```
oriA = orientation.rand(1000,2,csAlpha);
[~,fit] = calcParent(oriA,beta2alpha);
```



Child to Child Misorientations

assume two child variants

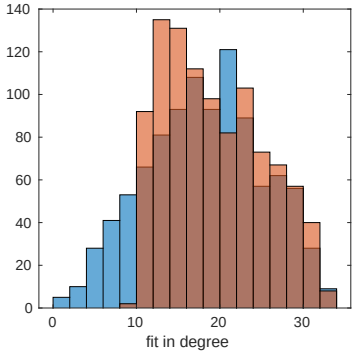
- `oriA1 = oriB * SymB1 * inv(b2a)`
- `oriA2 = oriB * SymB2 * inv(b2a)`

then its misorientation is

$$\begin{aligned} &\texttt{inv(oriA1)} * \texttt{oriA2} \\ &= \texttt{b2a} * \texttt{symB1} * \texttt{inv(oriB)} * \texttt{oriB} * \texttt{symB2} * \texttt{inv(b2a)} \\ &= \texttt{b2a} * \texttt{symB} * \texttt{inv(b2a)} \end{aligned}$$

list of all possibe child to child misorientations

```
c2c = b2a.variants * b2a
```

compute fit and parent orientation

```
oriA = orientation.rand(1000,3,csAlpha);
[~,fit] = calcParent(oriA,beta2alpha);
```