Crystal Geometry

R. Hielscher

Faculty of Mathematics,
Chemnitz University of Technology, Germany

Chemnitz **MTEX** Workshop 2015

Table of Content

Three dimensional vectors - The MTEX Class vector3d

Three dimensional vectors are given by there coordinates with respect to a orthogonal coordinate system $\vec{X}, \vec{Y}, \vec{Z}$

$$\vec{r} = x \cdot \vec{X} + y \cdot \vec{Y} + z \cdot \vec{Z}$$

For general vectors, MTEX does not care about the coordinate system, but works only with the coordinates.

```
r = vector3d(1,2,3)
```

The alignment of the coordinate system is only important when plotting data

Only for directions relative to the crystal coordinate system the reference frame is considered.

Three dimensional vectors - The MTEX Class vector3d

Three dimensional vectors are given by there coordinates with respect to a orthogonal coordinate system $\vec{X}, \vec{Y}, \vec{Z}$

$$\vec{r} = x \cdot \vec{X} + y \cdot \vec{Y} + z \cdot \vec{Z}$$

For general vectors, MTEX does not care about the coordinate system, but works only with the coordinates.

```
r = vector3d(1,2,3)
```

```
r = vector3d (show methods, plot)
  size: 1 x 1
  x y z
  1 2 3
```

The alignment of the coordinate system is only important when plotting data

Only for directions relative to the crystal coordinate system the reference frame is considered.

Three dimensional vectors - The MTEX Class vector3d

Three dimensional vectors are given by there coordinates with respect to a orthogonal coordinate system $\vec{X}, \vec{Y}, \vec{Z}$

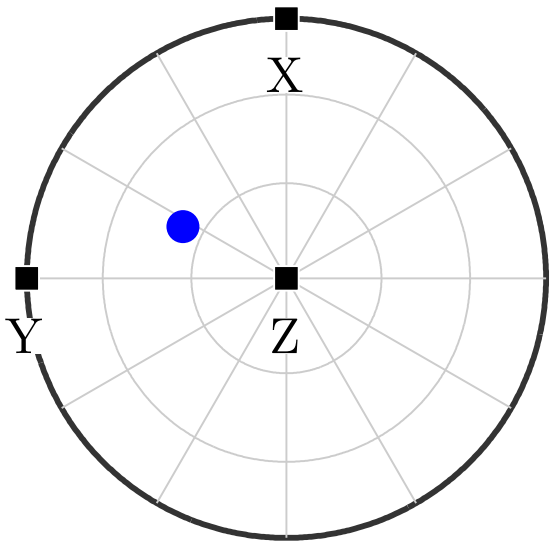
$$\vec{r} = x \cdot \vec{X} + y \cdot \vec{Y} + z \cdot \vec{Z}$$

For general vectors, MTEX does not care about the coordinate system, but works only with the coordinates.

```
r = vector3d(1,2,3)
```

The alignment of the coordinate system is only important when plotting data

```
plotx2north, plotzOutOfPlane
plot(r)
```



Only for directions relative to the crystal coordinate system the reference frame is considered.

Three dimensional vectors - The MTEX Class vector3d

Three dimensional vectors are given by there coordinates with respect to a orthogonal coordinate system $\vec{X}, \vec{Y}, \vec{Z}$

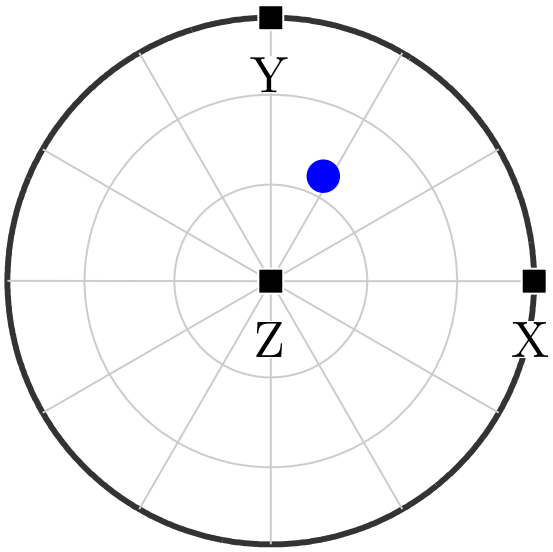
$$\vec{r} = x \cdot \vec{X} + y \cdot \vec{Y} + z \cdot \vec{Z}$$

For general vectors, MTEX does not care about the coordinate system, but works only with the coordinates.

```
r = vector3d(1,2,3)
```

The alignment of the coordinate system is only important when plotting data

```
plotx2east, plotzOutOfPlane
plot(r)
```



Only for directions relative to the crystal coordinate system the reference frame is considered.

Defining vectors

polar coordinates $\vec{r} = (\sin\theta\cos\rho, \sin\theta\sin\rho, \cos\theta)^t$

```
r = vector3d('theta',theta,'rho',rho)
```

predefined vectors

```
r = xvector, r = yvector, r = zvector
```

combine vectors

```
r = [xvector, yvector, vector3d(1,1,1)];
```

importing vectors

```
r = loadVector3d('file','ColumnNames',{'x','y','z'})
```

Defining vectors

polar coordinates $\vec{r} = (\sin\theta\cos\rho, \sin\theta\sin\rho, \cos\theta)^t$

```
r = vector3d('theta',theta,'rho',rho)
```

predefined vectors

```
r = xvector, r = yvector, r = zvector
```

combine vectors

```
r = [xvector, yvector, vector3d(1,1,1)];
```

importing vectors

```
r = loadVector3d('file','ColumnNames',{'x','y','z'})
```

Defining vectors

polar coordinates $\vec{r} = (\sin\theta\cos\rho, \sin\theta\sin\rho, \cos\theta)^t$

```
r = vector3d('theta',theta,'rho',rho)
```

predefined vectors

```
r = xvector, r = yvector, r = zvector
```

combine vectors

```
r = [xvector, yvector, vector3d(1,1,1)];
```

```
r = vector3d (show methods, plot)
  size: 1 x 3
  x y z
  1 0 0
  0 1 0
  1 1 1
```

importing vectors

```
r = loadVector3d('file','ColumnNames',{'x','y','z'})
```

Defining vectors

polar coordinates $\vec{r} = (\sin\theta\cos\rho, \sin\theta\sin\rho, \cos\theta)^t$

```
r = vector3d('theta',theta,'rho',rho)
```

predefined vectors

```
r = xvector, r = yvector, r = zvector
```

combine vectors

```
r = [xvector, yvector, vector3d(1,1,1)];
```

importing vectors

```
r = loadVector3d('file','ColumnNames',{'x','y','z'})
```

```
r = vector3d (show methods, plot)
  size: 200 x 1
```

Vector Calculations

simple algebra

```
r = 2*xvector - yvector;
```

basic operations

```
dot(v1,v2)    % dot product
cross(v1,v2)  % cross product
angle(v1,v2)  % angle between two vectors
```

extract properties

```
r.theta       % polar angle in radiant
r.rho         % azimuth angle in radiant
r.x, r.y, r.z
```

Vector Calculations

simple algebra

```
r = 2*xvector - yvector;
```

basic operations

```
dot(v1,v2)    % dot product
cross(v1,v2)  % cross product
angle(v1,v2)  % angle between two vectors
```

extract properties

```
r.theta       % polar angle in radiant
r.rho         % azimuth angle in radiant
r.x, r.y, r.z
```

Vector Calculations

simple algebra

```
r = 2*xvector - yvector;
```

basic operations

```
dot(v1,v2)    % dot product
cross(v1,v2)  % cross product
angle(v1,v2)  % angle between two vectors
```

extract properties

```
r.theta       % polar angle in radiant
r.rho         % azimuth angle in radiant
r.x, r.y, r.z
```

Indexing of Vectors

consider a list of vectors

```
r = vector3d([0 0 1 1],[1 0 1 1],[1 1 1 0]);
```

```
r = vector3d (show methods, plot)
  size: 1 x 4
  x y z
  0 1 1
  0 0 1
  1 1 1
  1 1 0
```

single out the second vector

```
r(2)
```

single out the second and the fourth vector

```
r([2 4])
```

single out vectors by a logical condition

Indexing of Vectors

consider a list of vectors

```
r = vector3d([0 0 1 1],[1 0 1 1],[1 1 1 0]);
```

single out the second vector

```
r(2)
```

```
r = vector3d (show methods, plot)
  size: 1 x 1
  x y z
  0 0 1
```

single out the second and the fourth vector

```
r([2 4])
```

single out vectors by a logical condition

```
r(r.x>0)
```

The above techniques applies also to lists of orientations, EBSD data

Indexing of Vectors

consider a list of vectors

```
r = vector3d([0 0 1 1],[1 0 1 1],[1 1 1 0]);
```

single out the second vector

```
r(2)
```

single out the second and the fourth vector

```
r([2 4])
```

```
r = vector3d (show methods, plot)
  size: 1 x 2
  x y z
  0 0 1
  1 1 0
```

single out vectors by a logical condition

```
r(r.x>0)
```

Indexing of Vectors

consider a list of vectors

```
r = vector3d([0 0 1 1],[1 0 1 1],[1 1 1 0]);
```

single out the second vector

```
r(2)
```

single out the second and the fourth vector

```
r([2 4])
```

single out vectors by a logical condition

```
r(r.x>0)
```

```
r = vector3d (show methods, plot)
  size: 1 x 2
  x y z
  1 1 1
  1 1 0
```

Indexing of Vectors

consider a list of vectors

```
r = vector3d([0 0 1 1],[1 0 1 1],[1 1 1 0]);
```

single out the second vector

```
r(2)
```

single out the second and the fourth vector

```
r([2 4])
```

single out vectors by a logical condition

```
r(r.x>0)
```

The above techniques applies also to lists of orientations, EBSD data, grains, boundary segments, etc.

Changing Vectors

consider again the list of vectors

```
r = vector3d([0 0 1 1],[1 0 1 1],[1 1 1 0]);
```

```
r = vector3d (show methods, plot)
  size: 1 x 4
  x y z
  0 1 1
  0 0 1
  1 1 1
  1 1 0
```

replace the second vector by another vector

```
r(2) = yvector
```

remove the second vector completely

```
r(2) = []
```

change the x coordinate of all vectors

Changing Vectors

consider again the list of vectors

```
r = vector3d([0 0 1 1],[1 0 1 1],[1 1 1 0]);
```

replace the second vector by another vector

```
r(2) = yvector
```

```
r = vector3d (show methods, plot)
  size: 1 x 4
  x y z
  0 1 1
  0 1 0
  1 1 1
  1 1 0
```

remove the second vector completely

```
r(2) = []
```

change the x coordinate of all vectors

Changing Vectors

consider again the list of vectors

```
r = vector3d([0 0 1 1],[1 0 1 1],[1 1 1 0]);
```

replace the second vector by another vector

```
r(2) = yvector
```

remove the second vector completely

```
r(2) = []
```

```
r = vector3d (show methods, plot)
  size: 1 x 3
  0 1 1
  1 1 1
  1 1 0
```

change the x coordinate of all vectors

```
r.x = 0
```

Changing Vectors

consider again the list of vectors

```
r = vector3d([0 0 1 1],[1 0 1 1],[1 1 1 0]);
```

replace the second vector by another vector

```
r(2) = yvector
```

remove the second vector completely

```
r(2) = []
```

change the x coordinate of all vectors

```
r.x = 0
```

```
r = vector3d (show methods, plot)
  size: 1 x 3
  0 1 1
  0 1 1
  0 1 0
```

Changing Vectors

consider again the list of vectors

```
r = vector3d([0 0 1 1],[1 0 1 1],[1 1 1 0]);
```

replace the second vector by another vector

```
r(2) = yvector
```

remove the second vector completely

```
r(2) = []
```

change the x coordinate of all vectors

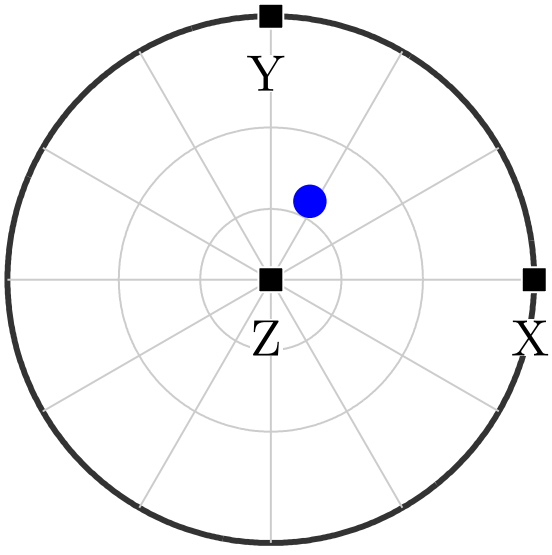
```
r.x = 0
```

The above techniques applies also to pole figure data, orientations, EBSD data, grains, etc.

Plotting Vectors

spherical projections: `earea`, `edist`, `eangle`
hemisphere: `upper`, `lower`

```
plot(r,'projection','eangle','upper')
```



combined plots

```
plot(vector3d(1,1,1),'upper');
hold all
plot(vector3d(1,2,3),'label','B');
plot(vector3d(-1,2,1),'label','A');
hold off
```

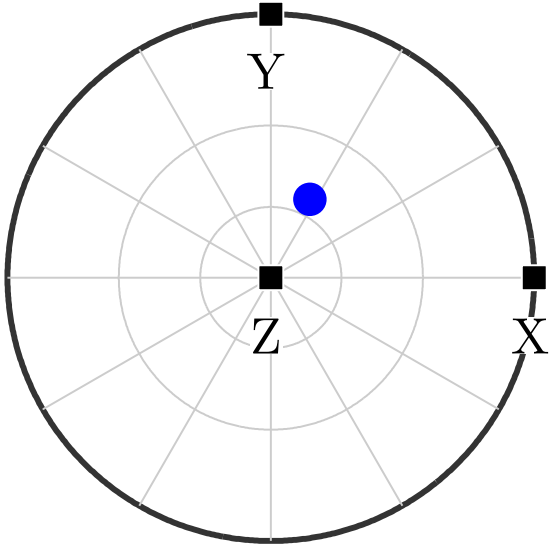
data plots

```
r = randv(100)  % 100 random points
plot(r,1:100)   % colored by number
quiver(r,orth(r)) % a vector field
```

Plotting Vectors

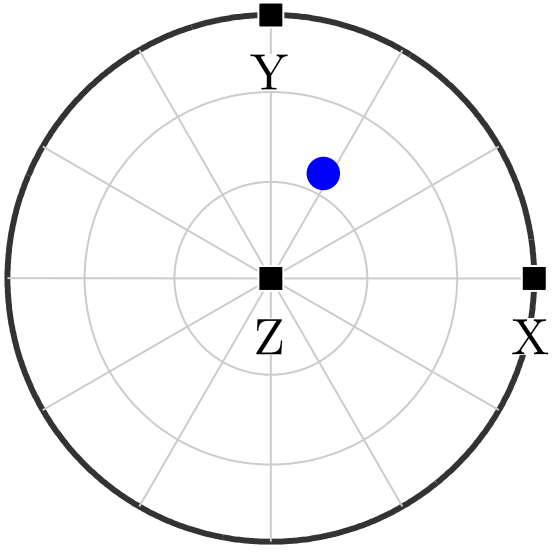
spherical projections: `earea`, `edist`, `eangle`
hemisphere: `upper`, `lower`

```
plot(r,'projection','earea','upper')
```



combined plots

```
plot(vector3d(1,1,1),'upper');
hold all
plot(vector3d(1,2,3),'label','B');
plot(vector3d(-1,2,1),'label','A');
hold off
```



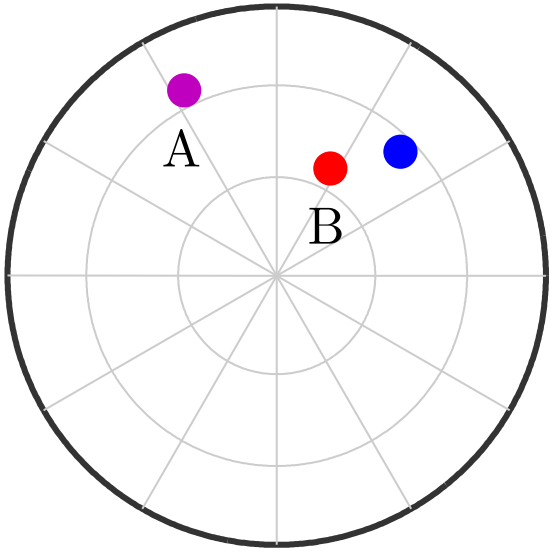
data plots

```
r = randv(100)  % 100 random points
plot(r,1:100)   % colored by number
quiver(r,orth(r)) % a vector field
```

Plotting Vectors

spherical projections: `earea`, `edist`, `eangle`
hemisphere: `upper`, `lower`

```
plot(r,'projection','earea','upper')
```



combined plots

```
plot(vector3d(1,1,1),'upper');
hold all
plot(vector3d(1,2,3),'label','B');
plot(vector3d(-1,2,1),'label','A');
hold off
```

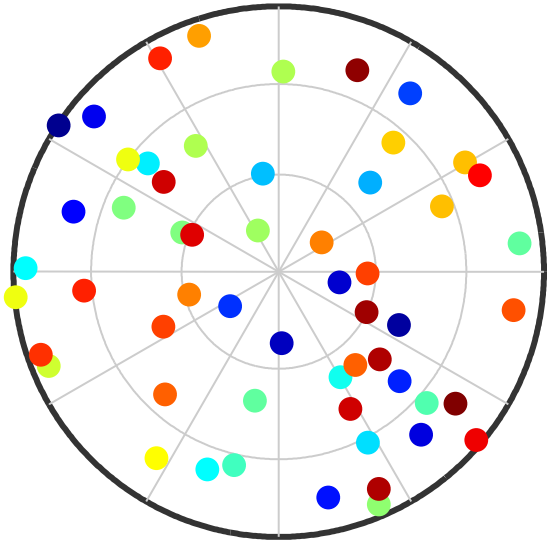
data plots

```
r = randv(100)  % 100 random points
plot(r,1:100)   % colored by number
quiver(r,orth(r)) % a vector field
```

Plotting Vectors

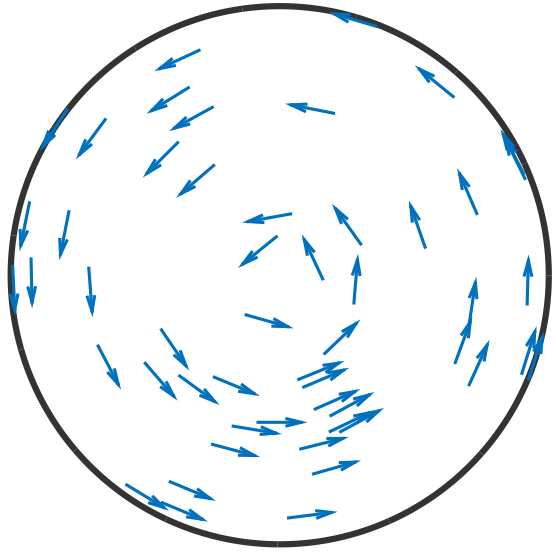
spherical projections: `earea`, `edist`, `eangle`
hemisphere: `upper`, `lower`

```
plot(r,'projection','earea','upper')
```



combined plots

```
plot(vector3d(1,1,1),'upper');
hold all
plot(vector3d(1,2,3),'label','B');
plot(vector3d(-1,2,1),'label','A');
hold off
```



data plots

```
r = randv(100)   % 100 random points
plot(r,1:100)    % colored by number
quiver(r,orth(r)) % a vector field
```

Customize Plots

General Syntax

```
plot(vector3d,<options>)
```

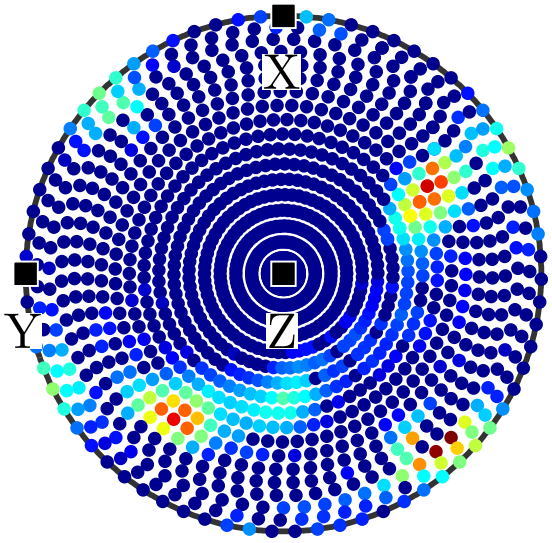
Options

```
Marker                % marker shape
MarkerSize            % marker size
MarkerFaceColor       % face color
MarkerEdgeColor       % edge color
label                 % a label text
color, background     % text colors
```

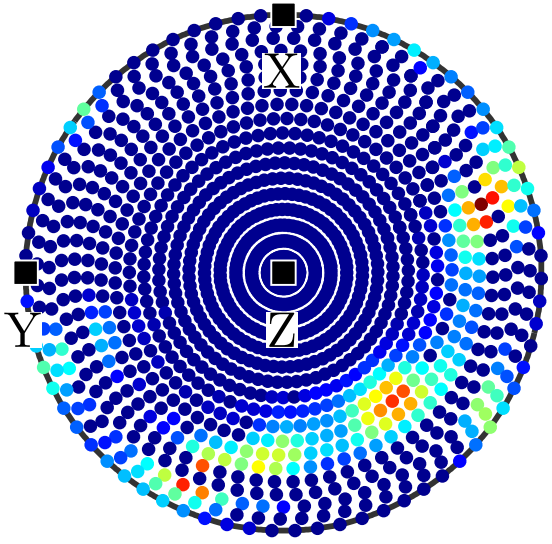
Example

```
plot([xvector,yvector,zvector],
  'Backgroundcolor','w','Marker','s',
  'MarkerEdgeColor','w','labeled',
  'MarkerFaceColor','k')
```

(02–21)



(10–10)



Customize Plots

General Syntax

```
annotate(orientation,<options>)
```

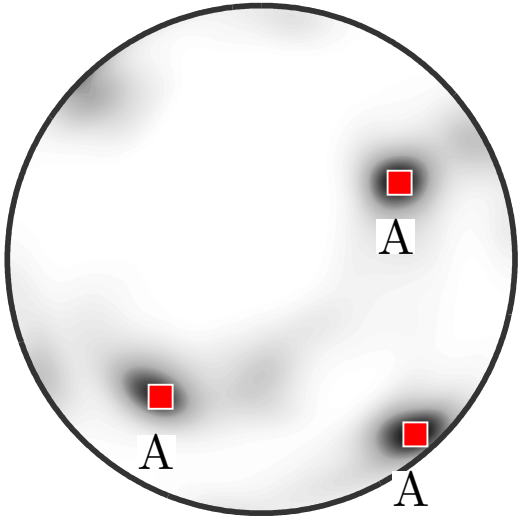
Options

```
Marker                % marker shape
MarkerSize            % marker size
MarkerFaceColor       % face color
MarkerEdgeColor       % edge color
label                 % a label text
color, background     % text colors
```

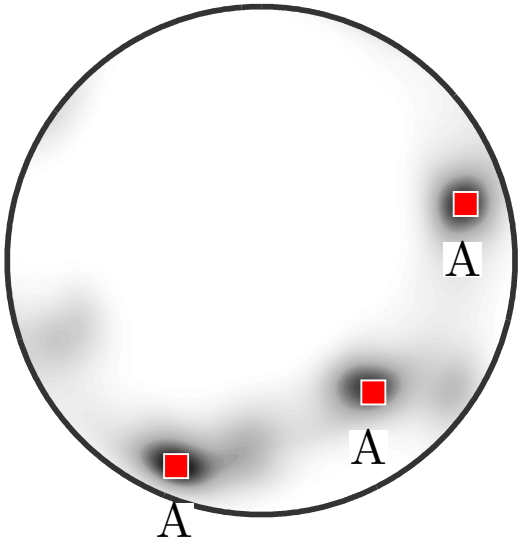
Example

```
plot(q0,'label','A',...
  'Backgroundcolor','w','Marker','s',
  'MarkerEdgeColor','w',...
  'MarkerFaceColor','r')
```

(02–21)



(10–10)



Axes

Axes are three dimensional vectors where we do not care about length and direction, e.g. plane normals.

```
r = vector3d(1,1,1,'antipodal')
```

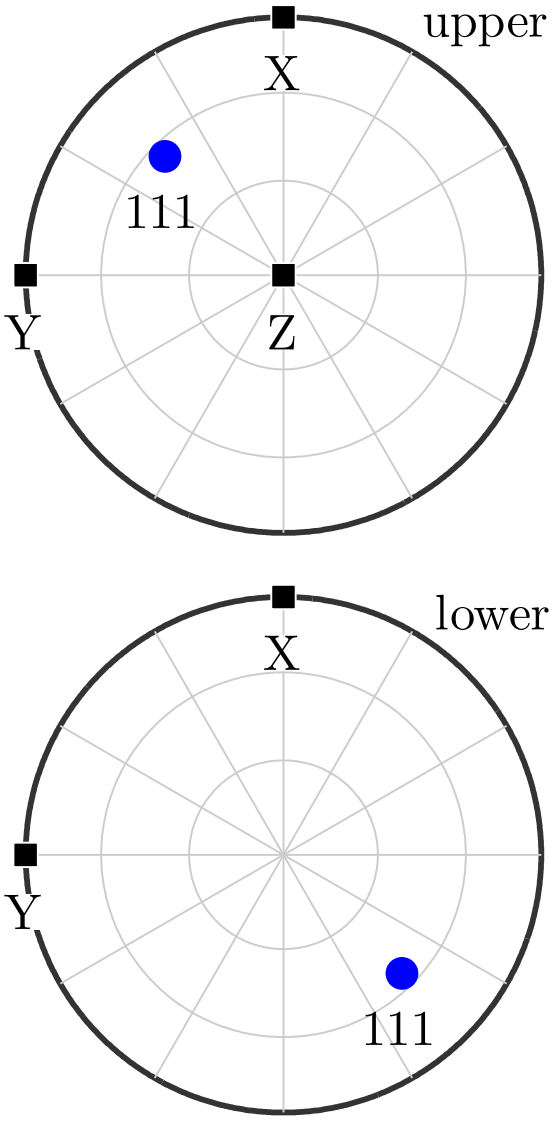
Then `r` and `-r` represent the same axis

```
eq(r, -r)
```

The angle to an axis is always less then 90°

```
angle(r,-xvector) / degree
```

The option **antipodal** in a contour plot



Axes

Axes are three dimensional vectors where we do not care about length and direction, e.g. plane normals.

```
r = vector3d(1,1,1,'antipodal')
```

Then `r` and `-r` represent the same axis

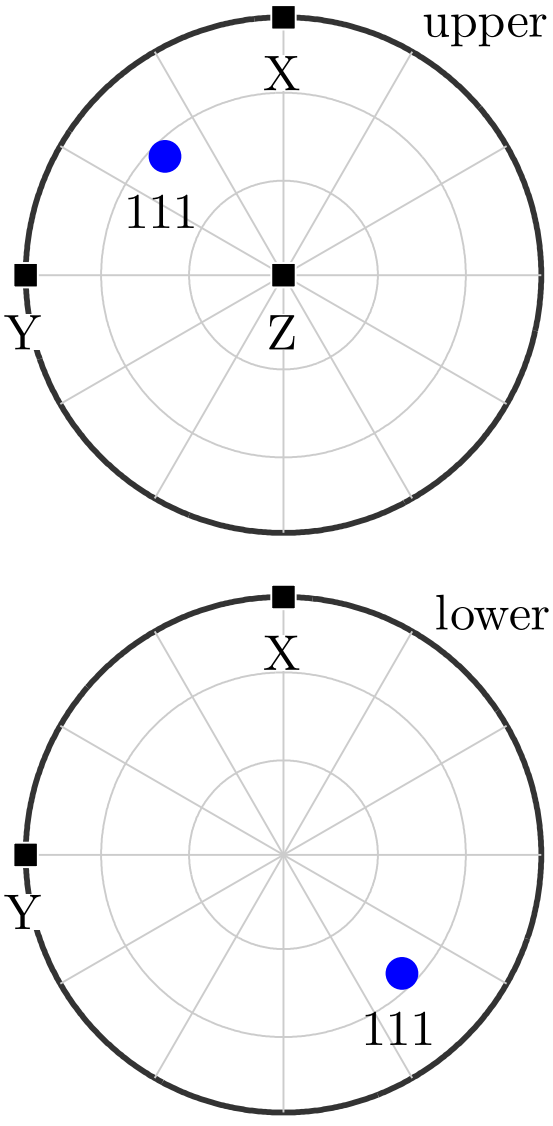
```
eq(r, -r)
```

```
1
```

The angle to an axis is always less then 90°

```
angle(r,-xvector) / degree
```

The option **antipodal** in a contour plot



Axes

Axes are three dimensional vectors where we do not care about length and direction, e.g. plane normals.

```
r = vector3d(1,1,1,'antipodal')
```

Then `r` and `-r` represent the same axis

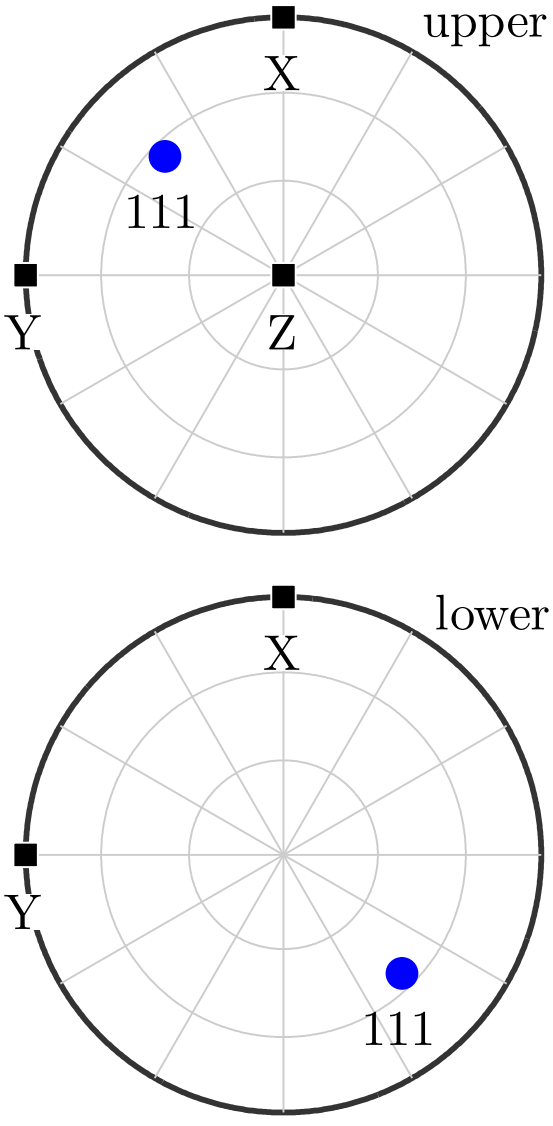
```
eq(r, -r)
```

The angle to an axis is always less then 90°

```
angle(r,-xvector) / degree
```

```
  54.7
```

The option **antipodal** in a contour plot



Axes

Axes are three dimensional vectors where we do not care about length and direction, e.g. plane normals.

```
r = vector3d(1,1,1,'antipodal')
```

Then `r` and `-r` represent the same axis

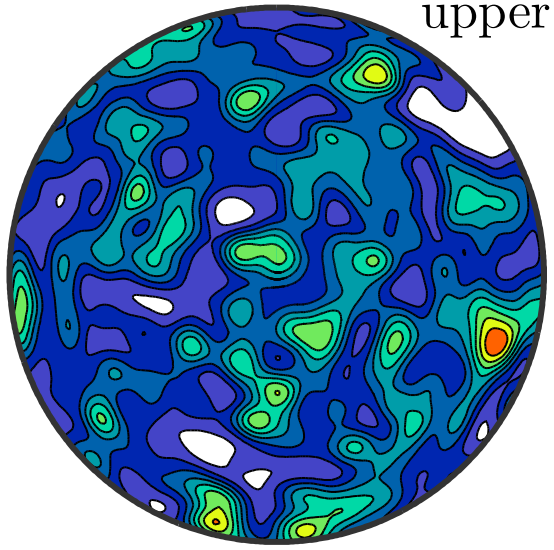
```
eq(r, -r)
```

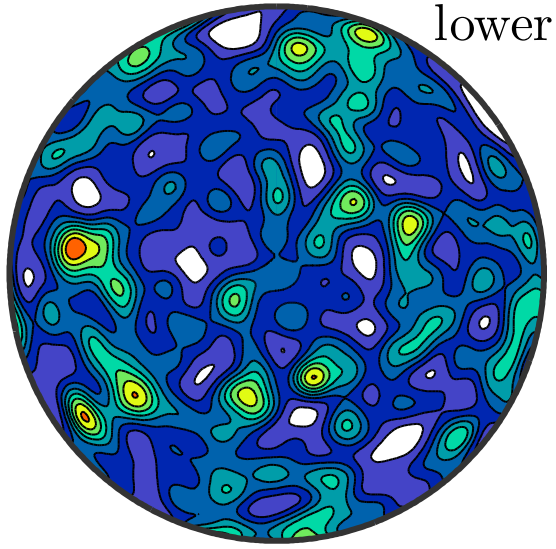
The angle to an axis is always less then 90°

```
angle(r,-xvector) / degree
```

The option **antipodal** in a contour plot

```
r = randv(1000)
plot(r,'contourf')
```





Axes

Axes are three dimensional vectors where we do not care about length and direction, e.g. plane normals.

```
r = vector3d(1,1,1,'antipodal')
```

Then `r` and `-r` represent the same axis

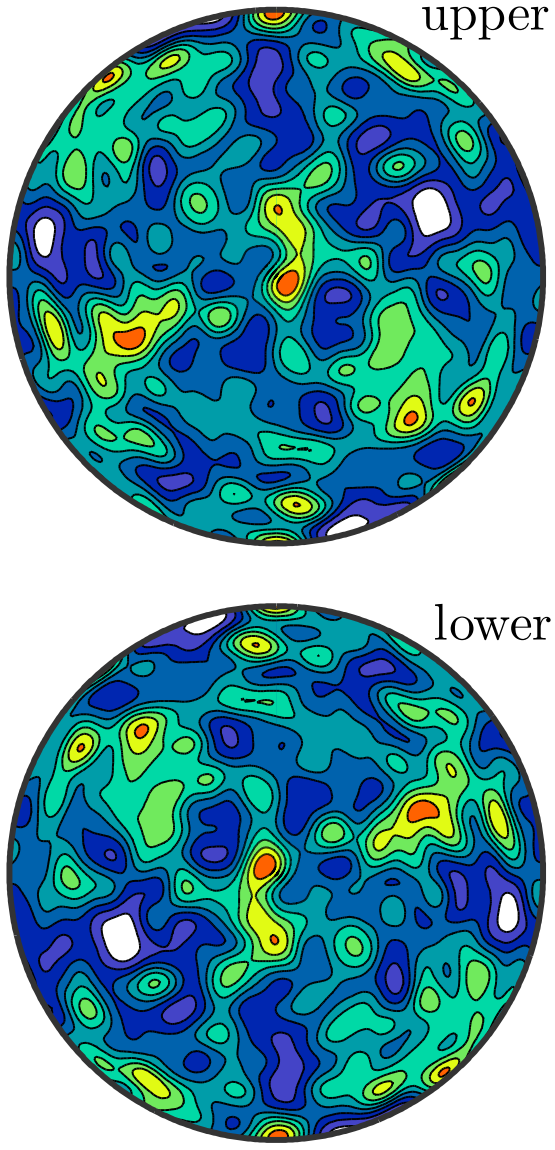
```
eq(r, -r)
```

The angle to an axis is always less then 90°

```
angle(r,-xvector) / degree
```

The option **antipodal** in a contour plot

```
r = randv(1000)
plot(r,'contourf','antipodal')
```



Rotations

A rotation is a transformation that maps a right handed coordinate system $(\vec{X}_1, \vec{Y}_1, \vec{Z}_1)$ onto another right handed coordinate system $(\vec{X}_2, \vec{Y}_2, \vec{Z}_2)$. It is given by the rotation matrix

$$\mathbf{R} = (\vec{X}_2, \vec{Y}_2, \vec{Z}_2) \cdot (\vec{X}_1, \vec{Y}_1, \vec{Z}_1)^t$$

We have $\mathbf{R}\vec{X}_1 = \vec{X}_2$, $\mathbf{R}\vec{Y}_1 = \vec{Y}_2$ and $\mathbf{R}\vec{Z}_1 = \vec{Z}_2$.

On the other hand, $\mathbf{R}$ transforms coordinates with respect to $(\vec{X}_2, \vec{Y}_2, \vec{Z}_2)$ into coordinates with respect to $(\vec{X}_1, \vec{Y}_1, \vec{Z}_1)$. I.e. for

$$\vec{r} = x_1\vec{X}_1 + y_1\vec{Y}_1 + z_1\vec{Z}_1 = x_2\vec{X}_2 + y_2\vec{Y}_2 + z_2\vec{Z}_2$$

we have

$$\mathbf{R}\begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} = \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix}$$

Rotations

A rotation is a transformation that maps a right handed coordinate system $(\vec{X}_1, \vec{Y}_1, \vec{Z}_1)$ onto another right handed coordinate system $(\vec{X}_2, \vec{Y}_2, \vec{Z}_2)$. It is given by the rotation matrix

$$\mathbf{R} = (\vec{X}_2, \vec{Y}_2, \vec{Z}_2) \cdot (\vec{X}_1, \vec{Y}_1, \vec{Z}_1)^t$$

We have $\mathbf{R}\vec{X}_1 = \vec{X}_2$, $\mathbf{R}\vec{Y}_1 = \vec{Y}_2$ and $\mathbf{R}\vec{Z}_1 = \vec{Z}_2$.

On the other hand, $\mathbf{R}$ transforms coordinates with respect to $(\vec{X}_2, \vec{Y}_2, \vec{Z}_2)$ into coordinates with respect to $(\vec{X}_1, \vec{Y}_1, \vec{Z}_1)$. I.e. for

$$\vec{r} = x_1\vec{X}_1 + y_1\vec{Y}_1 + z_1\vec{Z}_1 = x_2\vec{X}_2 + y_2\vec{Y}_2 + z_2\vec{Z}_2$$

we have

$$\mathbf{R}\begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} = \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix}$$

Euler Angles

Most commonly, rotations are given by Bunge Euler angles.

```
R = rotation('Euler',10*degree,20*degree,30*degree)
```

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

  Bunge Euler angles in degree
  phi1   Phi  phi2  Inv.
    10    20    30     0
```

```
R = rotation('Euler',...
    10*degree,20*degree,30*degree,'Roe')
```

Supported conventions are Bunge, Matthies, Roe, Kocks, Canova.

```
setMTEXpref('EulerAngleConvention','Roe')
```

Euler Angles

Most commonly, rotations are given by Bunge Euler angles.

```
R = rotation('Euler',10*degree,20*degree,30*degree)
```

```
R = rotation('Euler',...
    10*degree,20*degree,30*degree,'Roe')
```

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

  Bunge Euler angles in degree
  phi1   Phi  phi2  Inv.
   100    20   300     0
```

Supported conventions are Bunge, Matthies, Roe, Kocks, Canova.

```
setMTEXpref('EulerAngleConvention','Roe')
```

Euler Angles

Most commonly, rotations are given by Bunge Euler angles.

```
R = rotation('Euler',10*degree,20*degree,30*degree)
```

```
R = rotation('Euler',...
    10*degree,20*degree,30*degree,'Roe')
```

Supported conventions are `Bunge`, `Matthies`, `Roe`, `Kocks`, `Canova`.

```
setMTEXpref('EulerAngleConvention','Roe')
```

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

  Roe Euler angles in degree
  Psi  Theta    Phi  Inv.
   10     20     30     0
```

Other Ways to Define a Rotation

A rotation is uniquely defined by its rotation axis and its rotation angle

```
R = rotation('axis',xvector,'angle',45*degree)
```

Conversely, one can compute axis / angle from a rotation

```
R.axis, R.angle
```

Given four vectors $\vec{u}_1, \vec{u}_2, \vec{v}_1, \vec{v}_2$ there is a unique rotation **R** such that $\mathbf{R}\vec{u}_1 = \vec{v}_1$ and $\mathbf{R}\vec{u}_2 = \vec{v}_2$

```
R = rotation('map',u1,v1,u2,v2)
```

Of course one can also define a rotation by its 3×3 matrix

```
R = rotation('matrix',A)
```

or by quaternions

```
R = rotation('quaternion',a,b,c,d)
```

Other Ways to Define a Rotation

A rotation is uniquely defined by its rotation axis and its rotation angle

```
R = rotation('axis',xvector,'angle',45*degree)
```

Conversely, one can compute axis / angle from a rotation

```
R.axis, R.angle
```

Given four vectors $\vec{u}_1, \vec{u}_2, \vec{v}_1, \vec{v}_2$ there is a unique rotation $\mathbf{R}$ such that $\mathbf{R}\vec{u}_1 = \vec{v}_1$ and $\mathbf{R}\vec{u}_2 = \vec{v}_2$

```
R = rotation('map',u1,v1,u2,v2)
```

Of course one can also define a rotation by its 3×3 matrix

```
R = rotation('matrix',A)
```

or by quaternions

```
R = rotation('quaternion',a,b,c,d)
```

Other Ways to Define a Rotation

A rotation is uniquely defined by its rotation axis and its rotation angle

```
R = rotation('axis',xvector,'angle',45*degree)
```

Conversely, one can compute axis / angle from a rotation

```
R.axis, R.angle
```

Given four vectors $\vec{u}_1, \vec{u}_2, \vec{v}_1, \vec{v}_2$ there is a unique rotation $\mathbf{R}$ such that $\mathbf{R}\vec{u}_1 = \vec{v}_1$ and $\mathbf{R}\vec{u}_2 = \vec{v}_2$

```
R = rotation('map',u1,v1,u2,v2)
```

Of course one can also define a rotation by its 3×3 matrix

```
R = rotation('matrix',A)
```

or by quaternions

```
R = rotation('quaternion',a,b,c,d)
```

Other Ways to Define a Rotation

A rotation is uniquely defined by its rotation axis and its rotation angle

```
R = rotation('axis',xvector,'angle',45*degree)
```

Conversely, one can compute axis / angle from a rotation

```
R.axis, R.angle
```

Given four vectors $\vec{u}_1, \vec{u}_2, \vec{v}_1, \vec{v}_2$ there is a unique rotation **R** such that $\mathbf{R}\vec{u}_1 = \vec{v}_1$ and $\mathbf{R}\vec{u}_2 = \vec{v}_2$

```
R = rotation('map',u1,v1,u2,v2)
```

Of course one can also define a rotation by its 3×3 matrix

```
R = rotation('matrix',A)
```

or by quaternions

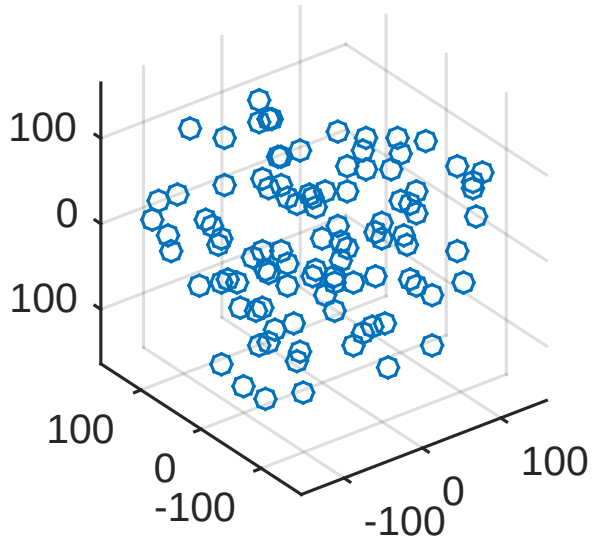
```
R = rotation('quaternion',a,b,c,d)
```

Basic Calculations

rotate a vector

```
R = rotation('axis',xvector,...
             'angle',-45*degree);
R * vector3d(0,1,1)
```

```
ans = vector3d (show methods, plot)
  size: 1 x 1
  x         y         z
  0  1.41421         0
```



the inverse rotation

```
inv(R)
```

combine rotations

```
R * inv(R)
```

plotting

Basic Calculations

rotate a vector

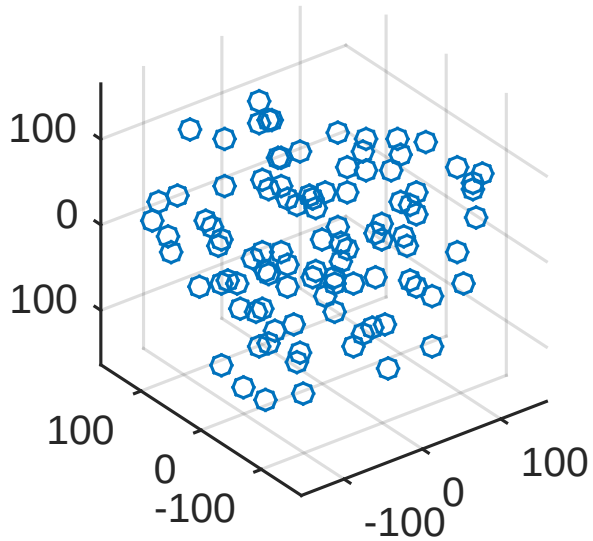
```
R = rotation('axis',xvector,...
             'angle',-45*degree);
R * vector3d(0,1,1)
```

the inverse rotation

```
inv(R)
```

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

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



combine rotations

```
R * inv(R)
```

Basic Calculations

rotate a vector

```
R = rotation('axis',xvector,...
             'angle',-45*degree);
R * vector3d(0,1,1)
```

the inverse rotation

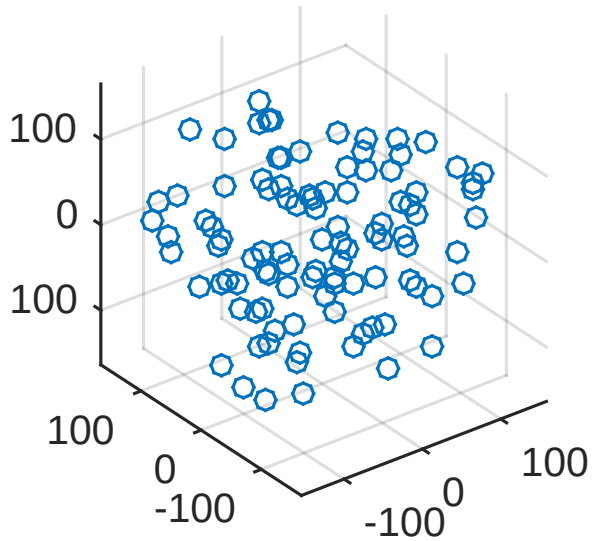
```
inv(R)
```

combine rotations

```
R * inv(R)
```

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

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



Basic Calculations

rotate a vector

```
R = rotation('axis',xvector,...
             'angle',-45*degree);
R * vector3d(0,1,1)
```

the inverse rotation

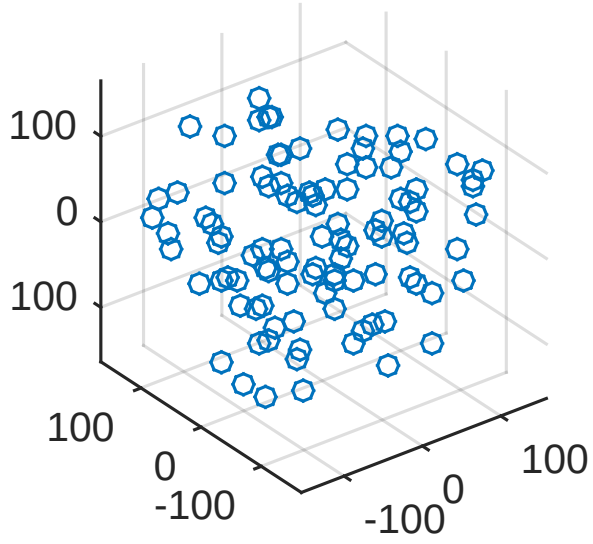
```
inv(R)
```

combine rotations

```
R * inv(R)
```

plotting

```
R = randq(100)
scatter(R) % Rodriguez space
```



Improper Rotations

An improper rotation is a rotation followed by an inversion

```
I = -rotation('Euler',10*degree,20*degree,30*degree)
```

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

  Bunge Euler angles in degree
  phi1   Phi  phi2  Inv.
    10    20    30     1
```

reflections

```
R = reflection(xvector+yvector)
```

angles between proper and improper rotations

```
angle(I,[R,-R])
```

Improper Rotations

An improper rotation is a rotation followed by an inversion

```
I = -rotation('Euler',10*degree,20*degree,30*degree)
```

reflections

```
R = reflection(xvector+yvector)
```

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

  Bunge Euler angles in degree
  phi1   Phi  phi2  Inv.
    45   180   315     1
```

angles between proper and improper rotations

```
angle(I,[R,-R])
```

Improper Rotations

An improper rotation is a rotation followed by an inversion

```
I = -rotation('Euler',10*degree,20*degree,30*degree)
```

reflections

```
R = reflection(xvector+yvector)
```

angles between proper and improper rotations

```
angle(I,[R,-R])
```

```
  168.5677  180.0000
```

check for improper rotations

```
I.isImproper
```

Improper Rotations

An improper rotation is a rotation followed by an inversion

```
I = -rotation('Euler',10*degree,20*degree,30*degree)
```

reflections

```
R = reflection(xvector+yvector)
```

angles between proper and improper rotations

```
angle(I,[R,-R])
```

check for improper rotations

```
I.isImproper
```

```
  1
```

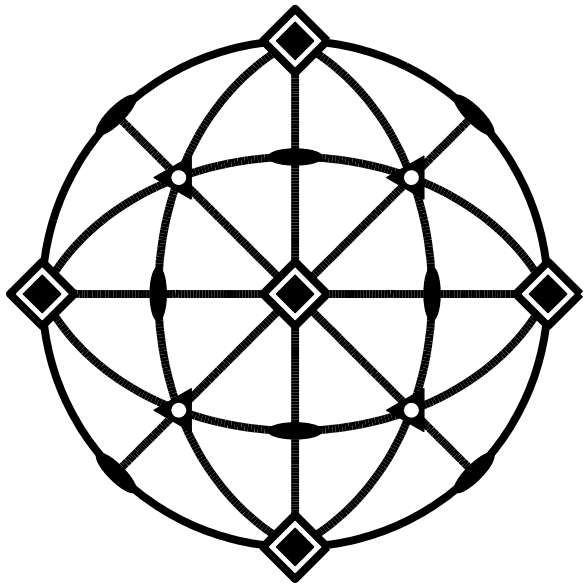
Crystal Symmetry

The point group **C** of a crystal are all rotations **R** that keep the crystal lattice invariant.

```
CS = crystalSymmetry('m-3m')
```

```
CS = crystalSymmetry (show methods, plot)

  symmetry: m-3m
  a, b, c : 1, 1, 1
```



extract the rotations of a point group

```
rotation(crystalSymmetry('222'))
```

import data from crystal information files

```
CS = loadCIF('Quarz.cif')
```

```
CS = loadPHL('minerals.phl')
```

switch to Laue / purely rotational group

Crystal Symmetry

The point group **C** of a crystal are all rotations **R** that keep the crystal lattice invariant.

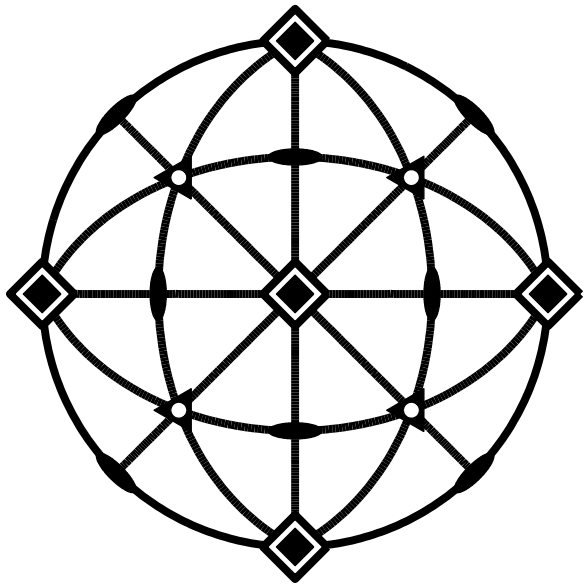
```
CS = crystalSymmetry('m-3m')
```

extract the rotations of a point group

```
rotation(crystalSymmetry('222'))
```

```
ans = rotation (show methods, plot)
size: 4 x 1

Bunge Euler angles in degree
phi1   Phi  phi2  Inv.
   0     0     0     0
 180     0     0     0
  45   180    45     0
  45   180   225     0
```



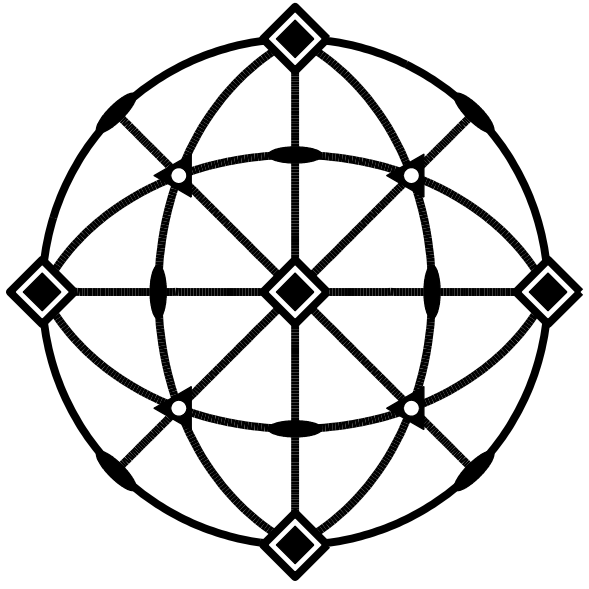
import data from crystal information files

```
CS = loadCIF('Quarz.cif')
```

Crystal Symmetry

The point group **C** of a crystal are all rotations **R** that keep the crystal lattice invariant.

```
CS = crystalSymmetry('m-3m')
```



extract the rotations of a point group

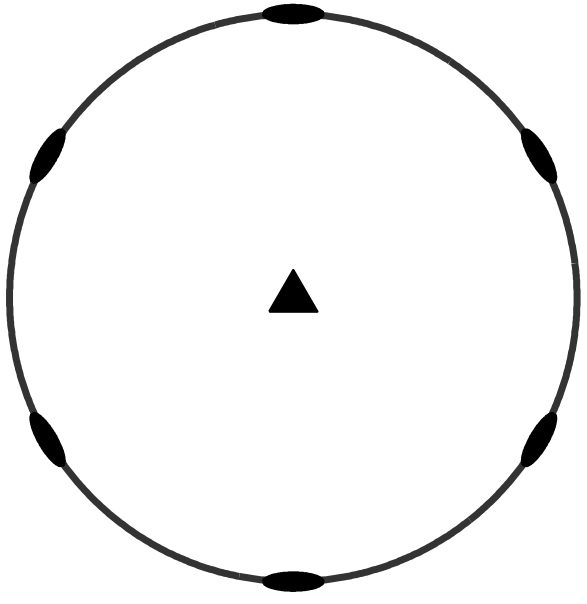
```
rotation(crystalSymmetry('222'))
```

import data from crystal information files

```
CS = loadCIF('Quarz.cif')
```

```
CS = crystalSymmetry (show methods, plot)

  mineral              : Quartz
  symmetry             : P 32 2 1 (321)
  a, b, c              : 4.9, 4.9, 5.4
  alpha, beta, gamma   : 90  , 90  , 120
  reference frame      : X||a*, Y||b, Z||c*
```

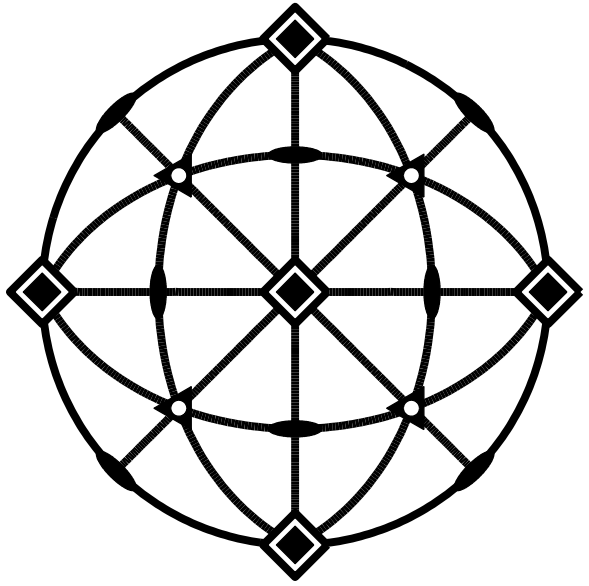


```
CS = loadPHL('minerals.phl')
```

Crystal Symmetry

The point group **C** of a crystal are all rotations **R** that keep the crystal lattice invariant.

```
CS = crystalSymmetry('m-3m')
```



extract the rotations of a point group

```
rotation(crystalSymmetry('222'))
```

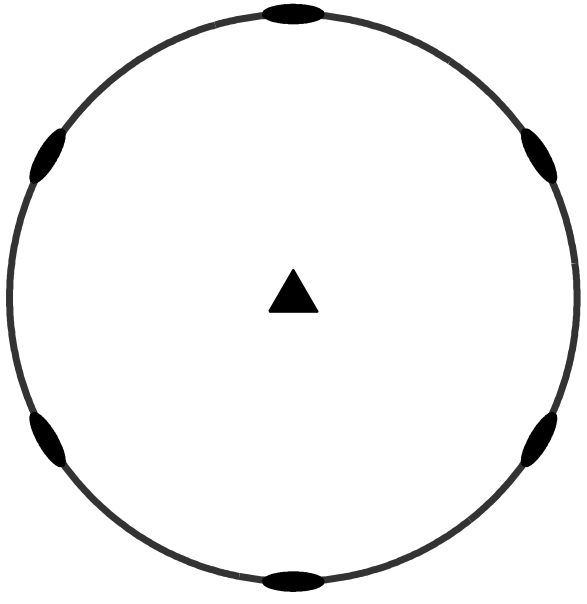
import data from crystal information files

```
CS = loadCIF('Quarz.cif')
```

```
CS = loadPHL('minerals.phl')
```

```
CS{1} = crystalSymmetry (show methods, plot)

  mineral : Magnetite
  density : 5.054
  symmetry: m-3m
  a, b, c : 8.4, 8.4, 8.4
```



Crystal Symmetry

The point group **C** of a crystal are all rotations **R** that keep the crystal lattice invariant.

```
CS = crystalSymmetry('m-3m')
```

extract the rotations of a point group

```
rotation(crystalSymmetry('222'))
```

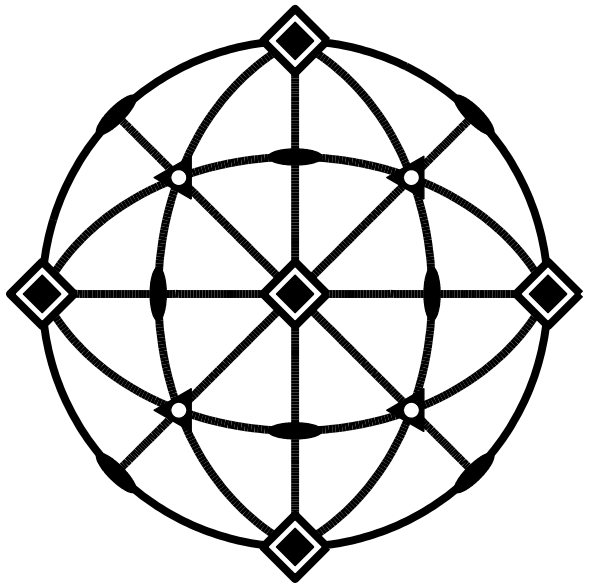
import data from crystal information files

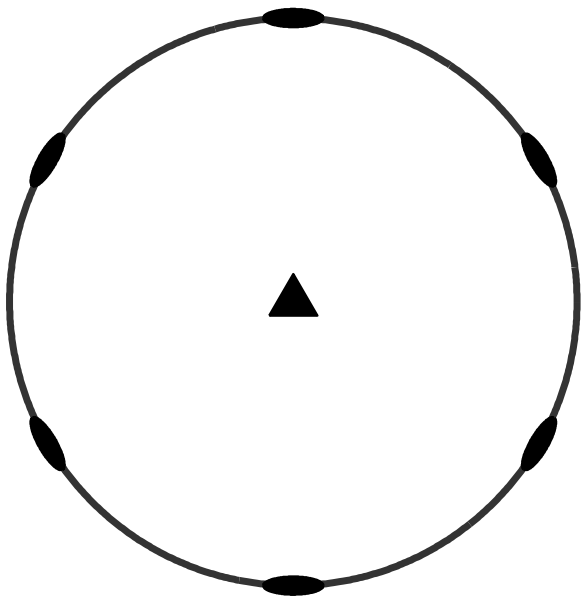
```
CS = loadCIF('Quarz.cif')
```

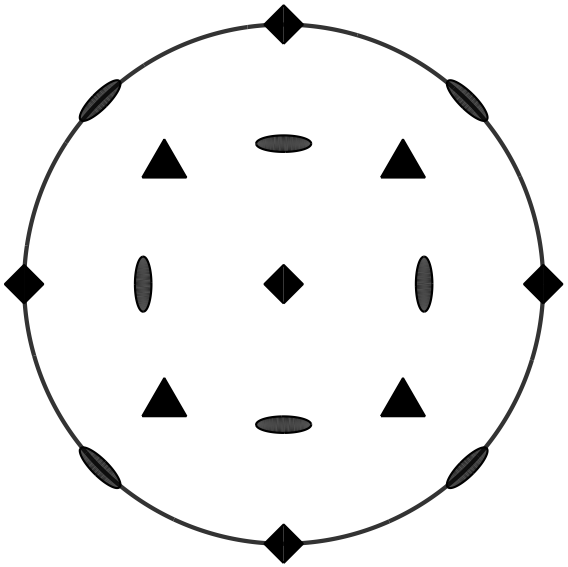
```
CS = loadPHL('minerals.phl')
```

switch to Laue / purely rotational group

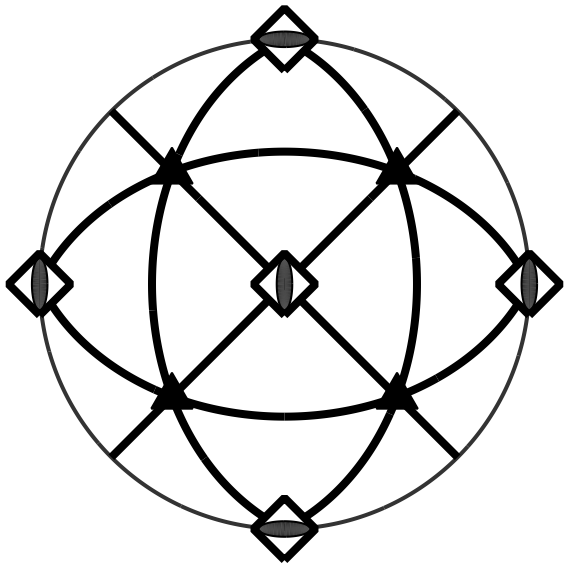
```
CS.Laue
CS.properGroup
```



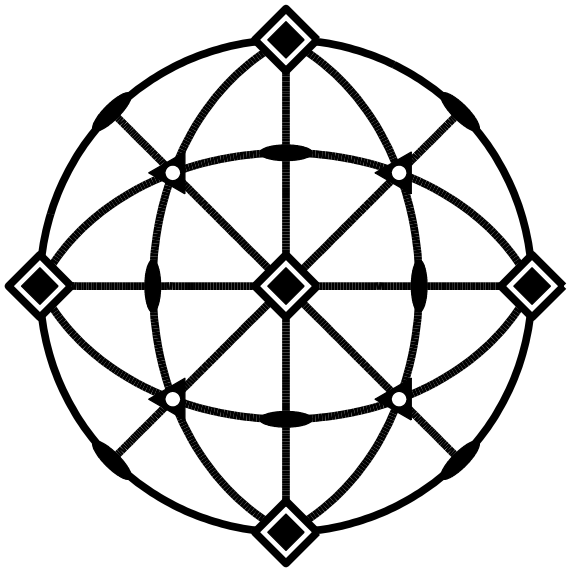




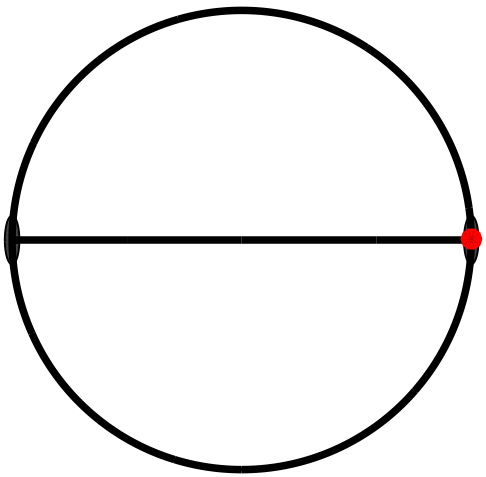
432



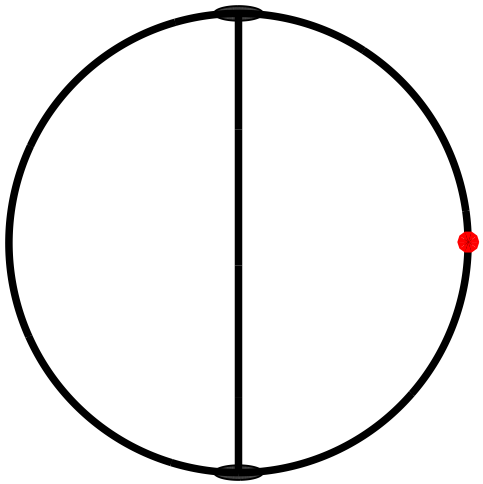
43m



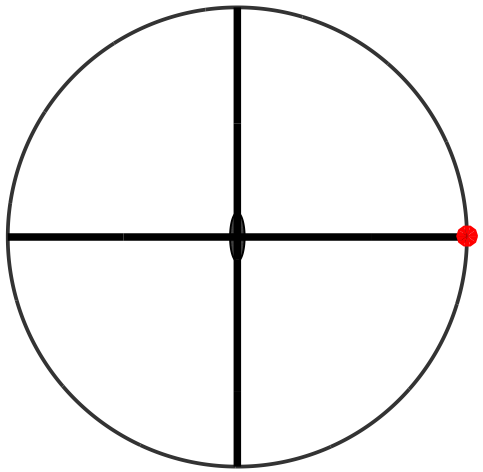
m-3m



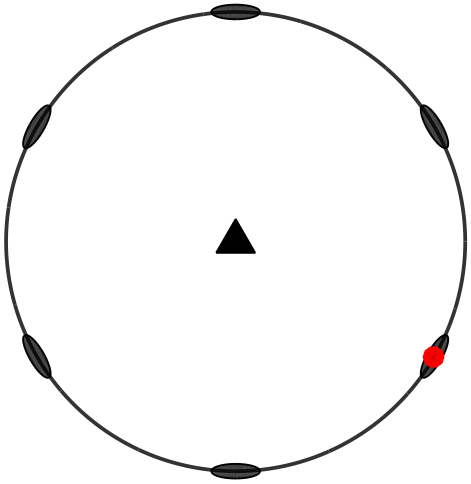
2mm



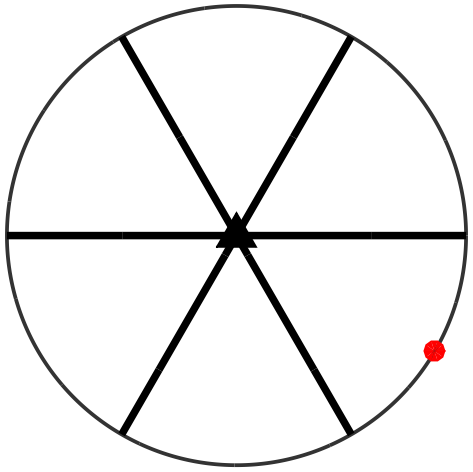
m2m



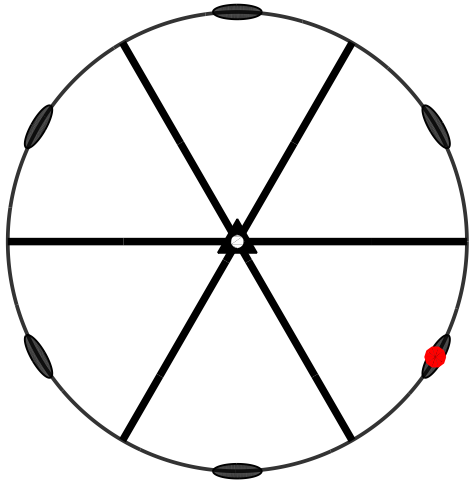
mm2



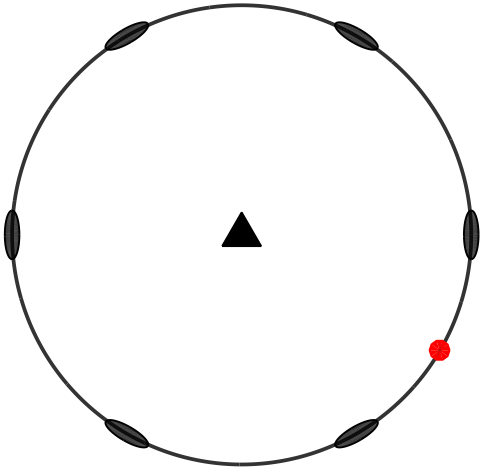
321



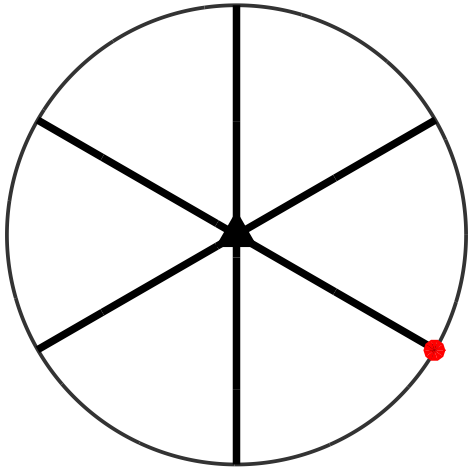
3m1



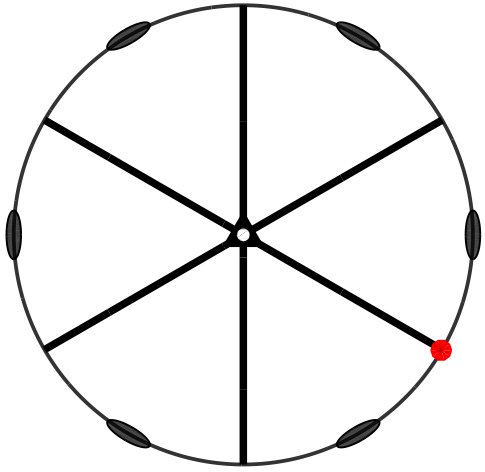
-3m1



312



31m



-31m

Unit Cell, Reciprocal and Orthogonal Coordinate System

The unit cell of a crystal is specified by the length of its three edges $\vec{a}, \vec{b}, \vec{c}$ and by angles α, β, γ they enclose.

```
C = crystalSymmetry('1',[a b c],[alpha beta gamma])
```

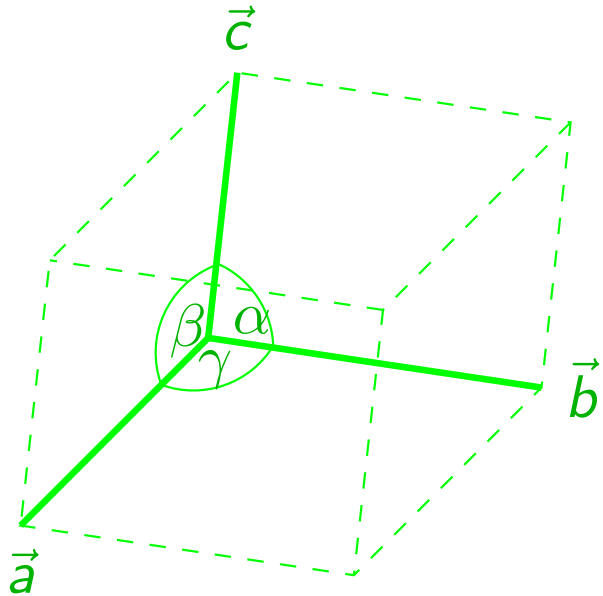
The axes of the reciprocal lattice are defined orthogonal to $\vec{a}, \vec{b}, \vec{c}$, i.e.

$$\vec{a}^* = \frac{\vec{b} \times \vec{c}}{V}, \; \vec{b}^* = \frac{\vec{c} \times \vec{a}}{V}, \; \vec{c}^* = \frac{\vec{a} \times \vec{b}}{V}$$

with $V = \vec{a} \cdot (\vec{b} \times \vec{c})$ volume of the unit cell

We will need also an orthogonal coordinate system $(\vec{x}, \vec{y}, \vec{z})$ fixed to the crystal.

There are different conventions.



Unit Cell, Reciprocal and Orthogonal Coordinate System

The unit cell of a crystal is specified by the length of its three edges $\vec{a}, \vec{b}, \vec{c}$ and by angles α, β, γ they enclose.

```
C = crystalSymmetry('1',[a b c],[alpha beta gamma])
```

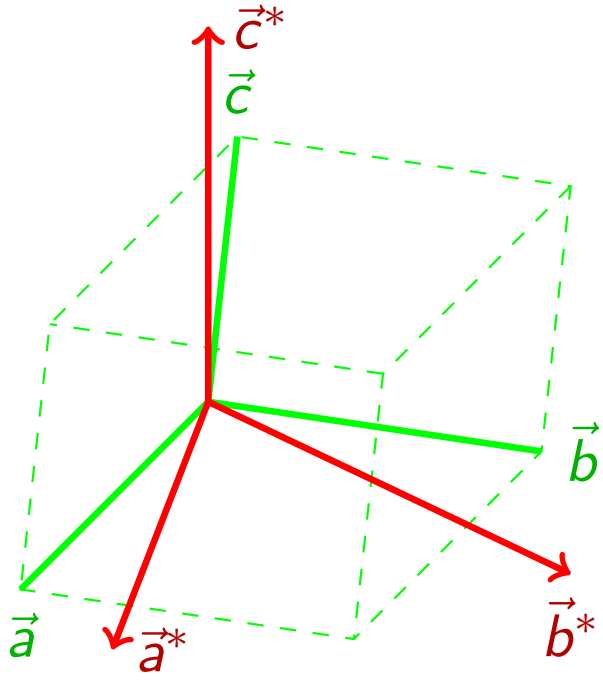
The axes of the reciprocal lattice are defined orthogonal to $\vec{a}, \vec{b}, \vec{c}$, i.e.

$$\vec{a}^* = \frac{\vec{b} \times \vec{c}}{V}, \quad \vec{b}^* = \frac{\vec{c} \times \vec{a}}{V}, \quad \vec{c}^* = \frac{\vec{a} \times \vec{b}}{V}$$

with $V = \vec{a} \cdot (\vec{b} \times \vec{c})$ volume of the unit cell

We will need also an orthogonal coordinate system $(\vec{x}, \vec{y}, \vec{z})$ fixed to the crystal.

There are different conventions.



Unit Cell, Reciprocal and Orthogonal Coordinate System

The unit cell of a crystal is specified by the length of its three edges $\vec{a}, \vec{b}, \vec{c}$ and by angles α, β, γ they enclose.

```
C = crystalSymmetry('1',[a b c],[alpha beta gamma])
```

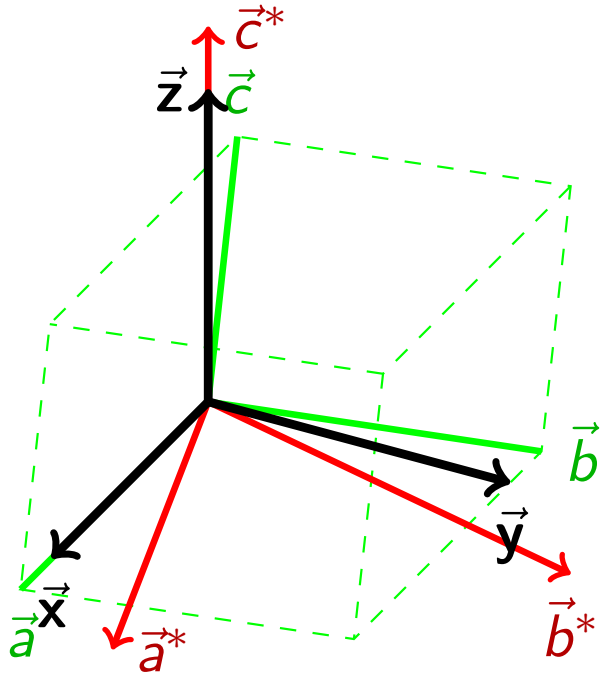
The axes of the reciprocal lattice are defined orthogonal to $\vec{a}, \vec{b}, \vec{c}$, i.e.

$$\vec{a}^* = \frac{\vec{b} \times \vec{c}}{V}, \quad \vec{b}^* = \frac{\vec{c} \times \vec{a}}{V}, \quad \vec{c}^* = \frac{\vec{a} \times \vec{b}}{V}$$

with $V = \vec{a} \cdot (\vec{b} \times \vec{c})$ volume of the unit cell

We will need also an orthogonal coordinate system $(\vec{x}, \vec{y}, \vec{z})$ fixed to the crystal.

There are different conventions.




```
C = crystalSymmetry('321',[a b c],'X||a','Z||c*')
```

Unit Cell, Reciprocal and Orthogonal Coordinate System

The unit cell of a crystal is specified by the length of its three edges $\vec{a}, \vec{b}, \vec{c}$ and by angles α, β, γ they enclose.

```
C = crystalSymmetry('1',[a b c],[alpha beta gamma])
```

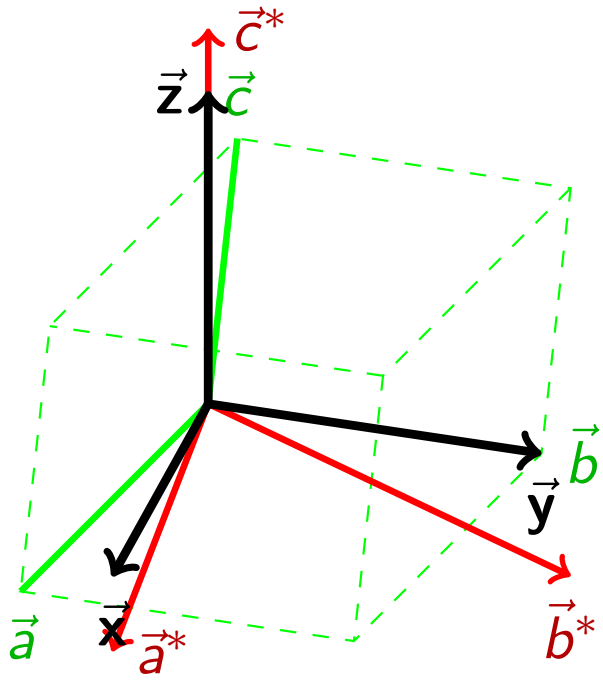
The axes of the reciprocal lattice are defined orthogonal to $\vec{a}, \vec{b}, \vec{c}$, i.e.

$$\vec{a}^* = \frac{\vec{b} \times \vec{c}}{V}, \quad \vec{b}^* = \frac{\vec{c} \times \vec{a}}{V}, \quad \vec{c}^* = \frac{\vec{a} \times \vec{b}}{V}$$

with $V = \vec{a} \cdot (\vec{b} \times \vec{c})$ volume of the unit cell

We will need also an orthogonal coordinate system $(\vec{x}, \vec{y}, \vec{z})$ fixed to the crystal.

There are different conventions.




```
C = crystalSymmetry('321',[a b c],'X||b','Z||c*')
```

The alignment of $\vec{x}$, $\vec{y}$, $\vec{z}$ is important as the Euler angles refer to them.

Miller Indices - Crystal Fixed Directions

A direction with respect to the crystal coordinate system **C**

$$\vec{m} = u \cdot \vec{a} + v \cdot \vec{b} + w \cdot \vec{c}.$$

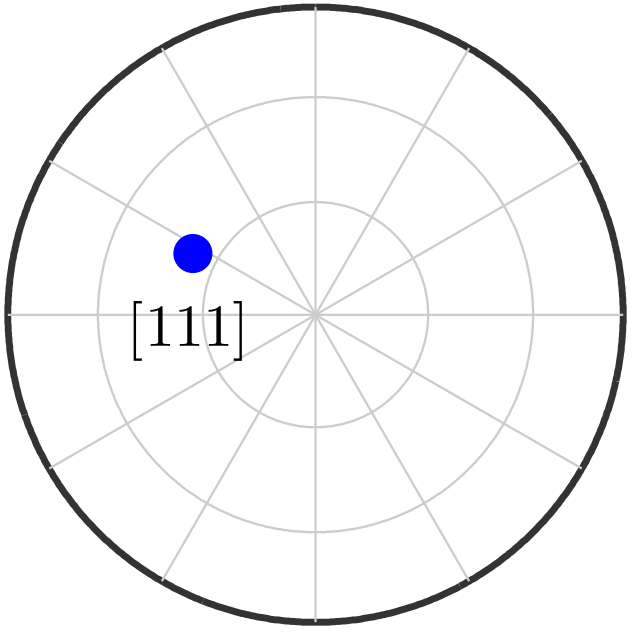
Miller Indices - Crystal Fixed Directions

A direction with respect to the crystal coordinate system **C**

$$\vec{m} = u \cdot \vec{a} + v \cdot \vec{b} + w \cdot \vec{c}.$$

```
CS = symmetry('mmm',[1 2 3])
m = Miller(1,1,1,CS,'uvw')
```

```
m = Miller (show methods, plot)
  size: 1 x 1
  symmetry: mmm
  u 1
  v 1
  w 1
```




```
plot(m,'labeled')
```

Miller Indices - Crystal Fixed Directions

A direction with respect to the crystal coordinate system **C**

$$\vec{m} = u \cdot \vec{a} + v \cdot \vec{b} + w \cdot \vec{c}.$$

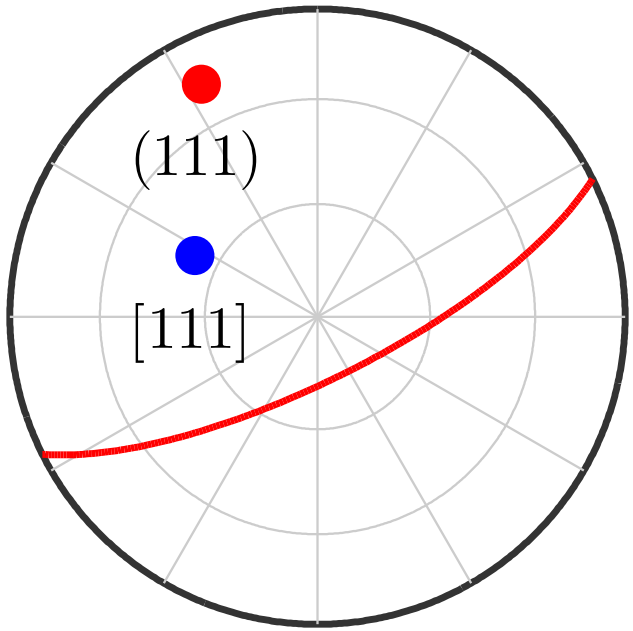
```
CS = symmetry('mmm',[1 2 3])
m = Miller(1,1,1,CS,'uvw')
```

A direction in reciprocal coordinates

$$\vec{r} = h \cdot \vec{a}^* + k \cdot \vec{b}^* + \ell \cdot \vec{c}^*.$$

```
m = Miller(1,1,1,CS,'hkl')
```

```
m = Miller (show methods, plot)
  size: 1 x 1
  symmetry: mmm
  h 1
  k 1
  l 1
```




```
plot(m,'labeled')
```

```
hold all
plot(m,'labeled')
```

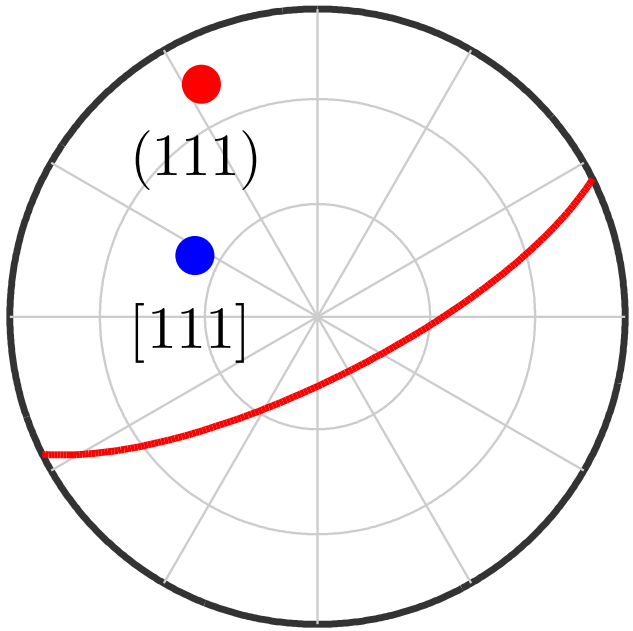
```
plot(m,'plane')
```

Miller Indices - Crystal Fixed Directions

A direction with respect to the crystal coordinate system **C**

$$\vec{m} = u \cdot \vec{a} + v \cdot \vec{b} + w \cdot \vec{c}.$$

```
CS = symmetry('mmm',[1 2 3])
m = Miller(1,1,1,CS,'uvw')
```



A direction in reciprocal coordinates

$$\vec{r} = h \cdot \vec{a}^* + k \cdot \vec{b}^* + \ell \cdot \vec{c}^*.$$

```
m = Miller(1,1,1,CS,'hkl')
```

A direction in the orthogonal coordinate system

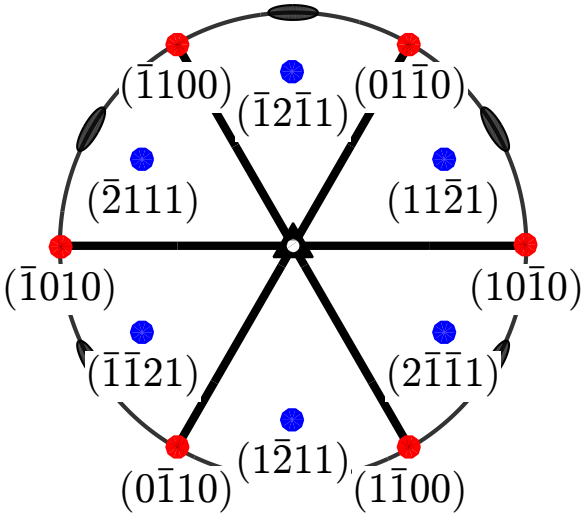
```
m = Miller(xvector,CS)
```

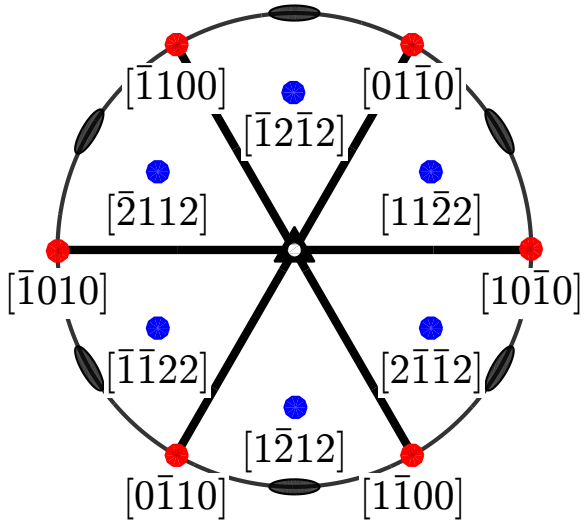
```
plot(m,'labeled')
```

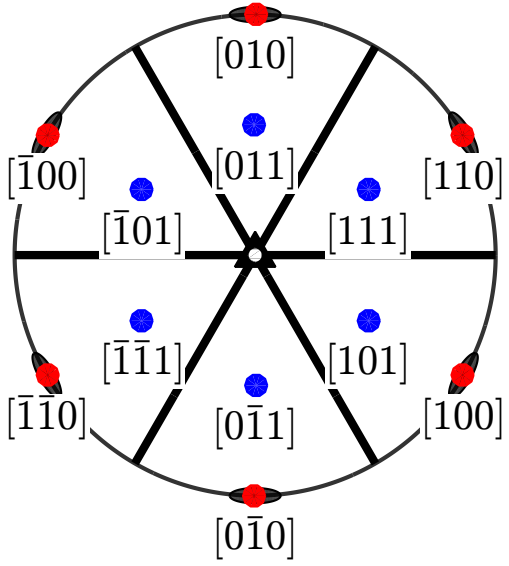
```
hold all
plot(m,'labeled')
```

```
plot(m,'plane')
```

Trigonal and Hexagonal Symmetries








```
m = [Miller(1,0,-1,0,cs,'hkil'),...
     Miller(1,1,-2,1,cs,'hkil')]
m = [Miller(1,0,-1,0,cs,'UVTW'),...
     Miller(1,1,-2,2,cs,'UVTW')]
m = [Miller(1,0,0,cs,'uvw'),Miller(1,0,1,cs,'uvw')]
```

Plot all symmetrically equivalent directions

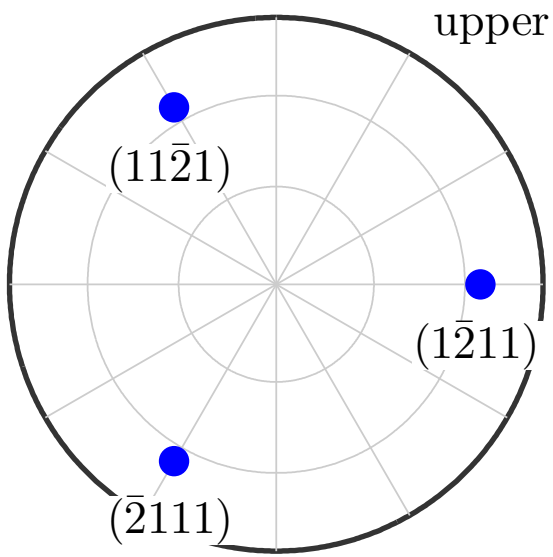
```
plot(m,'symmetrised','labeled')
```

Calculating with Crystal Directions

Find all symmetrically equivalent directions

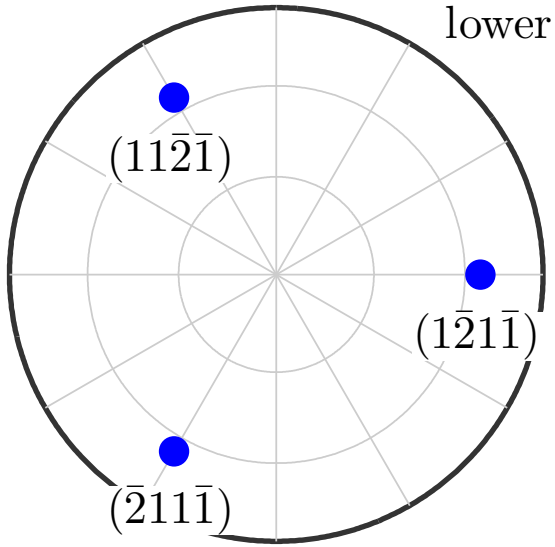
```
CS = loadCIF('quartz')
m = Miller(1,1,-2,1,CS,'hkil')
symmetrise(m)
```

```
ans = Miller (show methods, plot)
  size: 6 x 1
  mineral: Quartz (P 32 2 1, X||a*, Y||b, Z||c*)
  h   1 -2   1   1 -2   1
  k   1   1 -2 -2   1   1
  i -2   1   1   1   1 -2
  l   1   1   1 -1 -1 -1
```



Change from reciprocal coordinate system to $\vec{a}$, $\vec{b}$, $\vec{c}$

```
m.dispStyle = 'UVTW'
round(m)
```

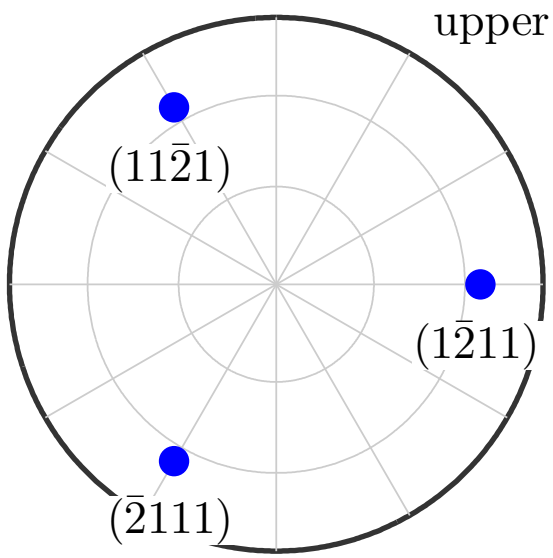


Access the coordinates and properties

Calculating with Crystal Directions

Find all symmetrically equivalent directions

```
CS = loadCIF('quartz')
m = Miller(1,1,-2,1,CS,'hkil')
symmetrise(m)
```

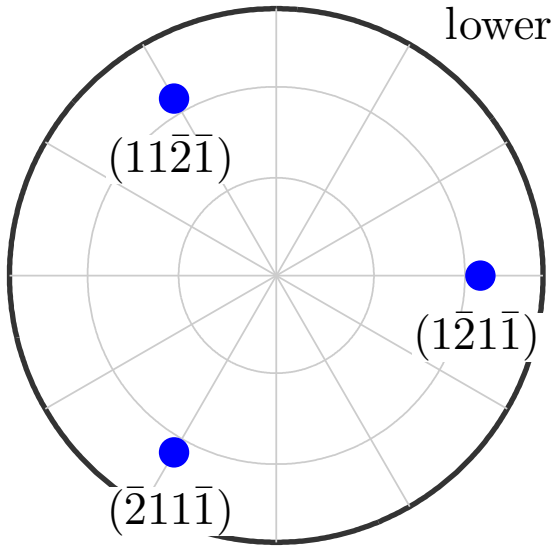


Change from reciprocal coordinate system to $\vec{a}$, $\vec{b}$, $\vec{c}$

```
m.dispStyle = 'UVTW'
```

```
m = Miller (show methods, plot)
  size: 6 x 1
  mineral: Quartz (P 32 2 1, X||a*, Y||b, Z||c*)
  U   0.0828
  V   0.0828
  T  -0.1655
  W   0.1027
```

```
round(m)
```

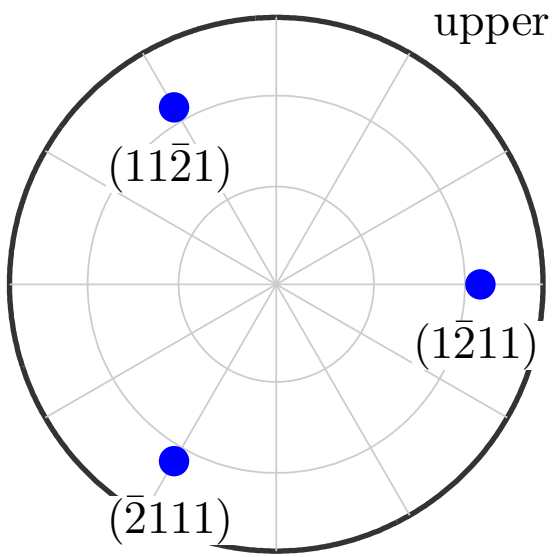


Access the coordinates and properties

Calculating with Crystal Directions

Find all symmetrically equivalent directions

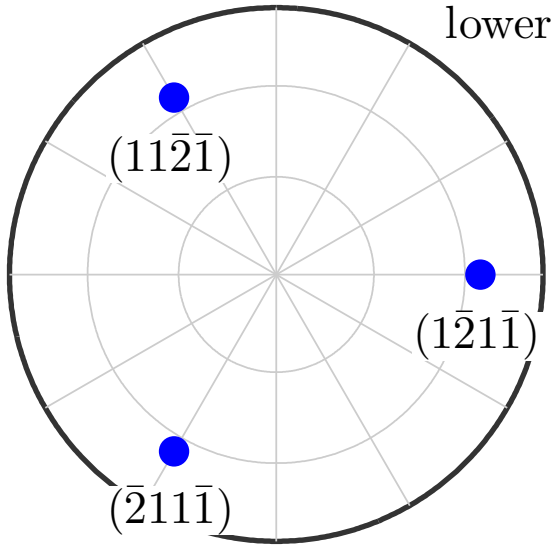
```
CS = loadCIF('quartz')
m = Miller(1,1,-2,1,CS,'hkil')
symmetrise(m)
```



Change from reciprocal coordinate system to $\vec{a}$, $\vec{b}$, $\vec{c}$

```
m.dispStyle = 'UVTW'
```

```
round(m)
```



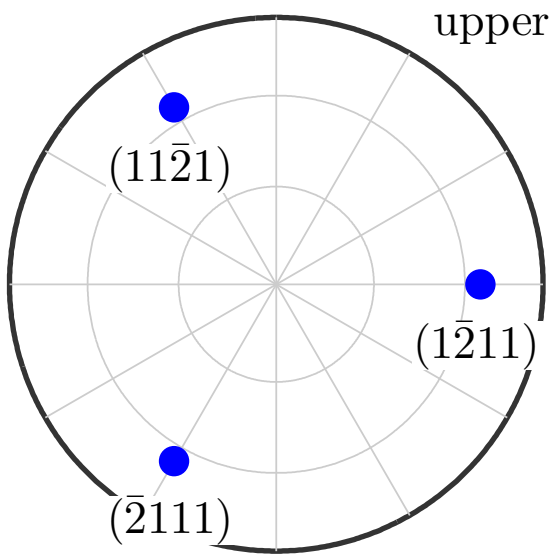

```
ans = Miller (show methods, plot)
  size: 6 x 1
  mineral: Quartz (P 32 2 1, X||a*, Y||b, Z||c*)
  U  4
  V  4
  T -8
  W  5
```

Access the coordinates and properties

Calculating with Crystal Directions

Find all symmetrically equivalent directions

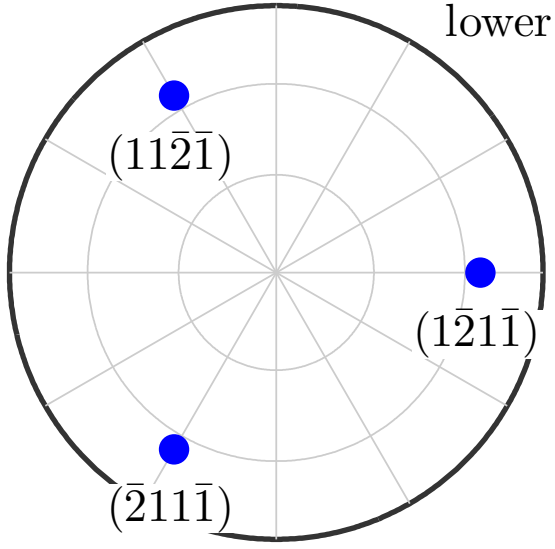
```
CS = loadCIF('quartz')
m = Miller(1,1,-2,1,CS,'hkil')
symmetrise(m)
```



Change from reciprocal coordinate system to $\vec{a}$, $\vec{b}$, $\vec{c}$

```
m.dispStyle = 'UVTW'
```

```
round(m)
```



Access the coordinates and properties

```
m.U, m.hkl, m.uvw, m.UVTW
m.dspacing % d-spacing of planes
```

angle modulo symmetry

```
angle(m1,m2) / degree
```

Calculating with Crystal Directions

Find all symmetrically equivalent directions

```
CS = loadCIF('quartz')
m = Miller(1,1,-2,1,CS,'hkil')
symmetrise(m)
```

Change from reciprocal coordinate system to $\vec{a}$, $\vec{b}$, $\vec{c}$

```
m.dispStyle = 'UVTW'
```

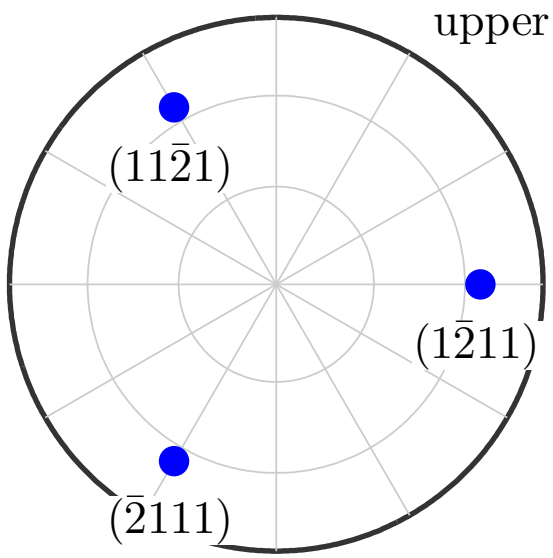
```
round(m)
```

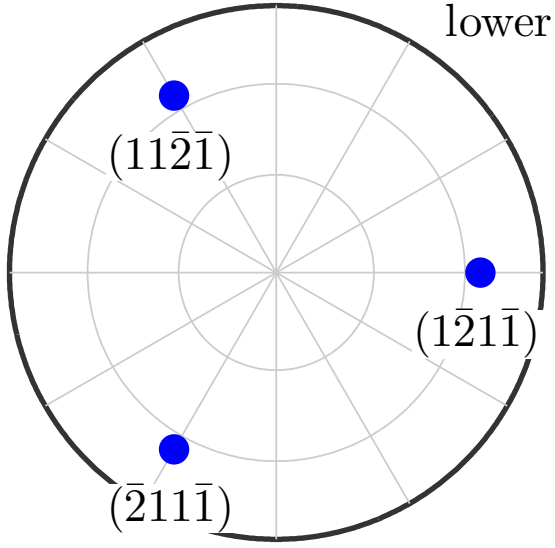
Access the coordinates and properties

```
m.U, m.hkl, m.uvw, m.UVTW
m.dspacing % d-spacing of planes
```

angle modulo symmetry

```
angle(m1,m2) / degree
```





Crystal Orientations

Let a vector $\vec{v}$ be given by specimen coordinates $(r_1, r_2, r_3)^t$ and crystal coordinates $(h_1, h_2, h_3)^t$, i.e.,

$$\vec{v} = r_1\vec{X} + r_2\vec{Y} + r_3\vec{Y} = h_1\vec{x} + h_2\vec{y} + h_3\vec{z}.$$

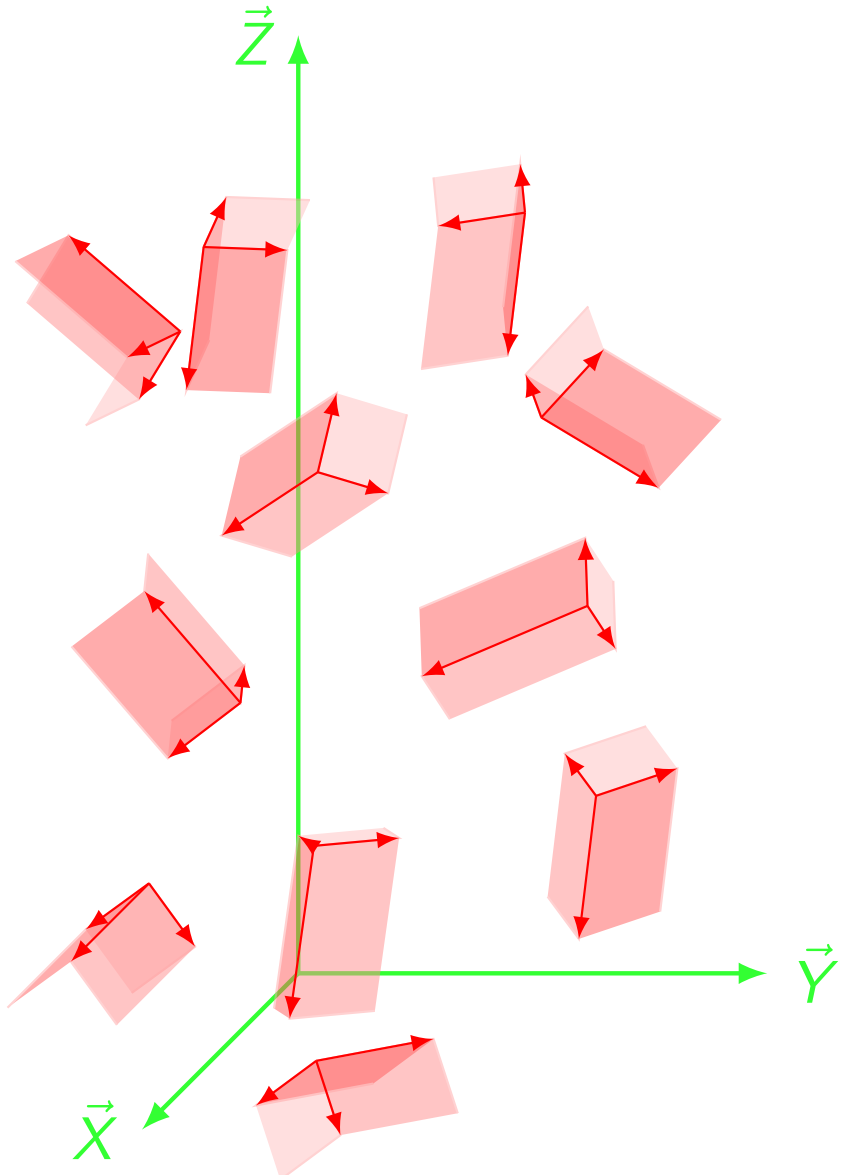
The coordinate transform $\mathbf{O}$ with

$$(r_1, r_2, r_3)^t = \mathbf{O}\,(h_1, h_2, h_3)^t$$

is called crystal orientation.

The orientation $\mathbf{O}$ maps the specimen coordinate system $\vec{X}, \vec{Y}, \vec{Z}$ onto the crystal coordinate systems $\vec{x}, \vec{y}, \vec{z}$.

The orientation $\mathbf{O}$ is well defined only up the crystal symmetry.



Crystal Orientations

Let a vector $\vec{v}$ be given by specimen coordinates $(r_1, r_2, r_3)^t$ and crystal coordinates $(h_1, h_2, h_3)^t$, i.e.,

$$\vec{v} = r_1\vec{X} + r_2\vec{Y} + r_3\vec{Y} = h_1\vec{x} + h_2\vec{y} + h_3\vec{z}.$$

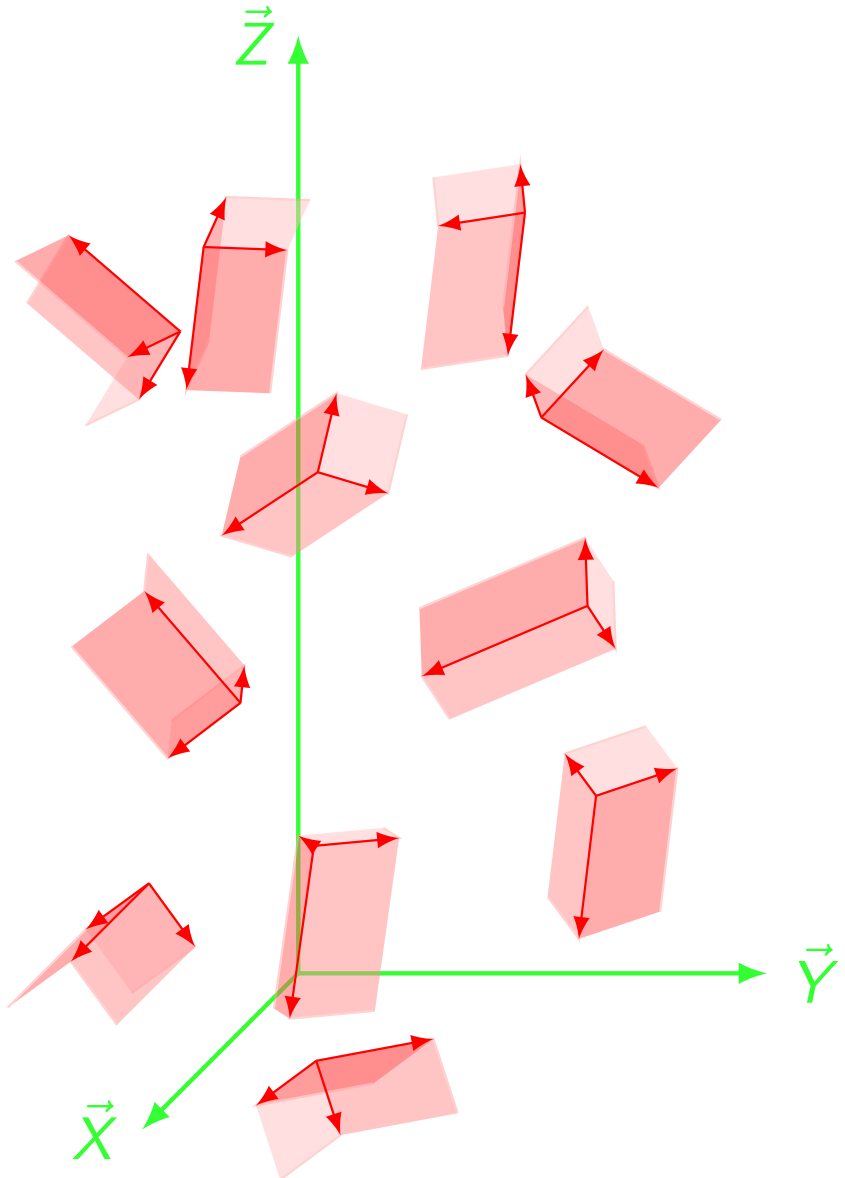
The coordinate transform $\mathbf{O}$ with

$$(r_1, r_2, r_3)^t = \mathbf{O}\,(h_1, h_2, h_3)^t$$

is called crystal orientation.

The orientation $\mathbf{O}$ maps the specimen coordinate system $\vec{X}, \vec{Y}, \vec{Z}$ onto the crystal coordinate systems $\vec{x}, \vec{y}, \vec{z}$.

The orientation $\mathbf{O}$ is well defined only up the crystal symmetry.



Crystal Orientations

Let a vector $\vec{v}$ be given by specimen coordinates $(r_1, r_2, r_3)^t$ and crystal coordinates $(h_1, h_2, h_3)^t$, i.e.,

$$\vec{v} = r_1\vec{X} + r_2\vec{Y} + r_3\vec{Y} = h_1\vec{x} + h_2\vec{y} + h_3\vec{z}.$$

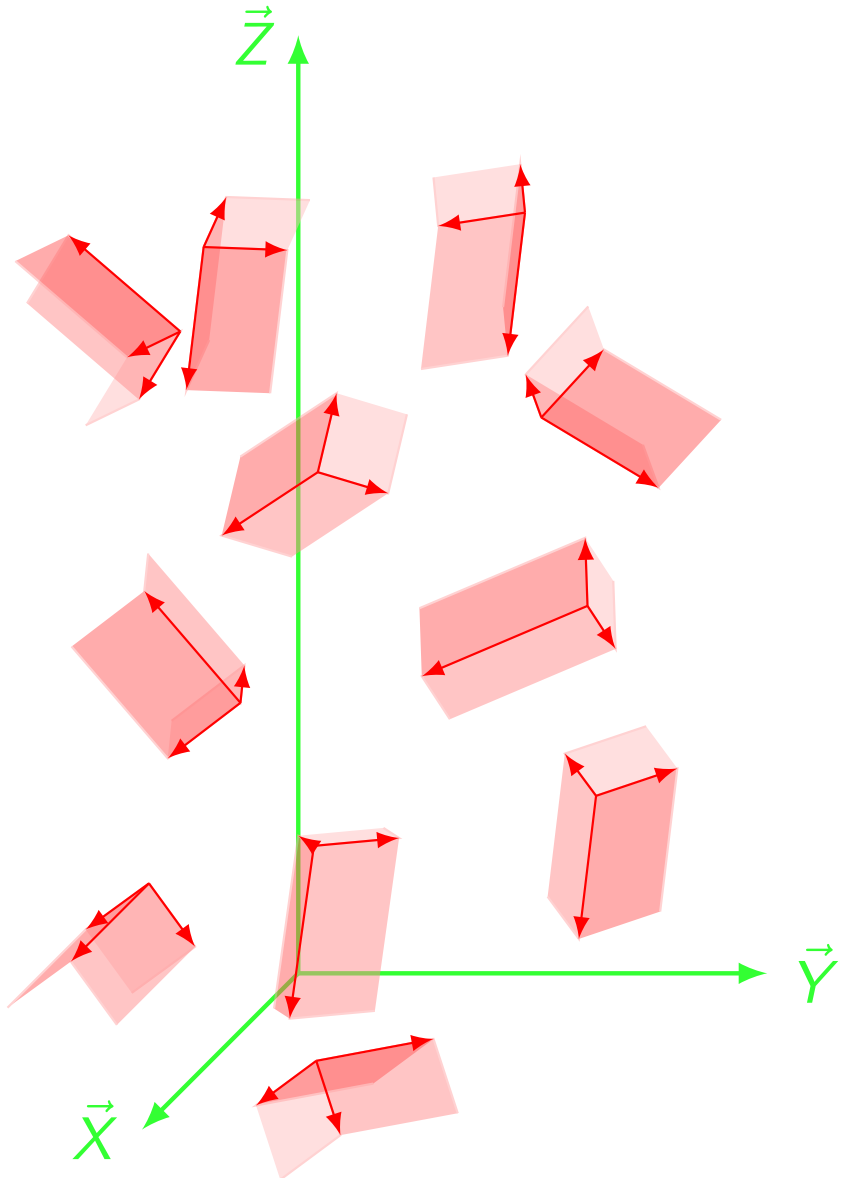
The coordinate transform $\mathbf{O}$ with

$$(r_1, r_2, r_3)^t = \mathbf{O}\,(h_1, h_2, h_3)^t$$

is called crystal orientation.

The orientation $\mathbf{O}$ maps the specimen coordinate system $\vec{X}, \vec{Y}, \vec{Z}$ onto the crystal coordinate systems $\vec{x}, \vec{y}, \vec{z}$.

The orientation $\mathbf{O}$ is well defined only up the crystal symmetry.



Crystal Orientations

Let a vector $\vec{v}$ be given by specimen coordinates $(r_1, r_2, r_3)^t$ and crystal coordinates $(h_1, h_2, h_3)^t$, i.e.,

$$\vec{v} = r_1\vec{X} + r_2\vec{Y} + r_3\vec{Y} = h_1\vec{x} + h_2\vec{y} + h_3\vec{z}.$$

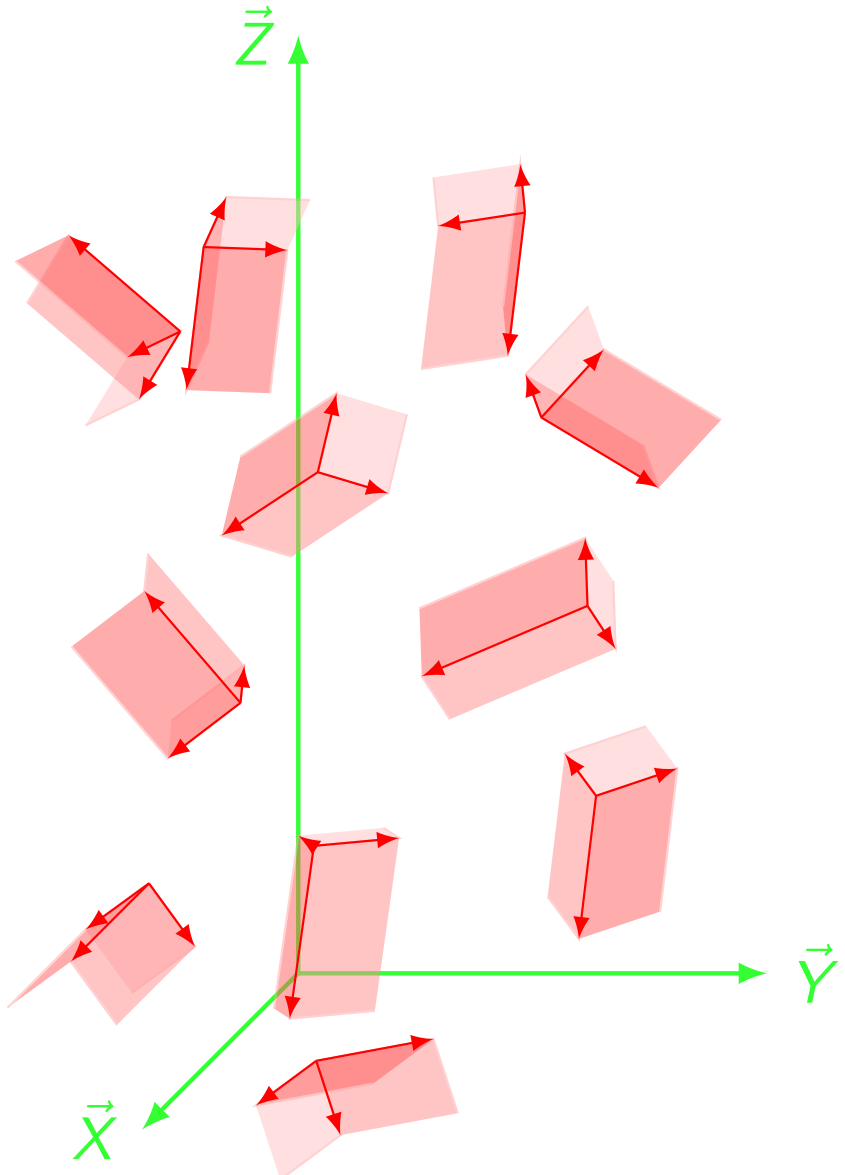
The coordinate transform $\mathbf{O}$ with

$$(r_1, r_2, r_3)^t = \mathbf{O}\,(h_1, h_2, h_3)^t$$

is called crystal orientation.

The orientation $\mathbf{O}$ maps the specimen coordinate system $\vec{X}, \vec{Y}, \vec{Z}$ onto the crystal coordinate systems $\vec{x}, \vec{y}, \vec{z}$.

The orientation $\mathbf{O}$ is well defined only up the crystal symmetry.



Defining Orientations

define an orientation by Euler angles

```
CS = crystalSymmetry('321')
SS = specimenSymmetry('1')
O  = orientation('Euler',10*degree,5*degree,0,CS,SS)
```

```
O = orientation (show methods, plot)
  size: 1 x 1
  crystal symmetry: 321, X||a*, Y||b, Z||c*
  sample symmetry : 1

  Bunge Euler angles in degree
  Psi  Theta    Phi  Inv.
   10      5      0    0
```

import orientations

```
O  = loadOrientation('filename',CS,SS,...
 'ColumnNames',{'phi1','Phi','phi2'})
```

define orientations by Miller indices

```
O  = orientation('Miller',[h,k,l],[u,v,w],CS,SS)
```

Defining Orientations

define an orientation by Euler angles

```
CS = crystalSymmetry('321')
SS = specimenSymmetry('1')
O  = orientation('Euler',10*degree,5*degree,0,CS,SS)
```

import orientations

```
O  = loadOrientation('filename',CS,SS,...
 'ColumnNames',{'phi1','Phi','phi2'})
```

```
O = orientation (show methods, plot)
  size: 1000 x 1
  crystal symmetry: 321, X||a*, Y||b, Z||c*
  sample symmetry : 1
```

define orientations by Miller indices

```
O  = orientation('Miller',[h k l],[u,v,w],CS,SS)
```

standard orientations: Cube, CubeND22, CubeND45, CubeRD, Goss, invGoss, Copper, Copper2, SR, SR2, SR3, SR4, Brass, Brass2,

Defining Orientations

define an orientation by Euler angles

```
CS = crystalSymmetry('321')
SS = specimenSymmetry('1')
O  = orientation('Euler',10*degree,5*degree,0,CS,SS)
```

import orientations

```
O  = loadOrientation('filename',CS,SS,...
 'ColumnNames',{'phi1','Phi','phi2'})
```

define orientations by Miller indices

```
O  = orientation('Miller',[h k l],[u,v,w],CS,SS)
```

standard orientations: `Cube`, `CubeND22`, `CubeND45`, `CubeRD`, `Goss`, `invGoss`, `Copper`, `Copper2`, `SR`, `SR2`, `SR3`, `SR4`, `Brass`, `Brass2`, `PLage`, `PLage2`, `QLage`, `QLage2`, `QLage3`, `QLage4`

```
O  = brassOrientation(CS,SS)
```

Defining Orientations

define an orientation by Euler angles

```
CS = crystalSymmetry('321')
SS = specimenSymmetry('1')
O  = orientation('Euler',10*degree,5*degree,0,CS,SS)
```

import orientations

```
O  = loadOrientation('filename',CS,SS,...
 'ColumnNames',{'phi1','Phi','phi2'})
```

define orientations by Miller indices

```
O  = orientation('Miller',[h k l],[u,v,w],CS,SS)
```

standard orientations: `Cube`, `CubeND22`, `CubeND45`, `CubeRD`, `Goss`, `invGoss`, `Copper`, `Copper2`, `SR`, `SR2`, `SR3`, `SR4`, `Brass`, `Brass2`, `PLage`, `PLage2`, `QLage`, `QLage2`, `QLage3`, `QLage4`

```
O  = brassOrientation(CS,SS)
```

Calculating with Orientations

find all symmetrically equivalent orientations

symmetrise(O)

```
ans = orientation (show methods, plot)
  size: 6 x 1
  crystal symmetry: 321, X||a*, Y||b, Z||c*
  sample symmetry : 1

  Roe Euler angles in degree
  Psi Theta   Phi Inv.
   10     5     0   0
   10     5   120   0
   10     5   240   0
  190   175    60   0
  190   175   180   0
  190   175   300   0
```

convert crystal into specimen coordinates

```
h = Miller(1,0,-1,0,CS);
r = O * h
```

Calculating with Orientations

find all symmetrically equivalent orientations

```
symmetrise(O)
```

convert crystal into specimen coordinates

```
h = Miller(1,0,-1,0,CS);
r = O * h
```

```
r = vector3d (show methods, plot)
size: 1 x 1
        x         y         z
0.984808  0.173648         0
```

convert specimen into crystal coordinates

```
inv(O) * r
```

change specimen coordinates

```
R = rotation('axis',zvector,'angle',90*degree)
```

Calculating with Orientations

find all symmetrically equivalent orientations

```
symmetrise(O)
```

convert crystal into specimen coordinates

```
h = Miller(1,0,-1,0,CS);
r = O * h
```

convert specimen into crystal coordinates

```
inv(O) * r
```

```
ans = Miller (show methods, plot)
  size: 1 x 1
  symmetry: 321, X||a*, Y||b, Z||c*
  h  1
  k  0
  i -1
  l  0
```

change specimen coordinates

Calculating with Orientations

find all symmetrically equivalent orientations

```
symmetrise(O)
```

convert crystal into specimen coordinates

```
h = Miller(1,0,-1,0,CS);
r = O * h
```

convert specimen into crystal coordinates

```
inv(O) * r
```

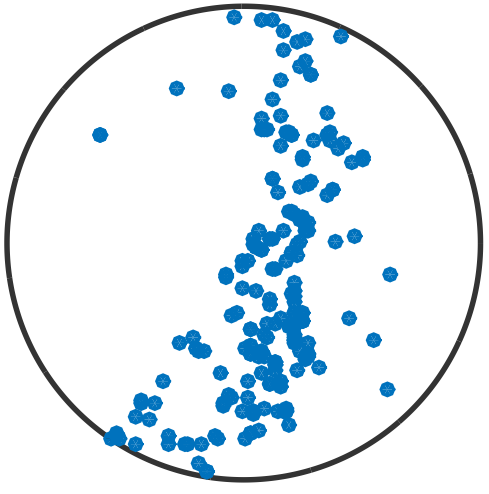
change specimen coordinates

```
R  = rotation('axis',zvector,'angle',90*degree)
O2 = R * O1
```

Pole Figures and Inverse Pole Figures

Lattice planes in specimen coordinates

```
h = [Miller(1,0,0,ori.CS),...
     Miller(0,1,0,ori.CS)]
plot(symmetrise(ori) * h(1))
plot(ori * symmetrise(h(1)))
```



```
plotPDF(ori,h)
plotPDF(ori,h,'contourf')
```

specimen directions in crystal coordinates

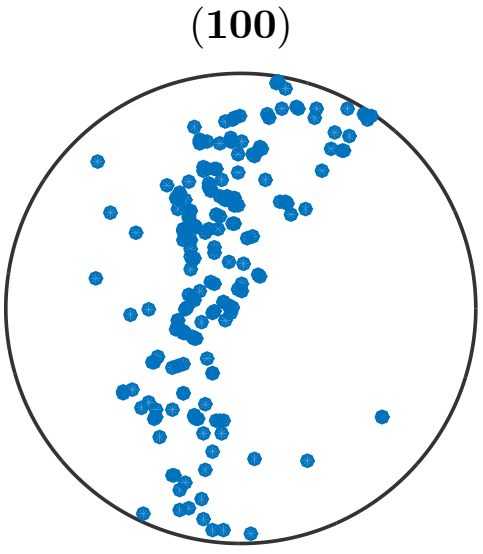
```
plot(symmetrise(inv(ori)) * xvector)
plot(symmetrise(inv(ori) * xvector))
```

```
v = [xvector,yvector]
plotIPDF(ori,v)
plotIPDF(ori,v,'contourf')
```

Pole Figures and Inverse Pole Figures

Lattice planes in specimen coordinates

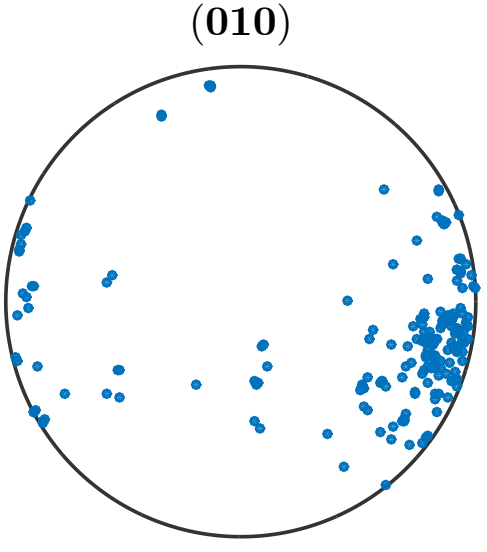
```
h = [Miller(1,0,0,ori.CS),...
     Miller(0,1,0,ori.CS)]
plot(symmetrise(ori) * h(1))
plot(ori * symmetrise(h(1)))
```




```
plotPDF(ori,h)
plotPDF(ori,h,'contourf')
```

specimen directions in crystal coordinates

```
plot(symmetrise(inv(ori)) * xvector)
plot(symmetrise(inv(ori) * xvector))
v = [xvector,yvector]
plotIPDF(ori,v)
plotIPDF(ori,v,'contourf')
```



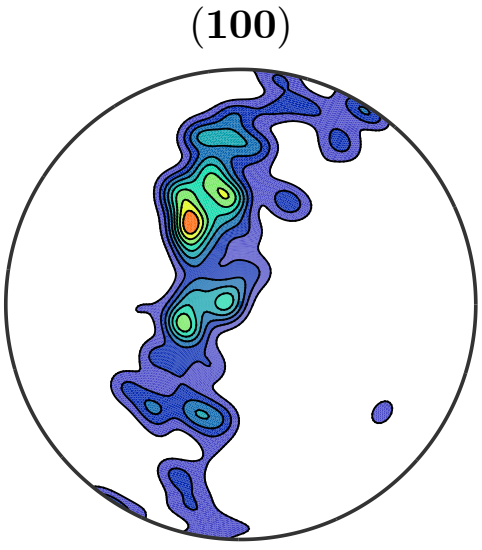
Pole Figures and Inverse Pole Figures

Lattice planes in specimen coordinates

```
h = [Miller(1,0,0,ori.CS),...
     Miller(0,1,0,ori.CS)]
plot(symmetrise(ori) * h(1))
plot(ori * symmetrise(h(1)))
```

```
plotPDF(ori,h)
```

```
plotPDF(ori,h,'contourf')
```

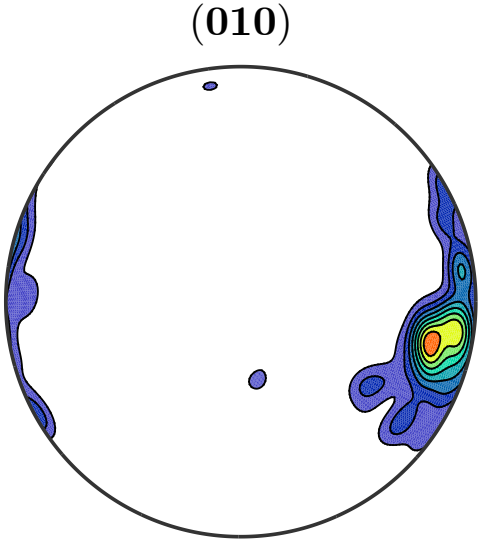


specimen directions in crystal coordinates

```
plot(symmetrise(inv(ori)) * xvector)
plot(symmetrise(inv(ori) * xvector))
```

```
v = [xvector,yvector]
plotIPDF(ori,v)
```

```
plotIPDF(ori,v,'contourf')
```



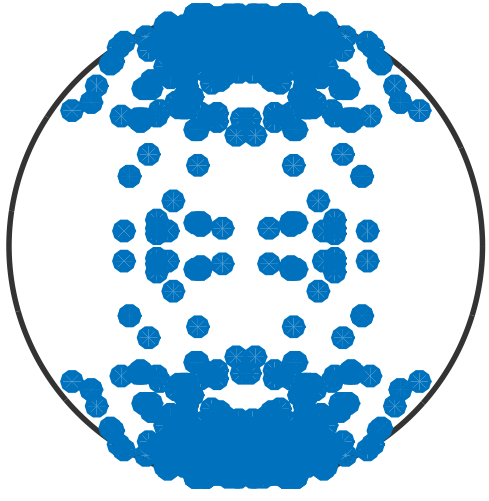
Pole Figures and Inverse Pole Figures

Lattice planes in specimen coordinates

```
h = [Miller(1,0,0,ori.CS),...
     Miller(0,1,0,ori.CS)]
plot(symmetrise(ori) * h(1))
plot(ori * symmetrise(h(1)))
```

```
plotPDF(ori,h)
```

```
plotPDF(ori,h,'contourf')
```



specimen directions in crystal coordinates

```
plot(symmetrise(inv(ori)) * xvector)
plot(symmetrise(inv(ori) * xvector))
```

```
v = [xvector,yvector]
plotIPDF(ori,v)
```

```
plotIPDF(ori,v,'contourf')
```

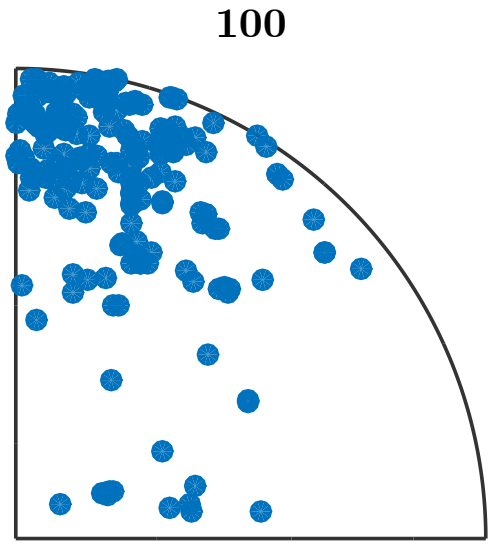
Pole Figures and Inverse Pole Figures

Lattice planes in specimen coordinates

```
h = [Miller(1,0,0,ori.CS),...
     Miller(0,1,0,ori.CS)]
plot(symmetrise(ori) * h(1))
plot(ori * symmetrise(h(1)))
```

```
plotPDF(ori,h)
```

```
plotPDF(ori,h,'contourf')
```

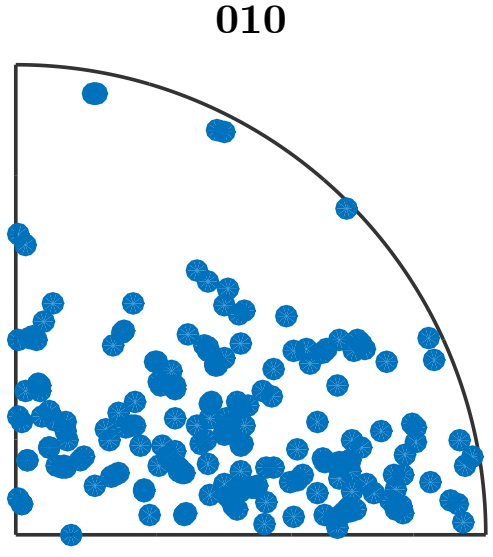


specimen directions in crystal coordinates

```
plot(symmetrise(inv(ori)) * xvector)
plot(symmetrise(inv(ori) * xvector))
```

```
v = [xvector,yvector]
plotIPDF(ori,v)
```

```
plotIPDF(ori,v,'contourf')
```



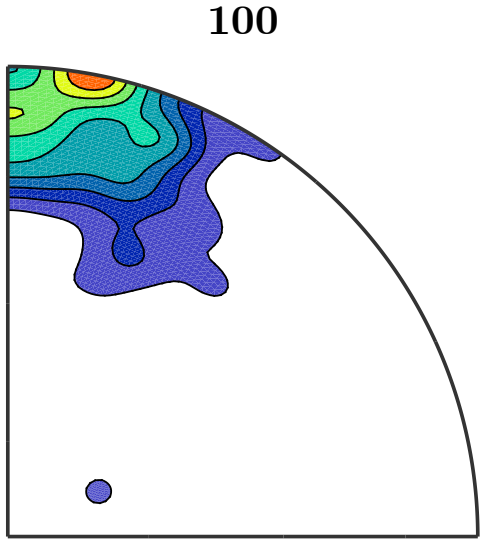
Pole Figures and Inverse Pole Figures

Lattice planes in specimen coordinates

```
h = [Miller(1,0,0,ori.CS),...
     Miller(0,1,0,ori.CS)]
plot(symmetrise(ori) * h(1))
plot(ori * symmetrise(h(1)))
```

```
plotPDF(ori,h)
```

```
plotPDF(ori,h,'contourf')
```

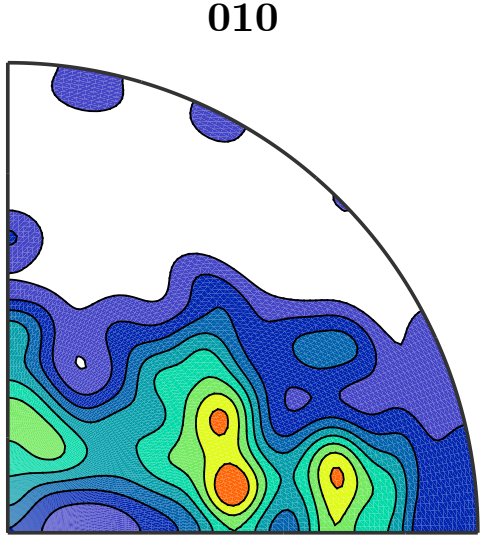


specimen directions in crystal coordinates

```
plot(symmetrise(inv(ori)) * xvector)
plot(symmetrise(inv(ori) * xvector))
```

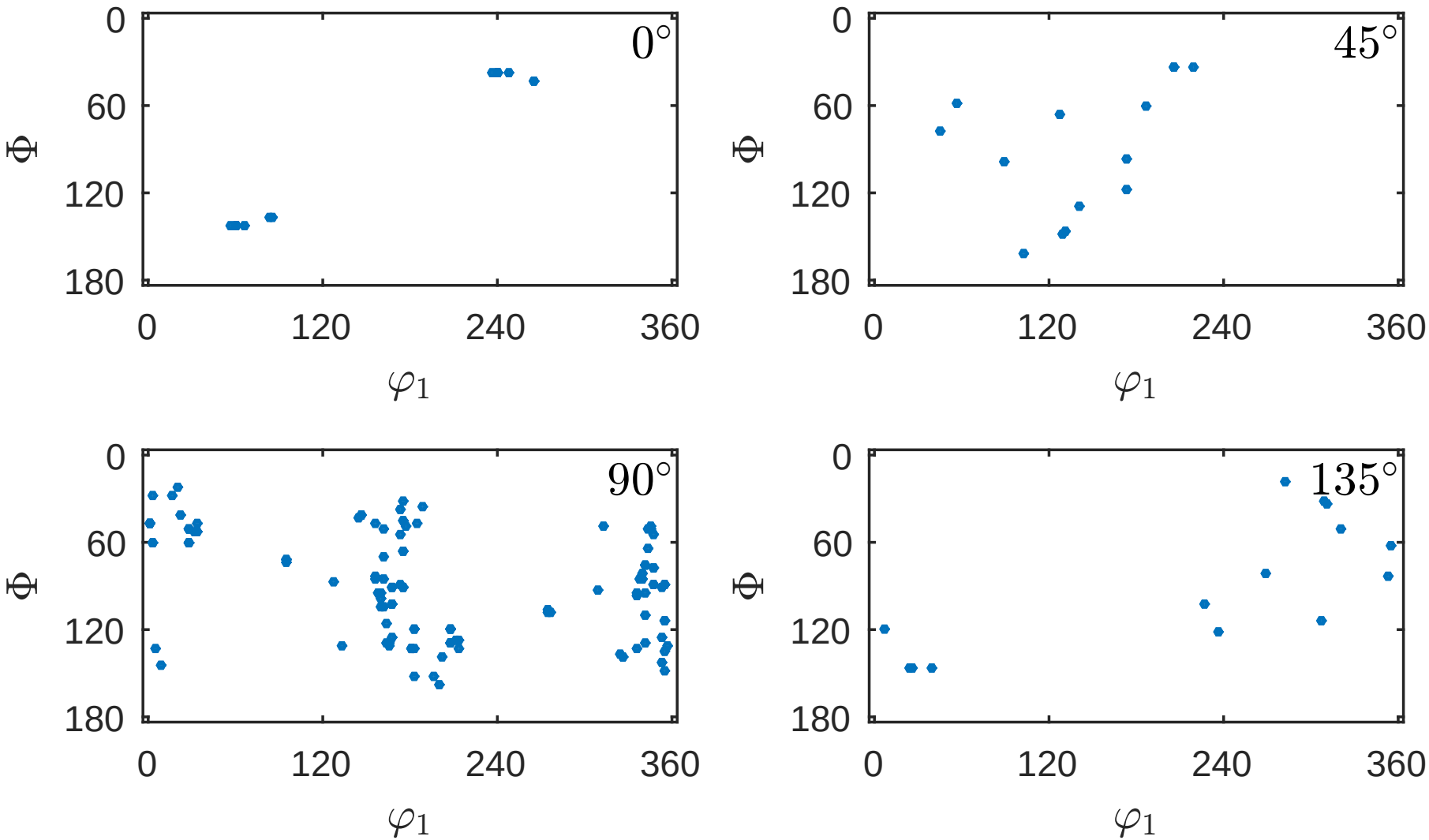
```
v = [xvector,yvector]
plotIPDF(ori,v)
```

```
plotIPDF(ori,v,'contourf')
```



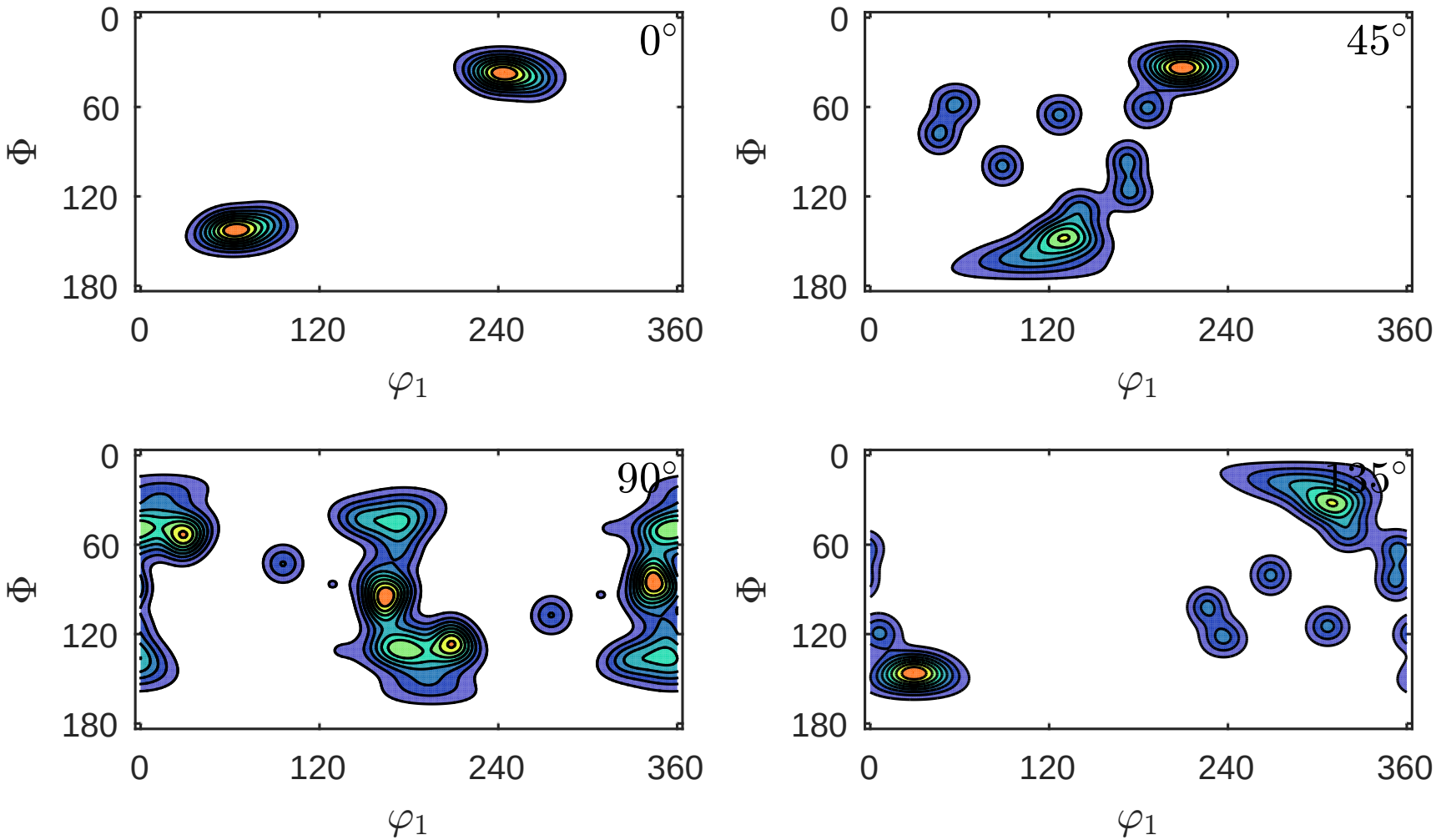
Plotting in Orientation Space

```
plotODF(ori,'phi2',[0 45 90]*degree)
```



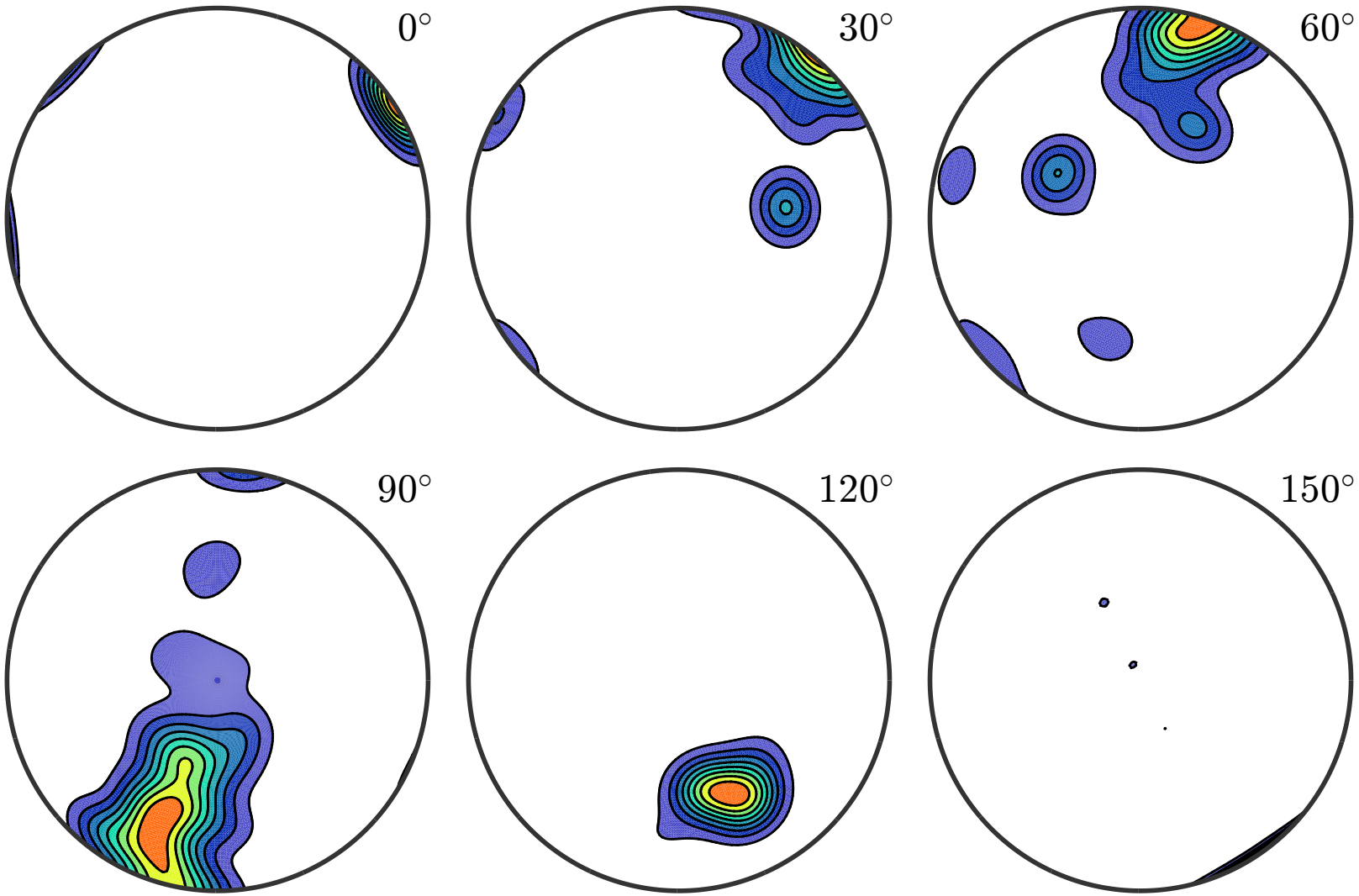
Plotting in Orientation Space

```
plotODF(ori,'phi2',[0 45 90]*degree,'contourf',...
        'halfwidth',10*degree)
```



Plotting in Orientation Space

```
plotODF(ori,'sigma',(0:5)*30*degree,'contourf')
```



Operations on Orientations

the misorientation angle and axis

```
angle(ori1,ori2) / degree
axis(ori1,ori2)
```

```
ans = Miller (show methods, plot)
  size: 1 x 1
  mineral: Forsterite (mmm)
  h 2.729
  k 7.6909
  l 1.9193
```

statistics

```
mean(ori)
```

volume portions

```
volume(ori,mean(ori),10*degree)
fibreVolume(ori,Miller(1,0,0,ori.CS),xvector)
```

export to ASCII file

Operations on Orientations

the misorientation angle and axis

```
angle(ori1,ori2) / degree
axis(ori1,ori2)
```

statistics

```
mean(ori)
```

```
ans = orientation (show methods, plot)
  size: 1 x 1
  crystal symmetry : Forsterite (mmm)
  specimen symmetry: 1

  Bunge Euler angles in degree
     phi1      Phi     phi2     Inv.
  342.532 68.3179 284.955        0
```

volume portions

```
volume(ori,mean(ori),10*degree)
fibreVolume(ori,Miller(1,0,0,ori.CS),xvector)
```

Operations on Orientations

the misorientation angle and axis

```
angle(ori1,ori2) / degree
axis(ori1,ori2)
```

statistics

```
mean(ori)
```

volume portions

```
volume(ori,mean(ori),10*degree)
fibreVolume(ori,Miller(1,0,0,ori.CS),xvector)
```

export to ASCII file

```
export(ori,'file.txt','bunge','degree')
```

Operations on Orientations

the misorientation angle and axis

```
angle(ori1,ori2) / degree
axis(ori1,ori2)
```

statistics

```
mean(ori)
```

volume portions

```
volume(ori,mean(ori),10*degree)
fibreVolume(ori,Miller(1,0,0,ori.CS),xvector)
```

export to ASCII file

```
export(ori,'file.txt','bunge','degree')
```

Coordinate Transforms

Remember, an orientation converts crystal into specimen coordinates

```
r = O * Miller(1,0,-1,0,CS)
```

```
r = vector3d (show methods, plot)
size: 1 x 1
        x         y         z
0.984808  0.173648         0
```

its inverse converts specimen into crystal coordinates

```
inv(O) * r
```

Hence, for two orientation $\mathbf{O_1}$ and $\mathbf{O_2}$ the concatenation $\mathbf{inv(O_2)} \cdot \mathbf{O_1}$ converts crystal coordinates into crystal coordinates

```
O2 = inv(O2) * O1 * Miller(1,0,-1,0,CS)
```

Coordinate Transforms

Remember, an orientation converts crystal into specimen coordinates

```
r = O * Miller(1,0,-1,0,CS)
```

its inverse converts specimen into crystal coordinates

```
inv(O) * r
```

```
ans = Miller (show methods, plot)
  size: 1 x 1
  symmetry: 321, X||a*, Y||b, Z||c*
  h  1
  k  0
  i -1
  l  0
```

Hence, for two orientation $\mathbf{O_1}$ and $\mathbf{O_2}$ the concatenation $\mathbf{inv(O_2) \cdot O_1}$ converts crystal coordinates into crystal coordinates

```
O2 = inv(O2) * O1 * Miller(1,0,-1,0,CS)
```

Coordinate Transforms

Remember, an orientation converts crystal into specimen coordinates

```
r = O * Miller(1,0,-1,0,CS)
```

its inverse converts specimen into crystal coordinates

```
inv(O) * r
```

Hence, for two orientation $\mathbf{O_1}$ and $\mathbf{O_2}$ the concatenation $\mathbf{inv(O_2)} \cdot \mathbf{O_1}$ converts crystal coordinates into crystal coordinates

```
O2 = inv(O2) * O1 * Miller(1,0,-1,0,CS)
```

```
ans = Miller (show methods, plot)
  size: 1 x 1
  symmetry: 321, X||a*, Y||b, Z||c*
  h  1
  k  0
  i -1
  l  0
```

Misorientations

The misorientation **MO** converts crystal coordinates from one crystal into crystal coordinates of another crystal.

```
MO = inv(O1) * O2
```

```
MO = misorientation (show methods, plot)
  size: 1 x 1
  crystal symmetry: Quartz (P 32 2 1, X||a*, Y||b, Z||c*)
  crystal symmetry: Fe (m-3m)

  Roe Euler angles in degree
      Psi    Theta      Phi  Inv.
  2.58238 56.6839   73.197     0
```

compute misorientation angle modulo symmetry

```
angle(O1,O2) / degree, angle(MO) / degree
```

compute the misorientation axis (the axis corresponding to the rotation with minimal angle)

```
axis(MO)
```

Misorientations

The misorientation **MO** converts crystal coordinates from one crystal into crystal coordinates of another crystal.

```
MO = inv(O1) * O2
```

compute misorientation angle modulo symmetry

```
angle(O1,O2) / degree, angle(MO) / degree
```

compute the misorientation axis (the axis corresponding to the rotation with minimal angle)

```
axis(MO)
```

Misorientations

The misorientation **MO** converts crystal coordinates from one crystal into crystal coordinates of another crystal.

```
MO = inv(O1) * O2
```

compute misorientation angle modulo symmetry

```
angle(O1,O2) / degree, angle(MO) / degree
```

compute the misorientation axis (the axis corresponding to the rotation with minimal angle)

```
axis(MO)
```

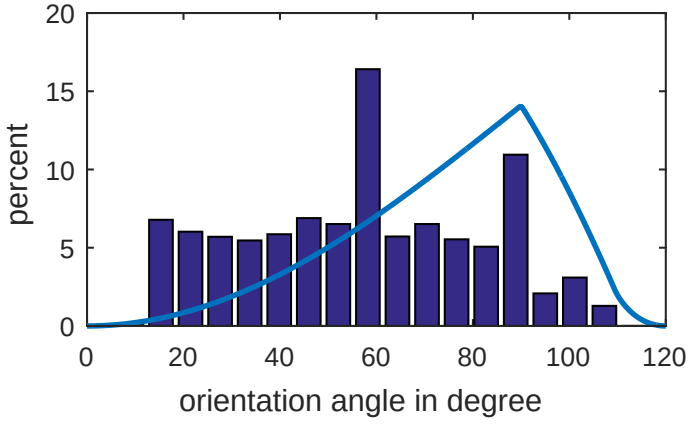
```
ans = Miller (show methods, plot)
  size: 1 x 1
  symmetry: C2
  h -2
  k  3
  l  9
```

Misorientation Statistics

```
plotAngleDistribution(mori)
hold on
plotAngleDistribution(mori.CS)
hold off
```

```
plotAxisDistribution(mori)
```

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

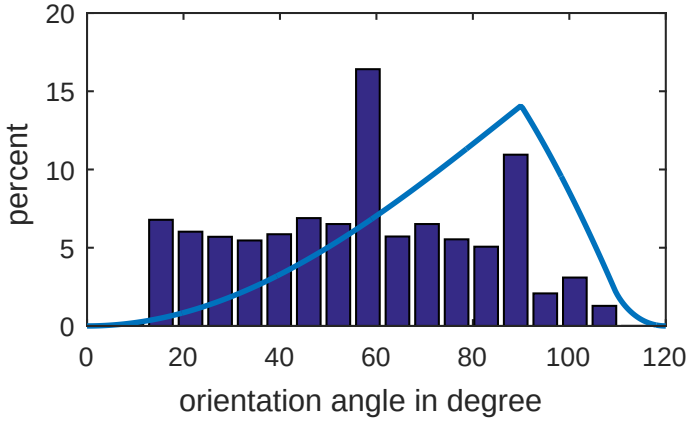


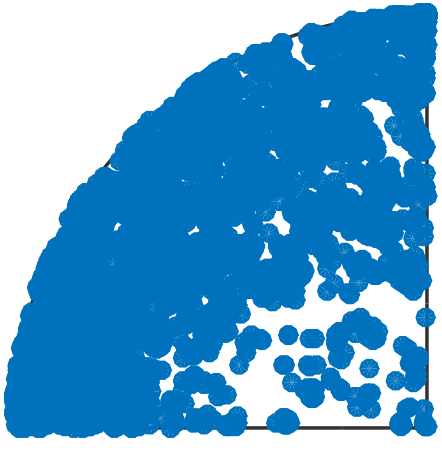
Misorientation Statistics

```
plotAngleDistribution(mori)
hold on
plotAngleDistribution(mori.CS)
hold off
```

```
plotAxisDistribution(mori)
```

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



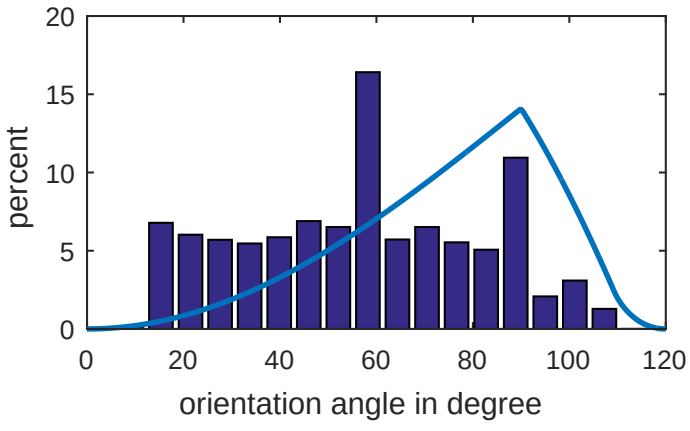


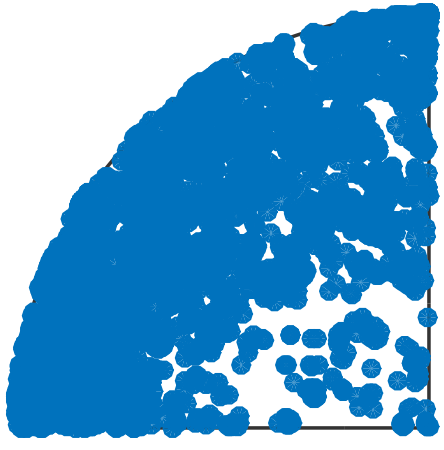
Misorientation Statistics

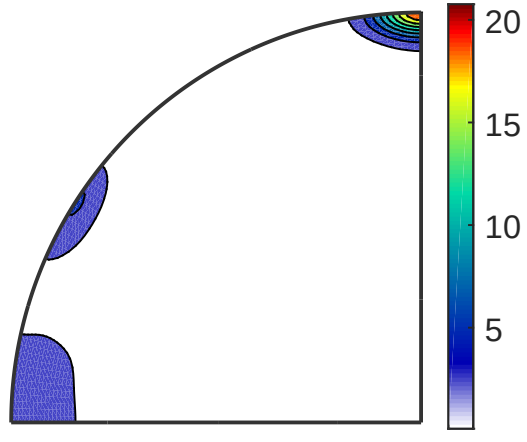
```
plotAngleDistribution(mori)
hold on
plotAngleDistribution(mori.CS)
hold off
```

```
plotAxisDistribution(mori)
```

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







Phase Transitions

We want to model the phase transition from magnetite to hematite given by the orientation relationship $\{111\}_m||\{0001\}_h$ and $\{\bar{1}01\}_m||\{00\bar{1}0\}_h$

```
CS_Mag = loadCif('Magnetite')
CS_Hem = loadCif('Hematite')
```

```
cs_Magnetite = crystalSymmetry (show methods, plot)
  mineral : Magnetite
  symmetry: m-3m
  a, b, c : 8.4, 8.4, 8.4

cs_Hematite = crystalSymmetry (show methods, plot)
  mineral : Hematite
  symmetry : -3m1
  a, b, c : 5, 5, 14
  reference frame: X||a*, Y||b, Z||c
```

```
Mag2Hem = orientation('map',...
  Miller(1,1,1,CS_Mag),Miller(0,0,0,1,CS_Hem),...
  Miller(-1,0,1,CS_Mag),Miller(1,0,-1,0,CS_Hem),...
  CS_Mag,CS_Hem)
```

Phase Transitions

We want to model the phase transition from magnetite to hematite given by the orientation relationship $\{111\}_m||\{0001\}_h$ and $\{\bar{1}01\}_m||\{00\bar{1}0\}_h$

```
CS_Mag = loadCif('Magnetite')
CS_Hem = loadCif('Hematite')
```

```
Mag2Hem = orientation('map',...
   Miller(1,1,1,CS_Mag),Miller(0,0,0,1,CS_Hem),...
   Miller(-1,0,1,CS_Mag),Miller(1,0,-1,0,CS_Hem),...
   CS_Mag,CS_Hem)
```

```
Mag2Hem = misorientation (show methods, plot)
  size: 1 x 1
  crystal symmetry : Magnetite (m-3m)
  crystal symmetry : Hematite (-3m1, X||a*, Y||b, Z||c)

  Bunge Euler angles in degree
  phi1 Phi phi2 Inv.
  120 54.7356 45 0
```

Phase Transition

For measured hematite orientation

```
ori_Hem = orientation('Euler',0,0,0,CS_Hem)
```

```
ori_Hem = orientation (show methods, plot)
  size: 1 x 1
  crystal symmetry : Hematite (-3m1, X||a*, Y||b, Z||c)
  specimen symmetry: 1

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

we can compute the initial magnetite orientation by

```
ori_Mag = ori_Hem * Mag2Hem
```

We should care about symmetric equivalence

```
ori_Mag = ori_Hem * symmetrise(Mag2Hem)
```

MTEX keeps track about the symmetries throughout all computations and warns in case of mismatch.

Phase Transition

For measured hematite orientation

```
ori_Hem = orientation('Euler',0,0,0,CS_Hem)
```

we can compute the initial magnetite orientation by

```
ori_Mag = ori_Hem * Mag2Hem
```

```
ori_Mag = orientation (show methods, plot)
  size: 1 x 1 crystal
  symmetry : Magnetite (m-3m)
  specimen symmetry: 1

  Bunge Euler angles in degree
  phi1 Phi phi2 Inv.
  120 54.7356 45 0
```

We should care about symmetric equivalence

```
ori_Mag = ori_Hem * symmetrise(Mag2Hem)
```

MTEX keeps track about the symmetries throughout all computations and warns in case of mismatch.

Phase Transition

For measured hematite orientation

```
ori_Hem = orientation('Euler',0,0,0,CS_Hem)
```

we can compute the initial magnetite orientation by

```
ori_Mag = ori_Hem * Mag2Hem
```

We should care about symmetric equivalence

```
ori_Mag = ori_Hem * symmetrise(Mag2Hem)
```

```
ori_Mag = orientation (show methods, plot)
  size: 1 x 96
  crystal symmetry : Magnetite (m-3m)
  specimen symmetry: 1
```

MTEX keeps track about the symmetries throughout all computations and warns in case of mismatch.

Phase Transition

For measured hematite orientation

```
ori_Hem = orientation('Euler',0,0,0,CS_Hem)
```

we can compute the initial magnetite orientation by

```
ori_Mag = ori_Hem * Mag2Hem
```

We should care about symmetric equivalence

```
ori_Mag = ori_Hem * symmetrise(Mag2Hem)
```

```
ori_Mag = orientation (show methods, plot)
  size: 1 x 96
  crystal symmetry : Magnetite (m-3m)
  specimen symmetry: 1
```

MTEX keeps track about the symmetries throughout all computations and warns in case of mismatch.

Relationship Between Lattice Planes

Compute the minimal angle between two lattice planes of two different grains having different orientation and different phase.

```
m_Mag = Miller(1,0,0,cs_Magnetite);
m_Hem = Miller(1,1,-2,0,cs_Hematite);
```

```
m_Mag = Miller (show methods, plot)
 size: 1 x 1
 mineral: Magnetite (432)
  h 1
  k 0
  l 0

m_Hem = Miller (show methods, plot)
 size: 1 x 1
 mineral: Hematite (321, X||a*, Y||b, Z||c)
  h  1
  k  1
  i -2
  l  0
```

The orientation relation should be given by

Relationship Between Lattice Planes

Compute the minimal angle between two lattice planes of two different grains having different orientation and different phase.

```
m_Mag = Miller(1,0,0,cs_Magnetite);
m_Hem = Miller(1,1,-2,0,cs_Hematite);
```

The orientation relation should be given by

```
Mag2Hem = orientation('map',...
   Miller(1,1,1,CS_Mag),Miller(0,0,0,1,CS_Hem),...
   Miller(-1,0,1,CS_Mag),Miller(1,0,-1,0,CS_Hem),...
   CS_Mag,CS_Hem)
```

```
Mag2Hem = misorientation (show methods, plot)
  size: 1 x 1
  crystal symmetry : Magnetite (m-3m)
  crystal symmetry : Hematite (-3m1, X||a*, Y||b, Z||c)

  Bunge Euler angles in degree
  phi1 Phi phi2 Inv.
  120 54.7356 45 0
```

Relationship Between Lattice Planes

Compute the minimal angle between two lattice planes of two different grains having different orientation and different phase.

```
m_Mag = Miller(1,0,0,cs_Magnetite);
m_Hem = Miller(1,1,-2,0,cs_Hematite);
```

The orientation relation should be given by

```
Mag2Hem = orientation('map',...
   Miller(1,1,1,CS_Mag),Miller(0,0,0,1,CS_Hem),...
   Miller(-1,0,1,CS_Mag),Miller(1,0,-1,0,CS_Hem),...
   CS_Mag,CS_Hem)
```

The minimum angle

```
min( angle( Mag2Hem * symmetrise(m_Mag), ...
            m_Hem ) ) / degree
```

```
35.2644
```