Crystal Geometry

R. Hielscher

Faculty of Mathematics,
Chemnitz University of Technology, Germany

MTEX Workshop 2016

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 $\vec{X}, \vec{Y}, \vec{Z}$

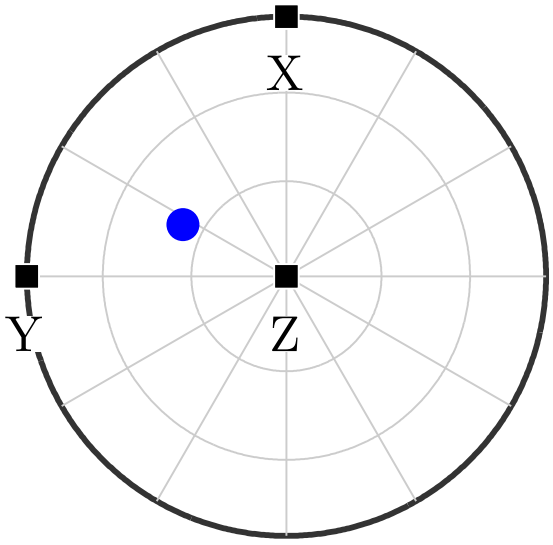
$$\vec{r} = x \cdot \vec{X} + y \cdot \vec{Y} + z \cdot \vec{Z}$$

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 $\vec{X}, \vec{Y}, \vec{Z}$

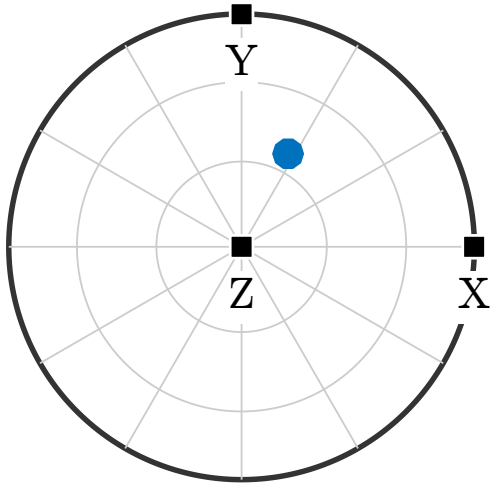
$$\vec{r} = x \cdot \vec{X} + y \cdot \vec{Y} + z \cdot \vec{Z}$$

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

```
vector3d.X, vector3d.Y, vector3d.Z
```

combine vectors

```
r = [vector3d.X, vector3d.Y, 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

```
vector3d.X, vector3d.Y, vector3d.Z
```

combine vectors

```
r = [vector3d.X, vector3d.Y, 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

```
vector3d.X, vector3d.Y, vector3d.Z
```

combine vectors

```
r = [vector3d.X, vector3d.Y, 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

```
vector3d.X, vector3d.Y, vector3d.Z
```

combine vectors

```
r = [vector3d.X, vector3d.Y, 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*vector3d.X - vector3d.Y;
```

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*vector3d.X - vector3d.Y;
```

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*vector3d.X - vector3d.Y;
```

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 rotations, orientations

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 rotations, orientations, tensors, EBSD data, grains, boundary segments, triple points, 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) = vector3d.Y
```

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) = vector3d.Y
```

```
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) = vector3d.Y
```

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) = vector3d.Y
```

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) = vector3d.Y
```

remove the second vector completely

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

change the x coordinate of all vectors

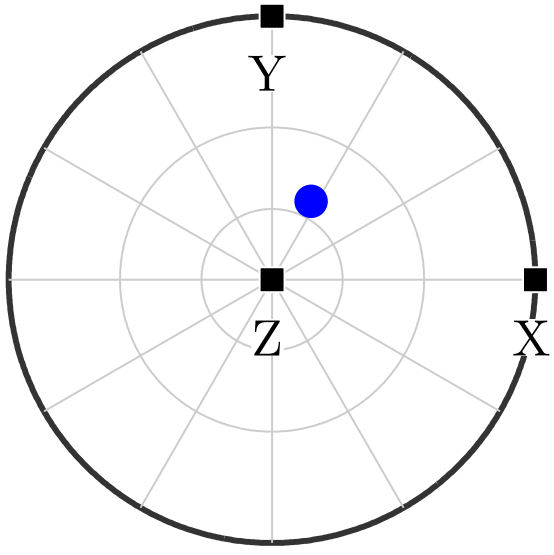
```
r.x = 0
```

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

Plotting Vectors

spherical projections: `earea, edist, eangle`

```
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
```

scatter plots

```
v = vector3d.rand(1000)
plot(v)
```

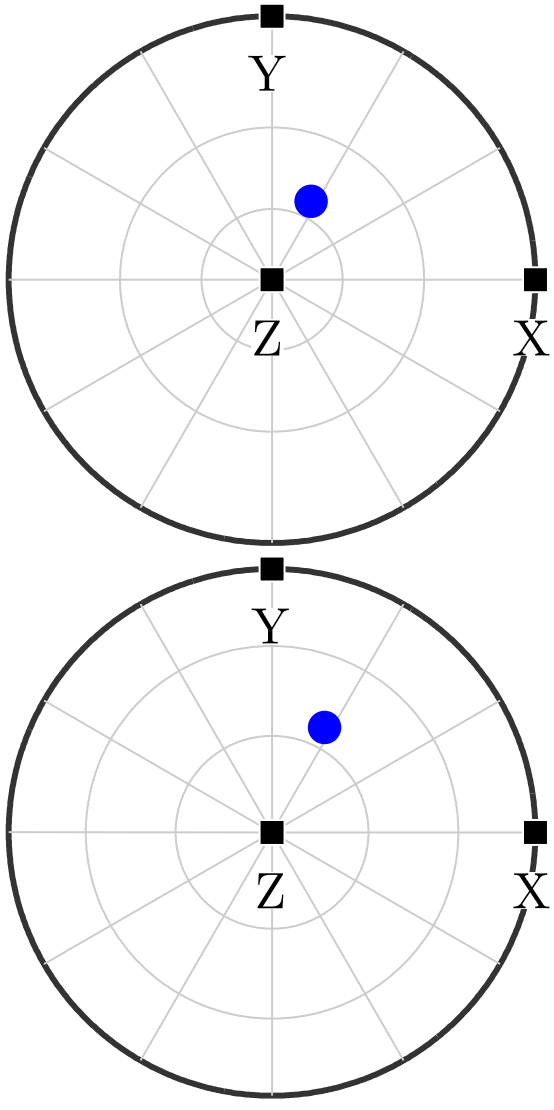
contour plots

```
plot(v,'contourf')
```

Plotting Vectors

spherical projections: `earea`, `edist`, `eangle`

```
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
```

scatter plots

```
v = vector3d.rand(1000)
plot(v)
```

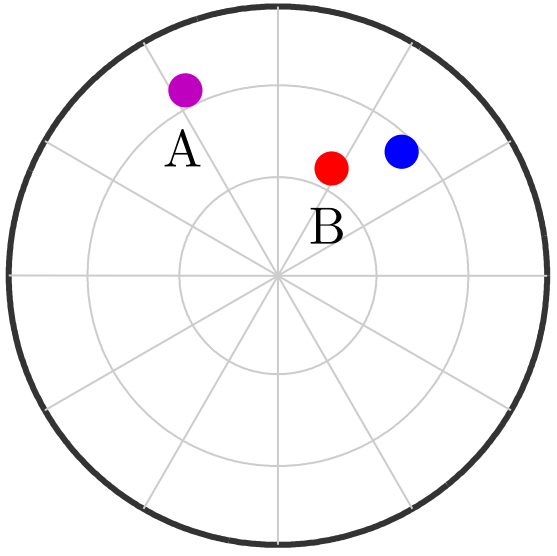
contour plots

```
plot(v,'contourf')
```

Plotting Vectors

spherical projections: `earea`, `edist`, `eangle`

```
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
```

scatter plots

```
v = vector3d.rand(1000)
plot(v)
```

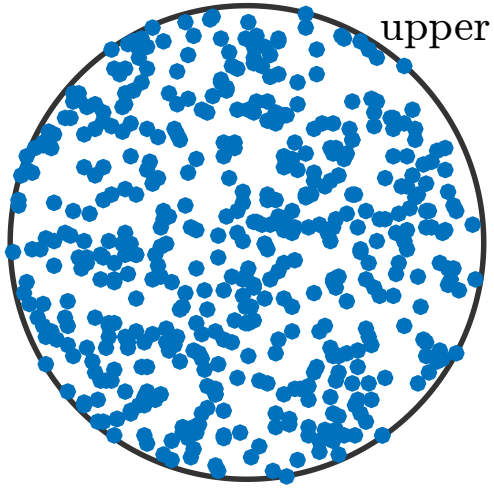
contour plots

```
plot(v,'contourf')
```

Plotting Vectors

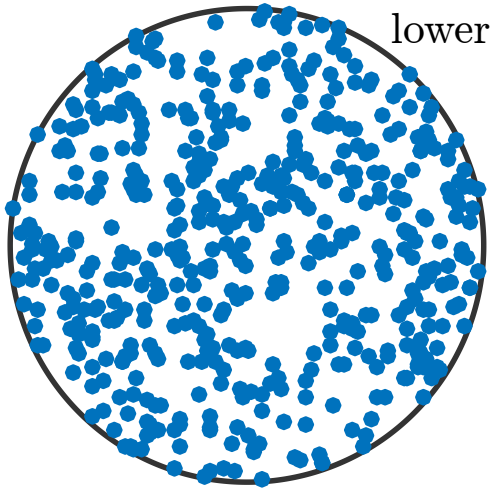
spherical projections: `earea`, `edist`, `eangle`

```
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
```



scatter plots

```
v = vector3d.rand(1000)
plot(v)
```

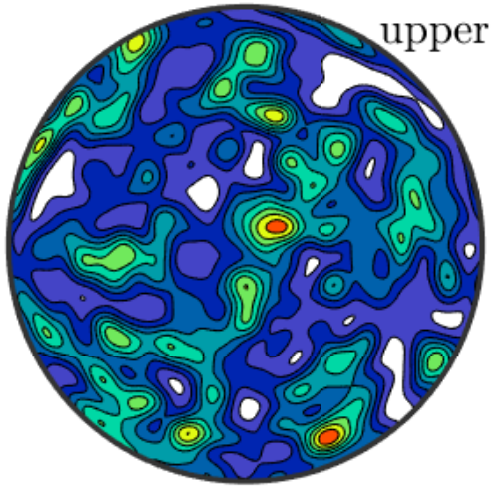
contour plots

```
plot(v,'contourf')
```

Plotting Vectors

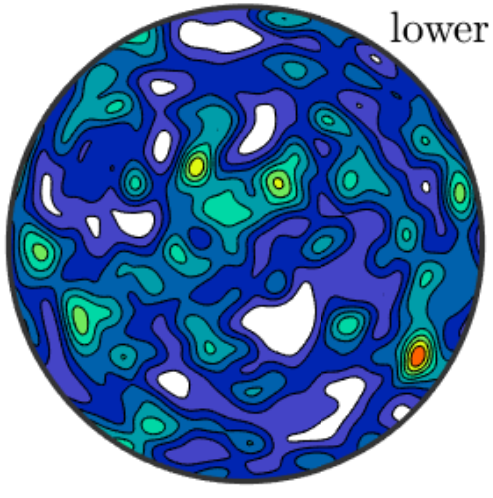
spherical projections: `earea, edist, eangle`

```
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
```



scatter plots

```
v = vector3d.rand(1000)
plot(v)
```

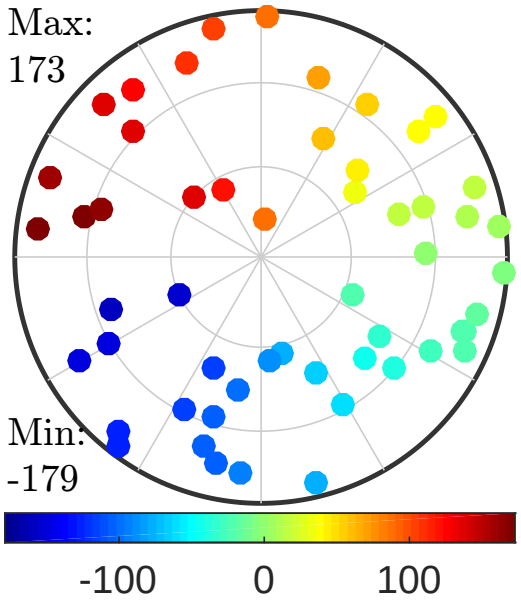
contour plots

```
plot(v,'contourf')
```

Data Plots

colorize vectors by value

```
v = vector3d.rand(100)
scatter(v,v.rho./degree,'upper',...
        'grid','on','minmax')
mtexColorbar('southoutside')
mtexColorMap jet
```



colorize by RGB triples

```
oM = ipdfHSVOrientationMapping
scatter(v,oM.Miller2color(v))
```

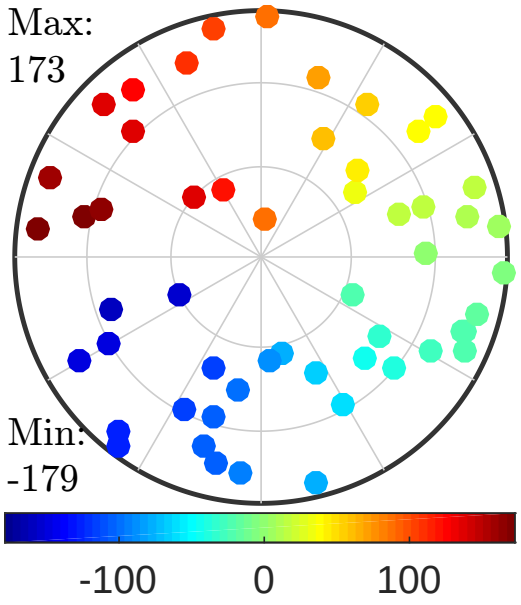
visualize directions

```
quiver(v,orth(v)) % a vector field
```

Data Plots

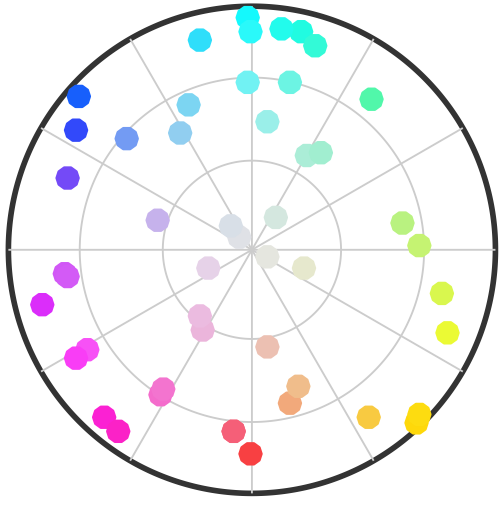
colorize vectors by value

```
v = vector3d.rand(100)
scatter(v,v.rho./degree,'upper',...
        'grid','on','minmax')
mtexColorbar('southoutside')
mtexColorMap jet
```



colorize by RGB triples

```
oM = ipdfHSVOrientationMapping
scatter(v,oM.Miller2color(v))
```



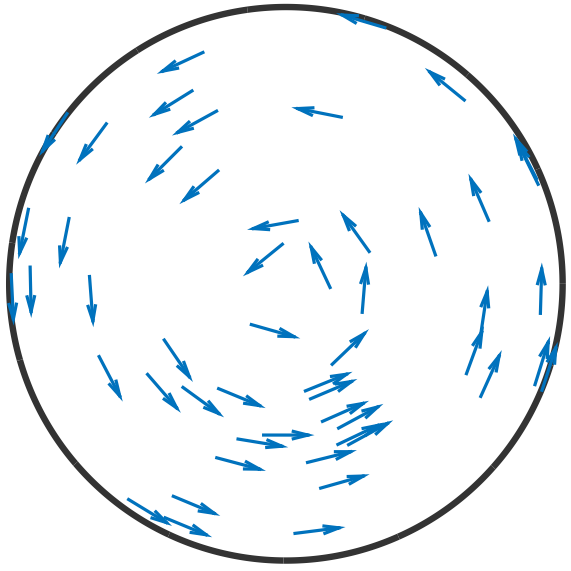
visualize directions

```
quiver(v,orth(v)) % a vector field
```

Data Plots

colorize vectors by value

```
v = vector3d.rand(100)
scatter(v,v.rho./degree,'upper',...
        'grid','on','minmax')
mtexColorbar('southoutside')
mtexColorMap jet
```



colorize by RGB triples

```
oM = ipdfHSVOrientationMapping
scatter(v,oM.Miller2color(v))
```

visualize directions

```
quiver(v,orth(v)) % a vector field
```

Customize Plots

General Syntax

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

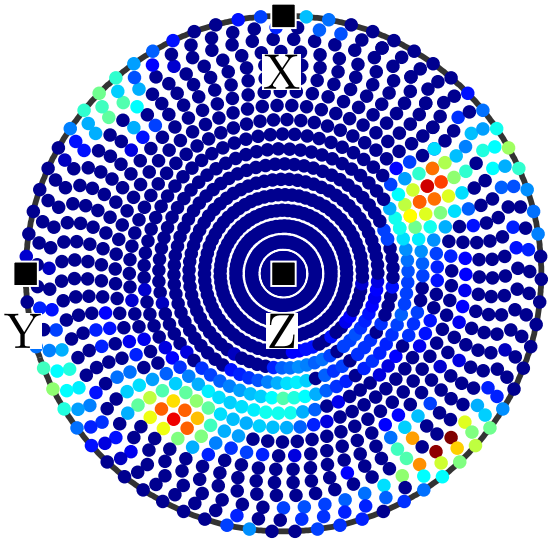
Options

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

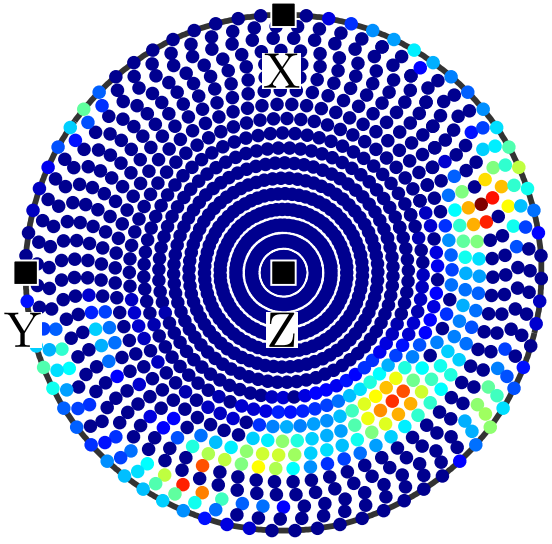
Example

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

(02–21)



(10–10)



Customize Plots

General Syntax

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

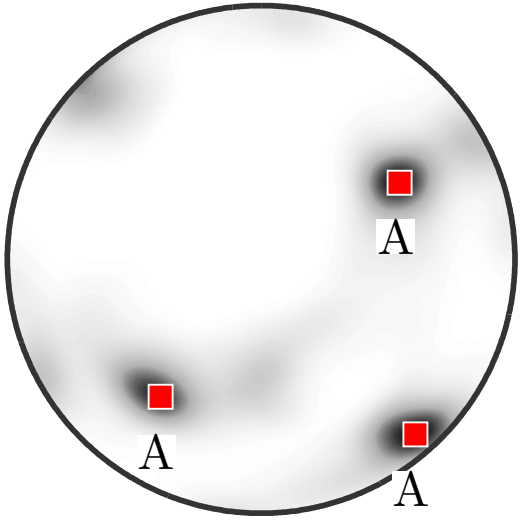
Options

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

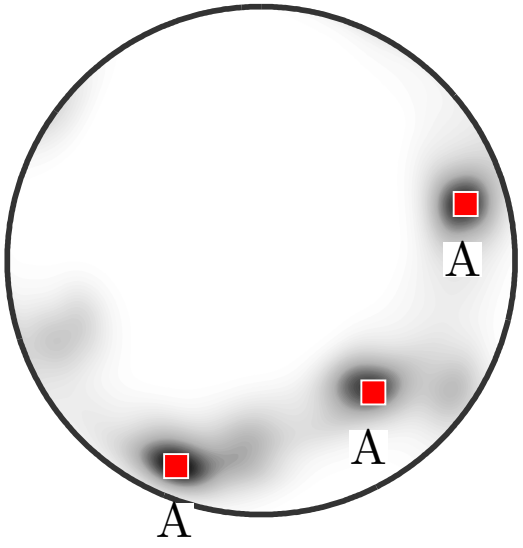
Example

```
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')
```

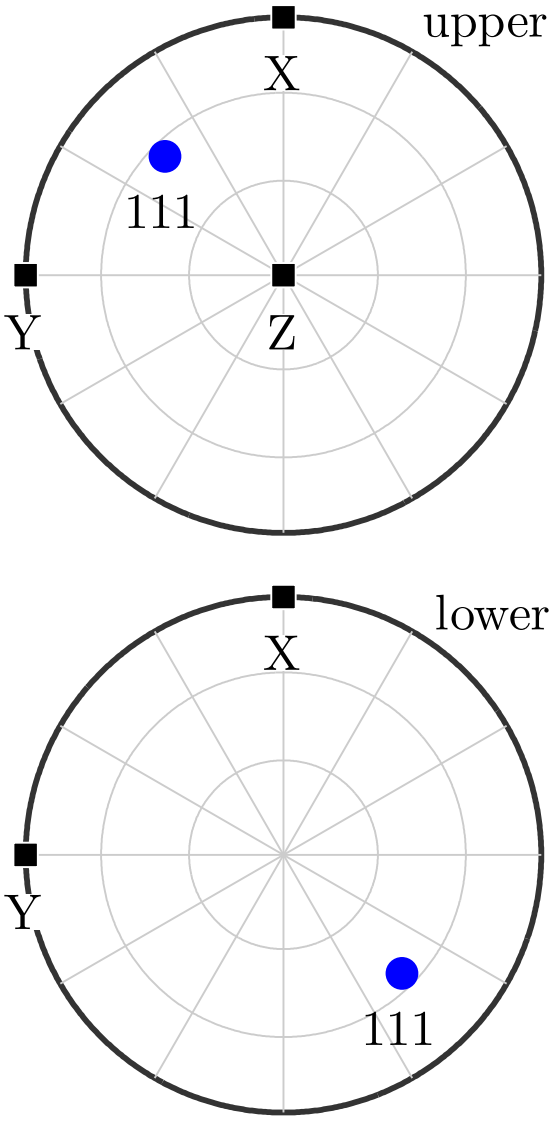
Then `r` and `-r` represent the same axis

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

The angle to an axis is always less then 90°

```
angle(r,-vector3d.X) / 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

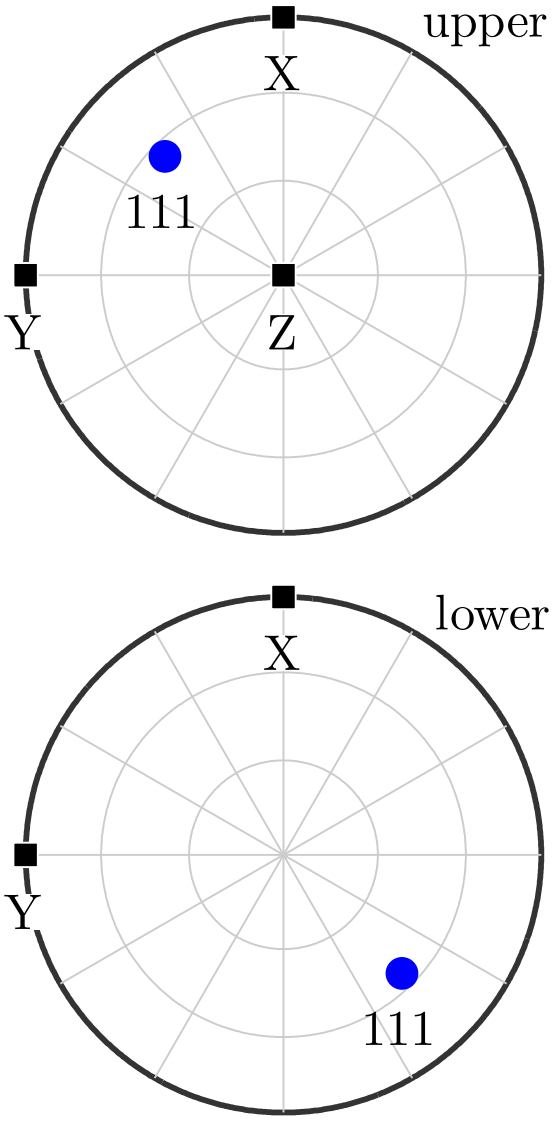
```
eq(r, -r)
```

```
1
```

The angle to an axis is always less then 90°

```
angle(r,-vector3d.X) / 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

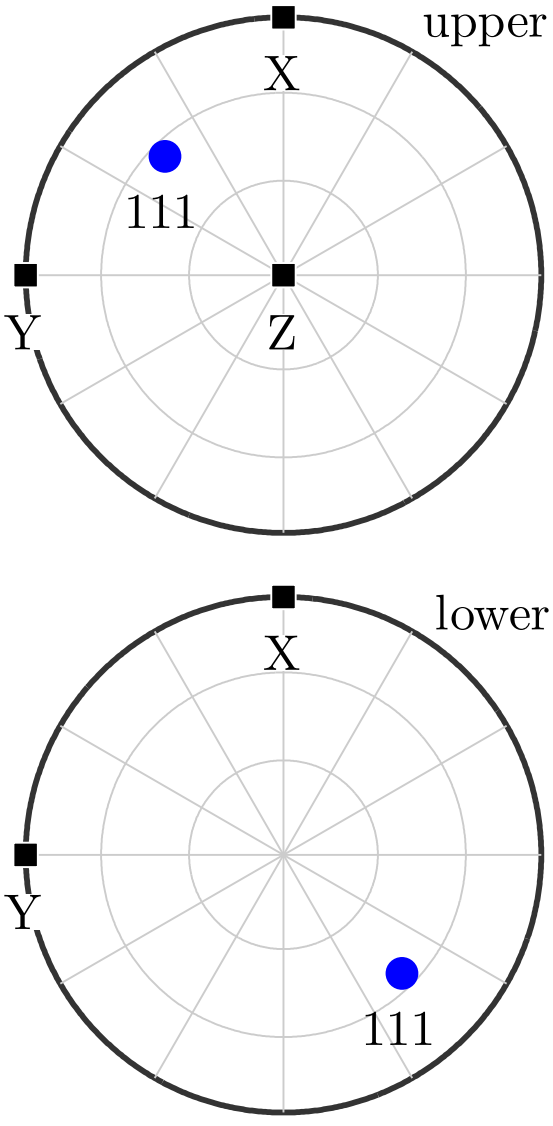
```
eq(r, -r)
```

The angle to an axis is always less then 90°

```
angle(r,-vector3d.X) / 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

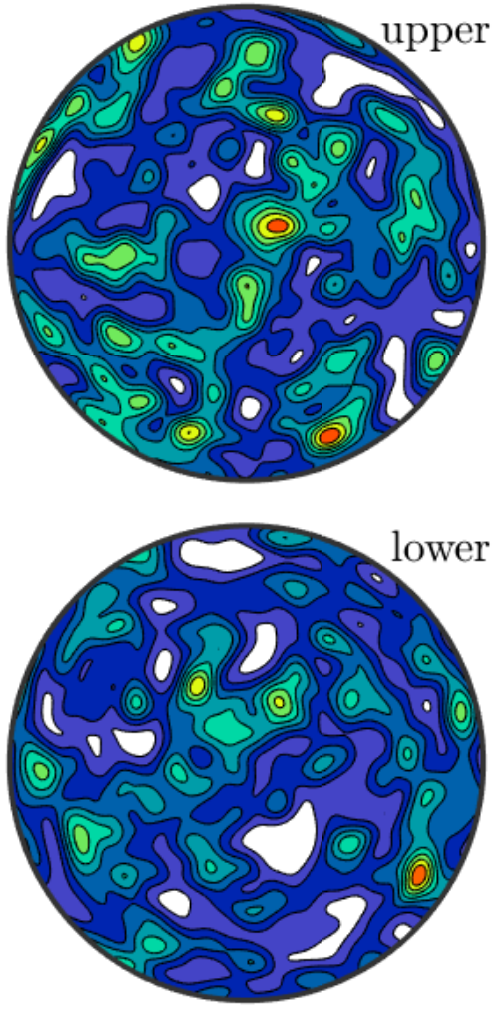
```
eq(r, -r)
```

The angle to an axis is always less then 90°

```
angle(r,-vector3d.X) / degree
```

The option **antipodal** in a contour plot

```
r = vector3d.rand(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

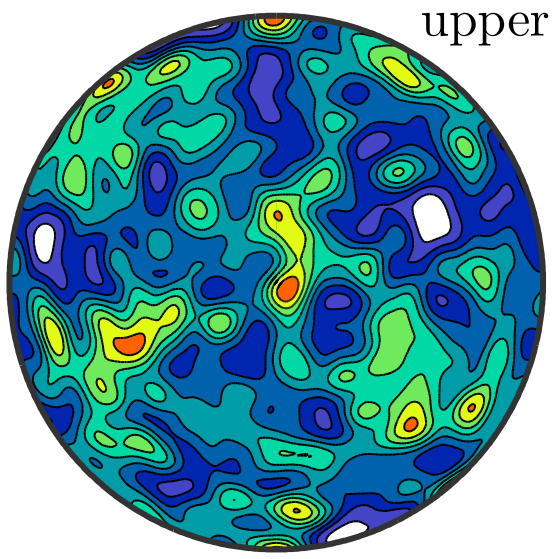
```
eq(r, -r)
```

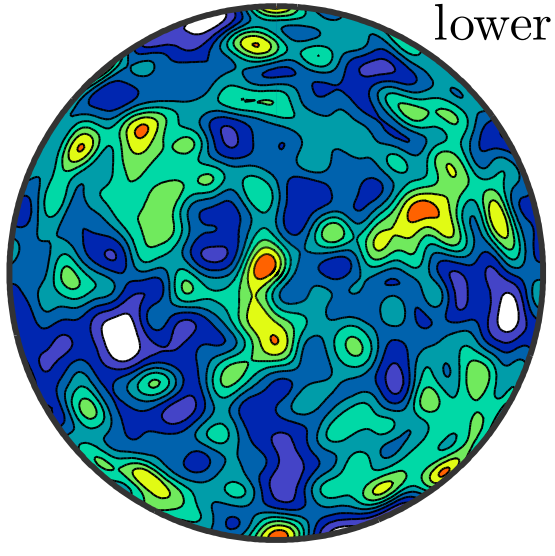
The angle to an axis is always less then 90°

```
angle(r,-vector3d.X) / 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',vector3d.X,'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',vector3d.X,'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',vector3d.X,'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',vector3d.X,'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)
```

Basic Calculations

rotate a vector

```
R = rotation('axis',vector3d.X,'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

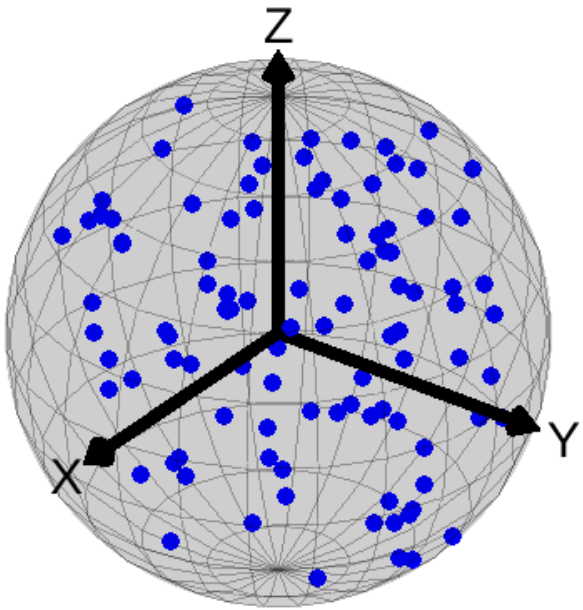
```
inv(R)
```

combine rotations

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

plotting

```
R = rotation.rand(100)
```



Basic Calculations

rotate a vector

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

the inverse rotation

```
inv(R)
```

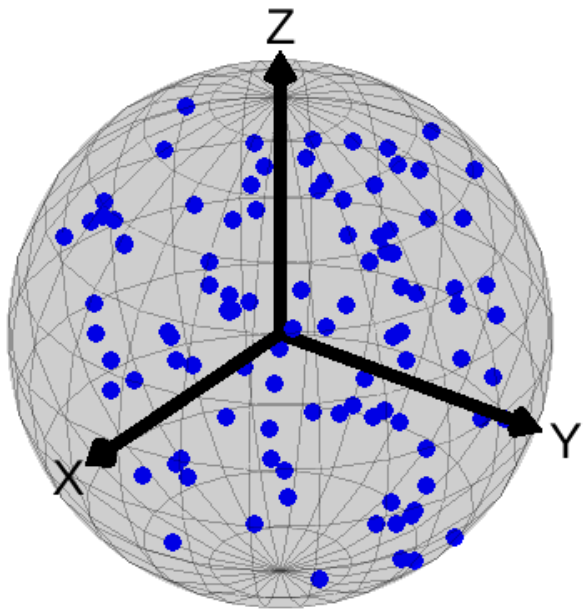
```
ans = rotation (show methods, plot)
  size: 1 x 1

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

combine rotations

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

plotting



Basic Calculations

rotate a vector

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

the inverse rotation

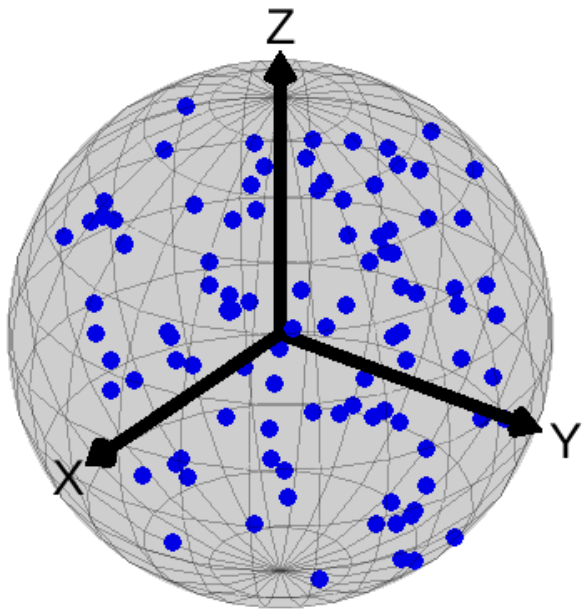
```
inv(R)
```

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
```



plotting

Basic Calculations

rotate a vector

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

the inverse rotation

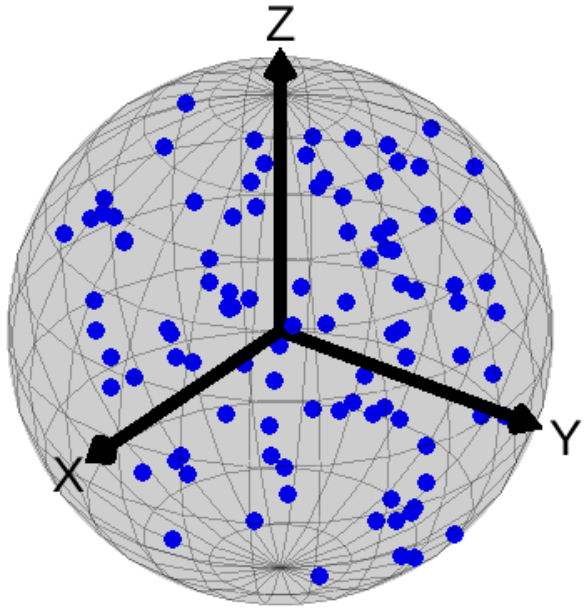
```
inv(R)
```

combine rotations

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

plotting

```
R = rotation.rand(100)
scatter(R)          % axis angle plot
```



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(vector3d.X + vector3d.Y)
```

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(vector3d.X + vector3d.Y)
```

```
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(vector3d.X + vector3d.Y)
```

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(vector3d.X + vector3d.Y)
```

angles between proper and improper rotations

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

check for improper rotations

```
I.isImproper
```

```
  1
```

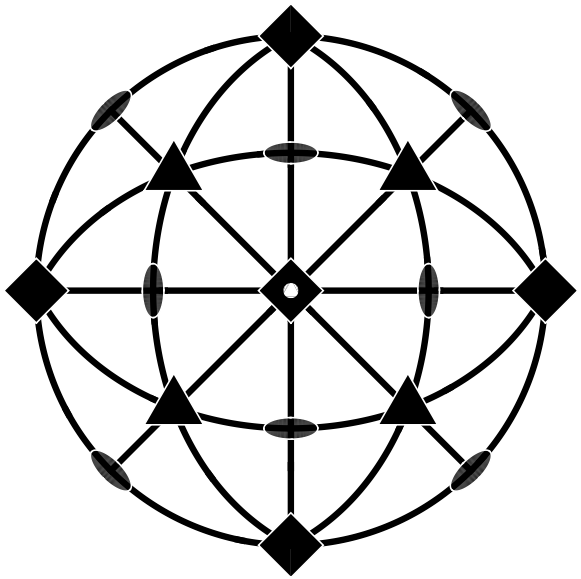
Crystal Symmetry

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.

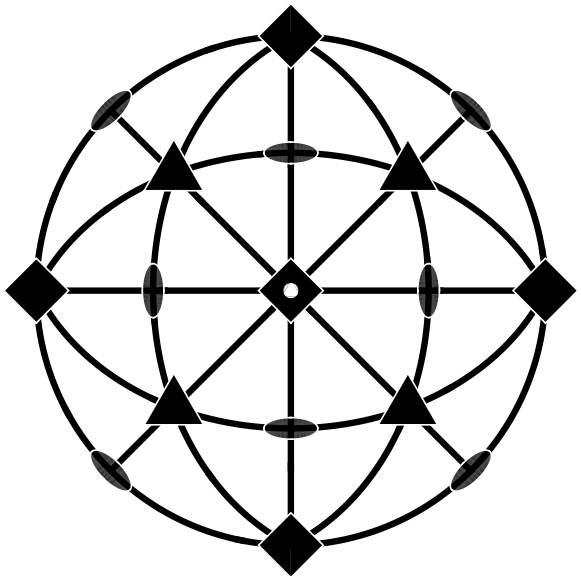
```
CS = crystalSymmetry('m-3m')
```

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
```



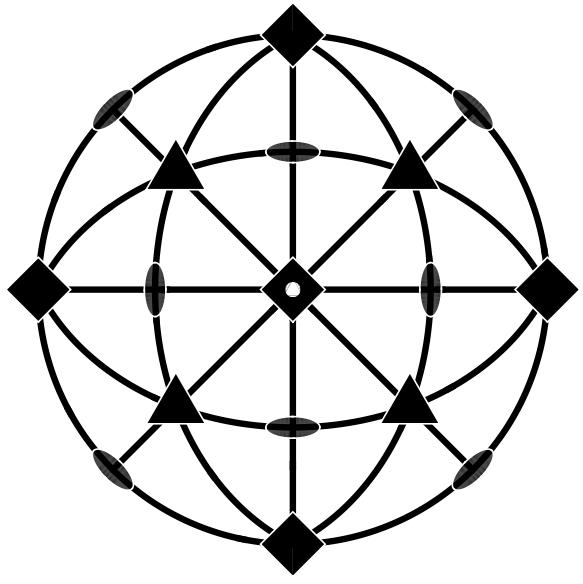
import data from crystal information files

```
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

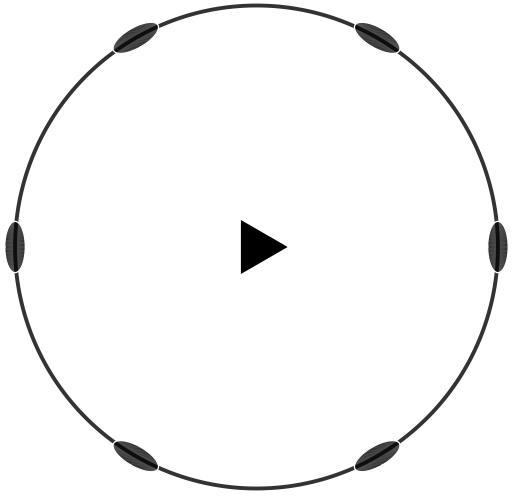
```
rotation(crystalSymmetry('222'))
```

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*
```

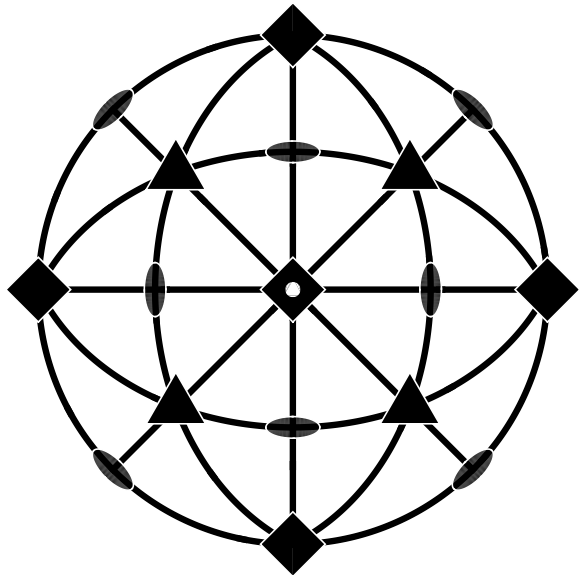


```
CS = loadPHL('minerals.phl')
```

Crystal Symmetry

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

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



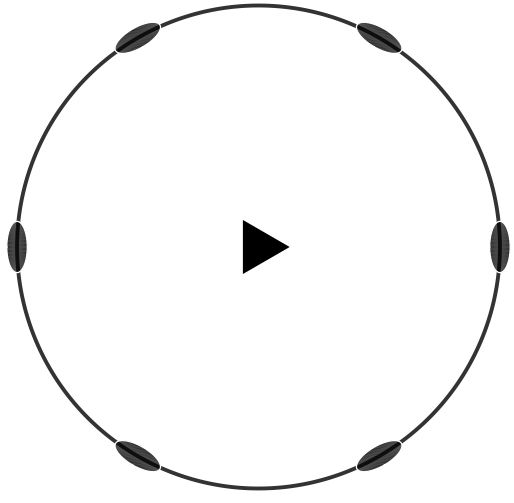
extract the rotations of a point group

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

import data from crystal information files

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

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



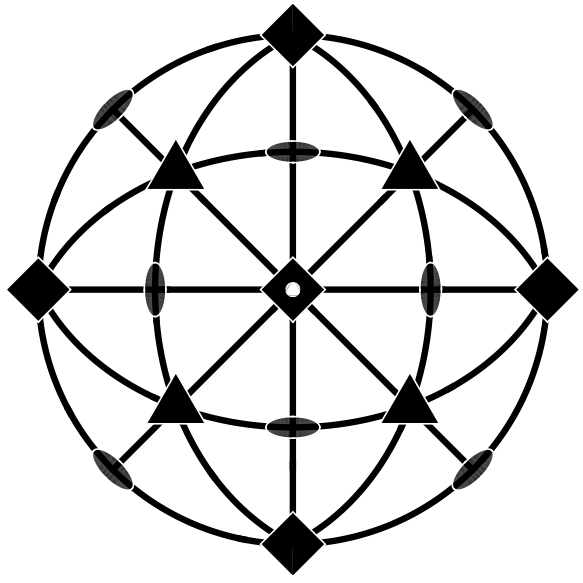
```
CS{1} = crystalSymmetry (show methods, plot)

  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')
```



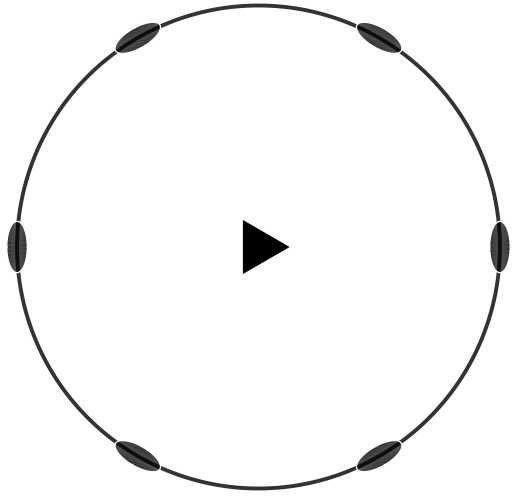
extract the rotations of a point group

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

import data from crystal information files

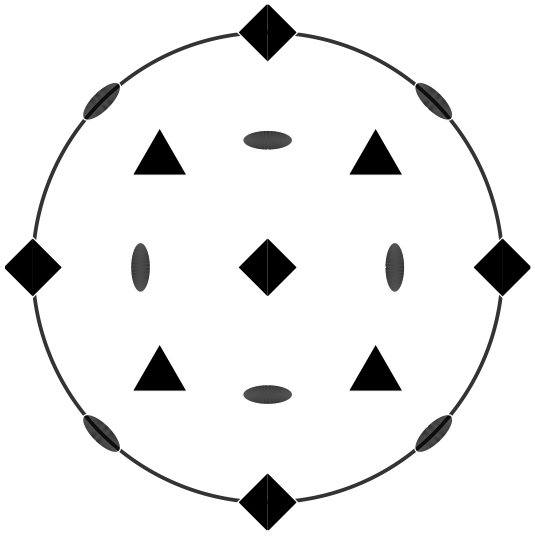
```
CS = loadCIF('Quarz.cif')
```

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

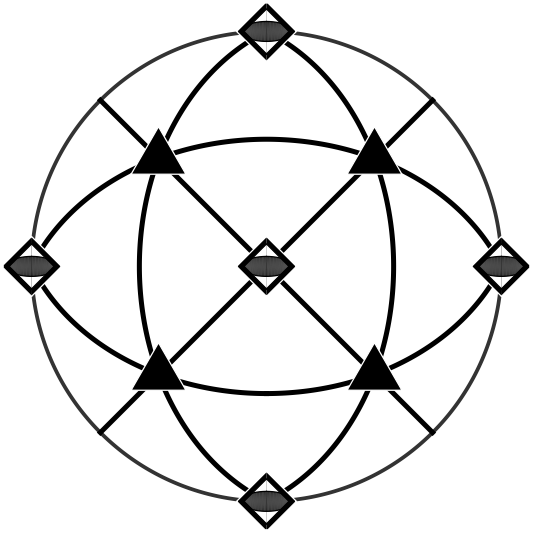


switch to Laue / purely rotational group

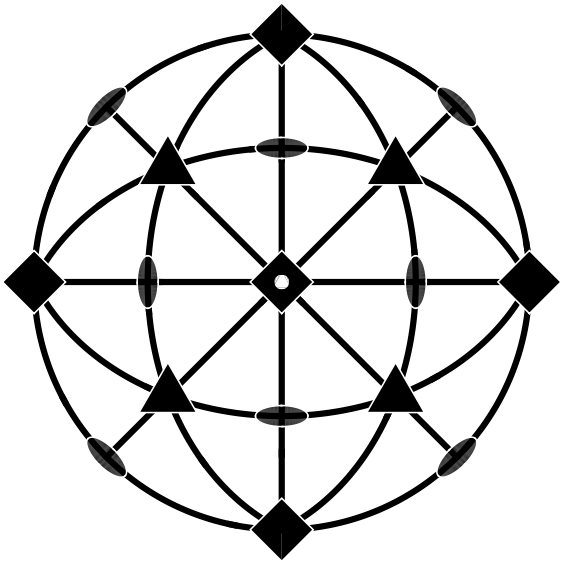
```
CS.Laue
CS.properGroup
```



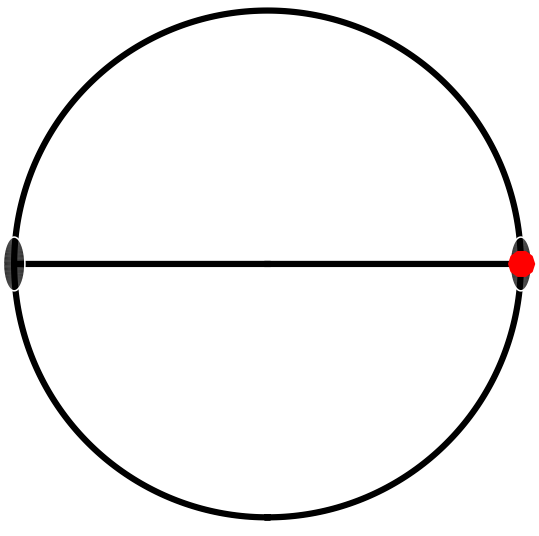
432



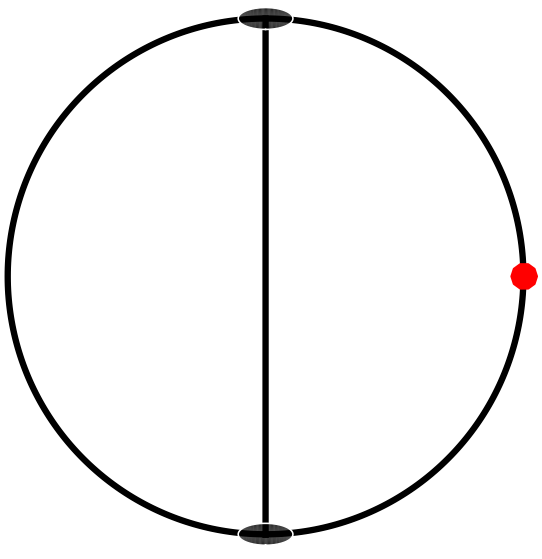
$\bar{4}3m$



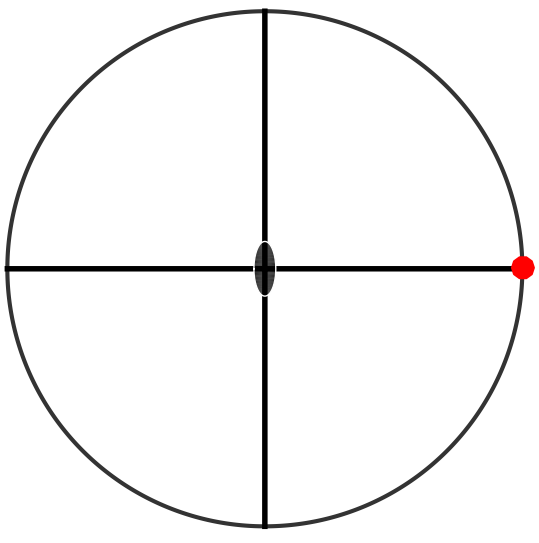
$m\bar{3}m$



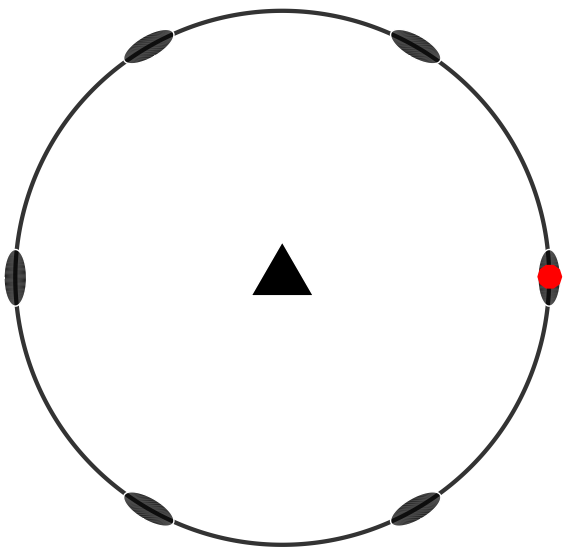
$2mm$



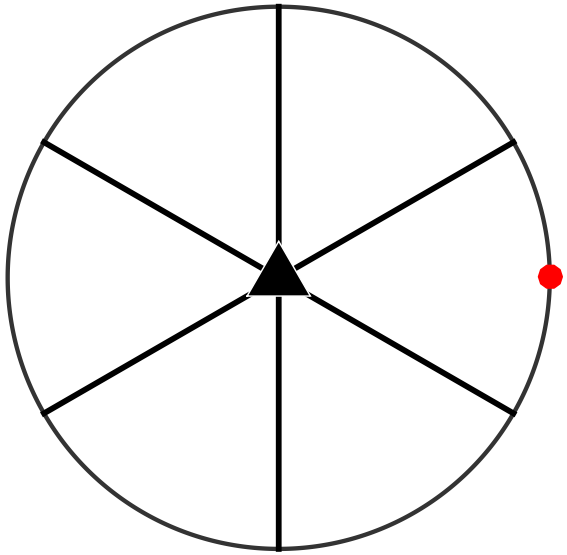
$m2m$



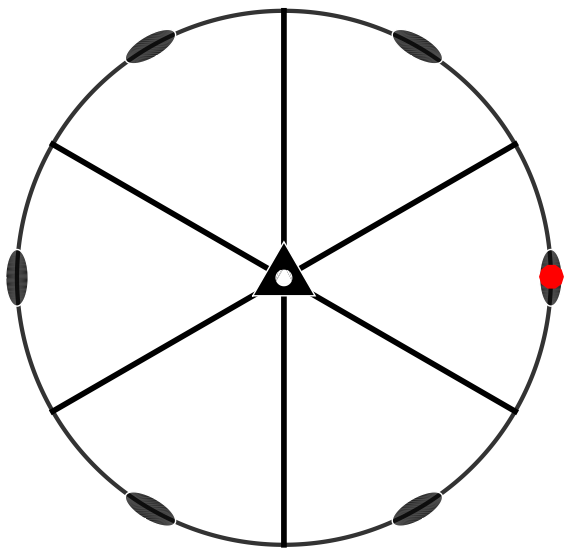
$mm2$



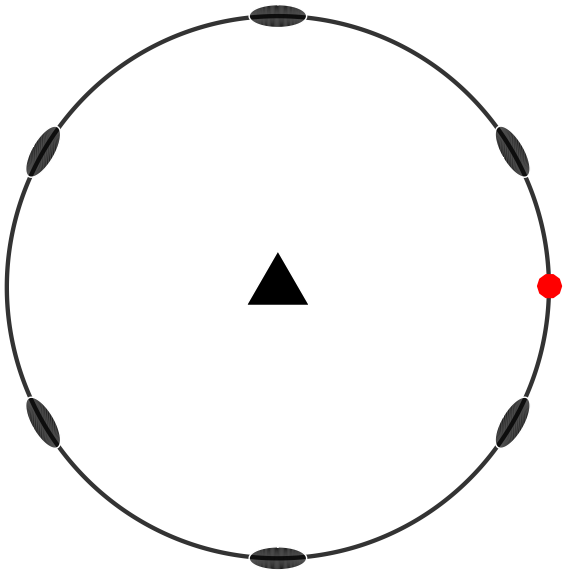
321



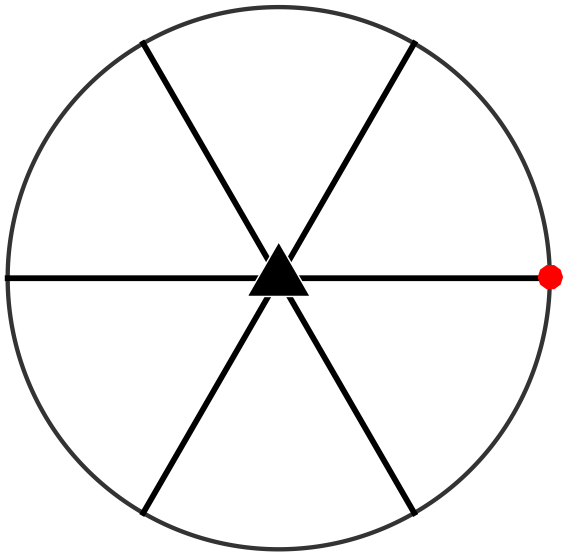
$3m1$



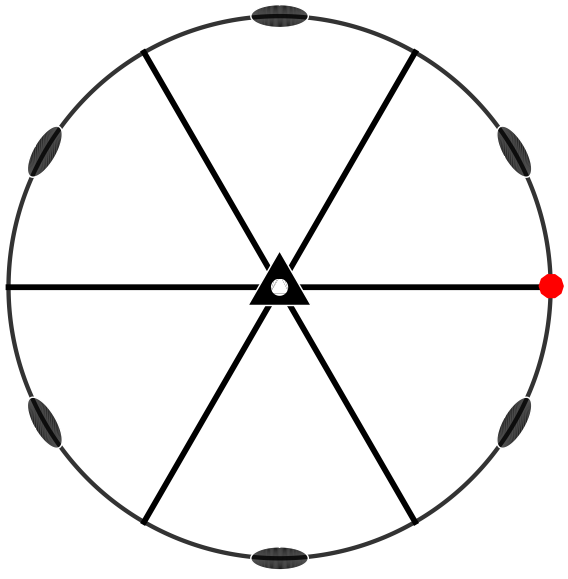
$\bar{3}m1$



312



$31m$



$\bar{3}1m$

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])
```

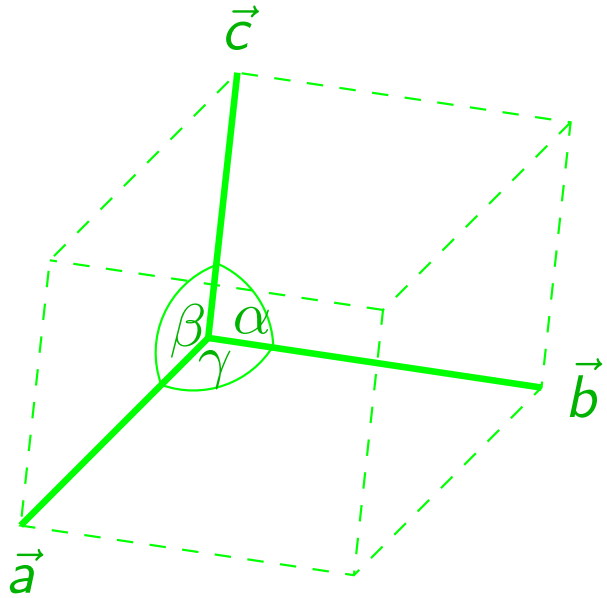
The axes of the reciprocal lattice are defined orthogonal to $\vec{a}, \vec{b}, \vec{c}$, i.e.

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

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])
```

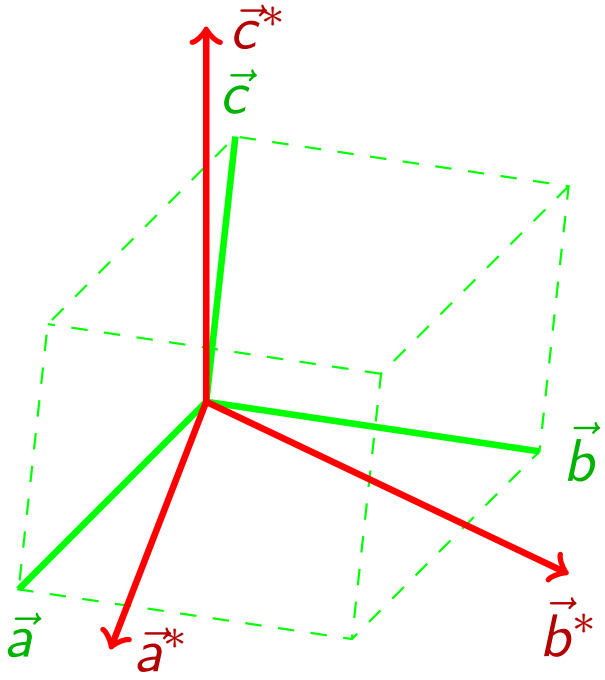
The axes of the reciprocal lattice are defined orthogonal to $\vec{a}, \vec{b}, \vec{c}$, i.e.

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

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.



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])
```

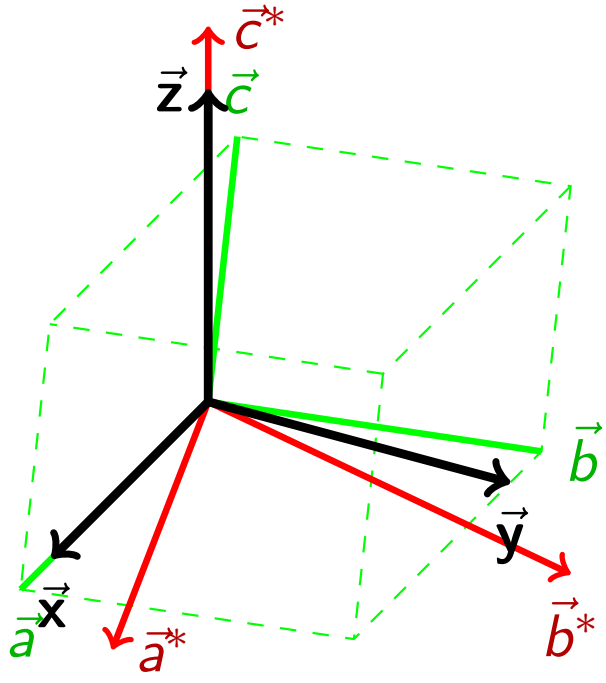
The axes of the reciprocal lattice are defined orthogonal to $\vec{a}, \vec{b}, \vec{c}$, i.e.

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

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.




```
CS = 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])
```

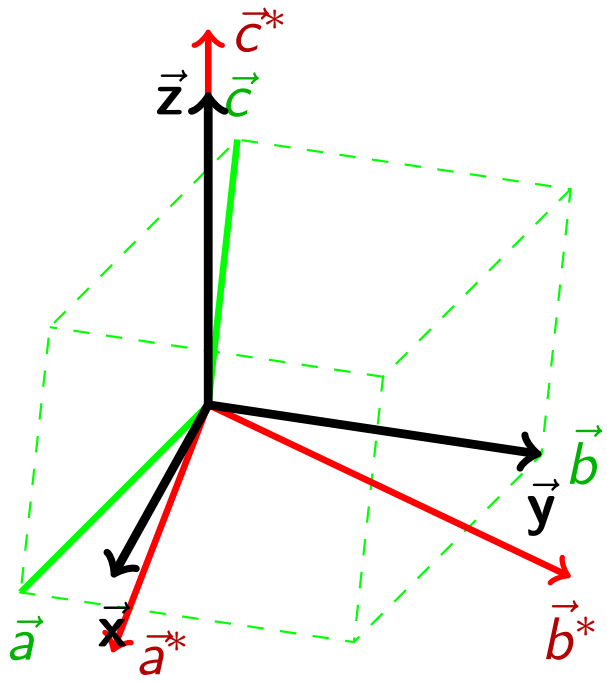
The axes of the reciprocal lattice are defined orthogonal to $\vec{a}, \vec{b}, \vec{c}$, i.e.

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

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.




```
CS = 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**

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

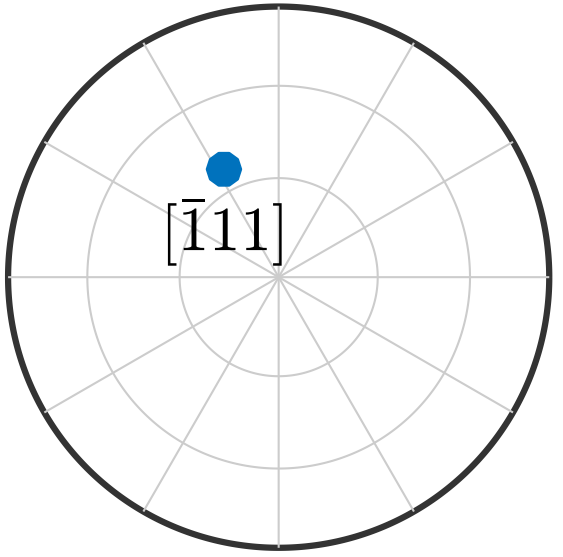
Miller Indices - Crystal Fixed Directions

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 = crystalSymmetry('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
```



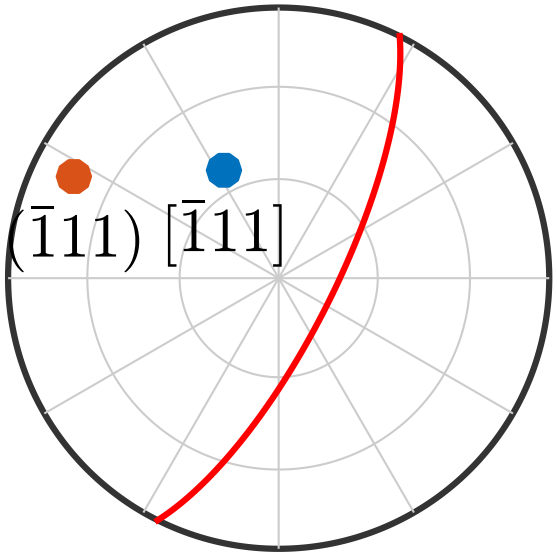

```
plot(m,'labeled')
```

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



A direction in reciprocal coordinates

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

```
n  = Miller(-1,1,1,CS,'hkl')
```

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

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

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

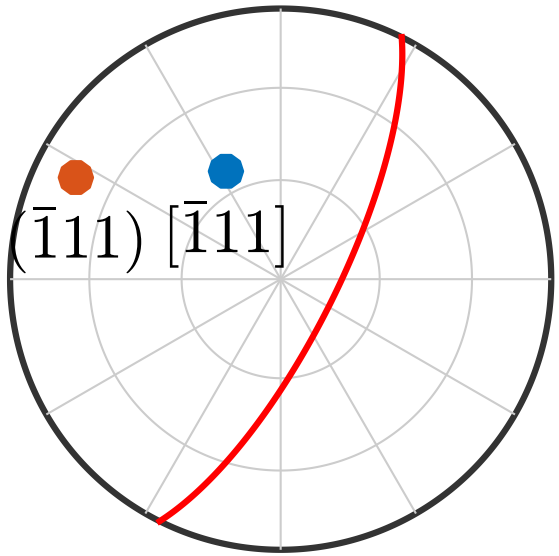
```
plot(n,'plane')
```

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



A direction in reciprocal coordinates

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

```
n  = Miller(-1,1,1,CS,'hkl')
```

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

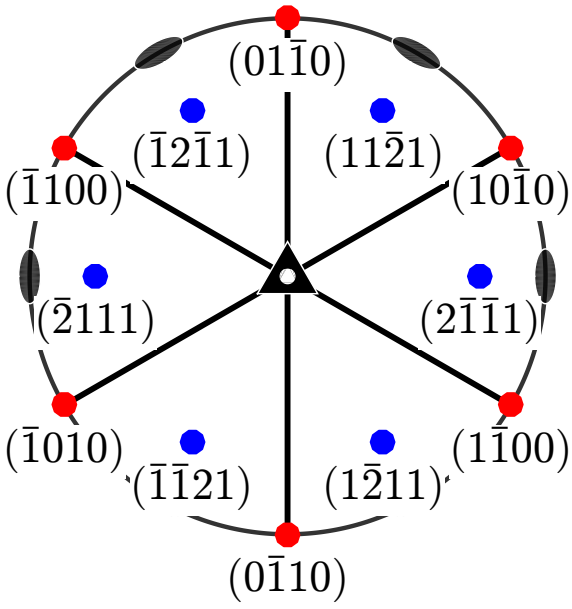
A direction in the orthogonal coordinate system

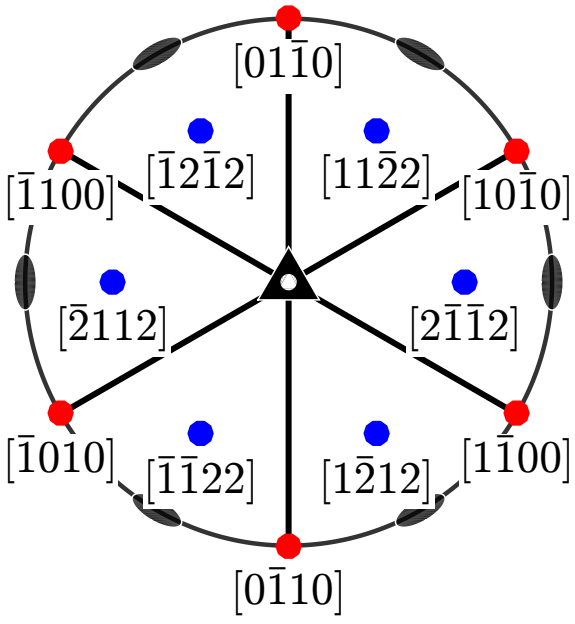
```
m = Miller(vector3d.X,CS)
```

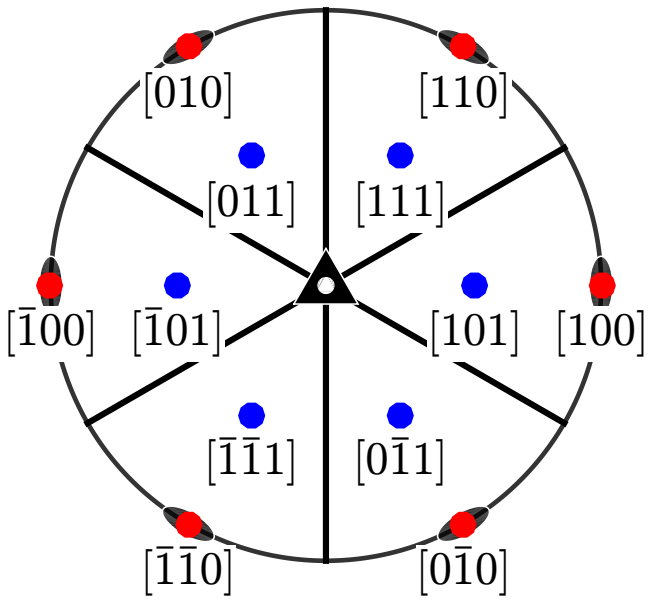
```
hold all
plot(n,'labeled')
```

```
plot(n,'plane')
```

Trigonal and Hexagonal Symmetries







define lists of crystal directions

```
m = Miller({1 0 -1 0},{1 1 -2 1},CS,'hkil')
m = Miller({1 0 -1 0},{1 1 -2 2},CS,'UVTW')
m = Miller({1 0 0},{1 0 1},CS,'uvw')
```

Plot all symmetrically equivalent directions

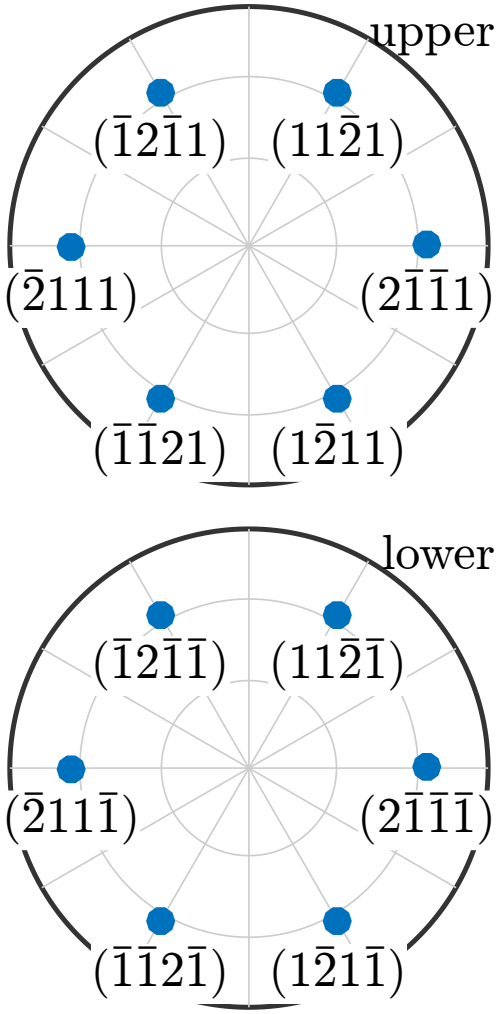
```
plot(m,'symmetrised','labeled')
```

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)
```

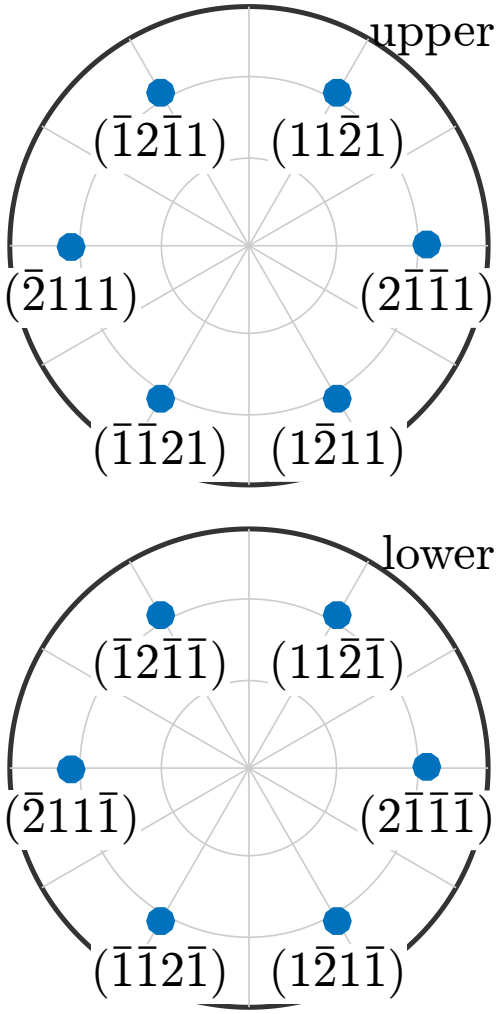
Access the coordinates and properties

```
m.U, m.hkl, m.uvw, m.UVTW
```

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)
```

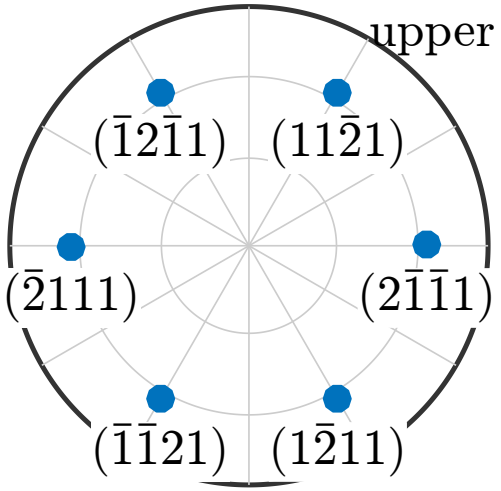
Access the coordinates and properties

```
m.U, m.hkl, m.uvw, m.UVTW
```

Calculating with Crystal Directions

Find all symmetrically equivalent directions

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

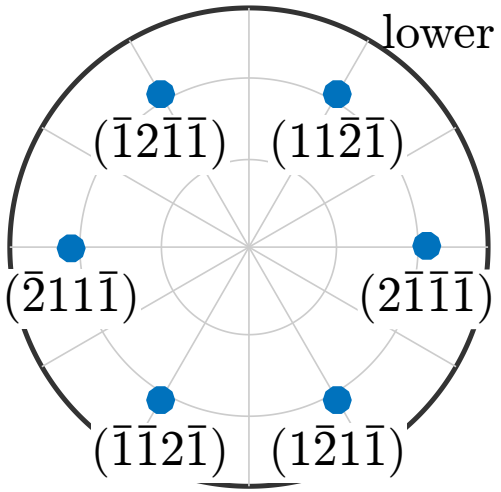


Change from reciprocal coordinate system to $\vec{a}$, $\vec{b}$, $\vec{c}$

```
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

```
m.U, m.hkl, m.uvw, m.UVTW
```

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'
```

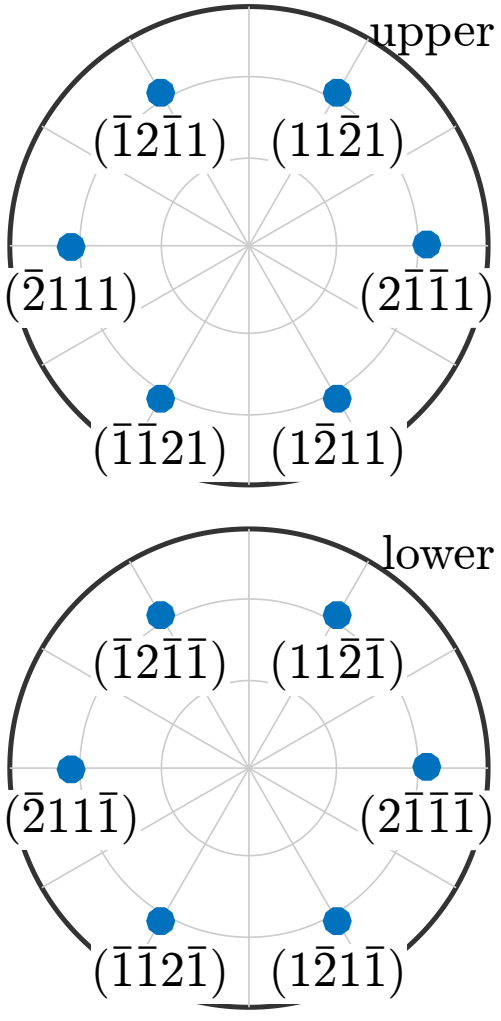
```
round(m)
```

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'
```

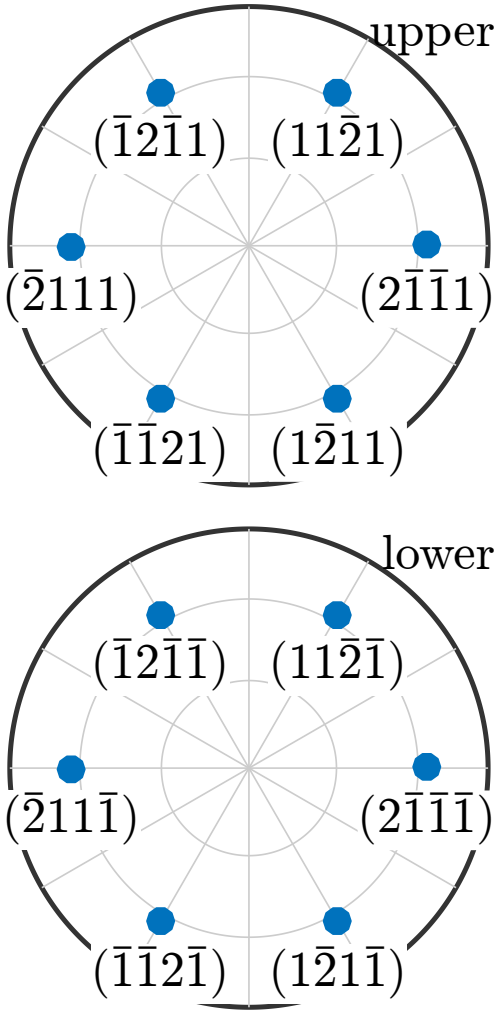
```
round(m)
```

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.,

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

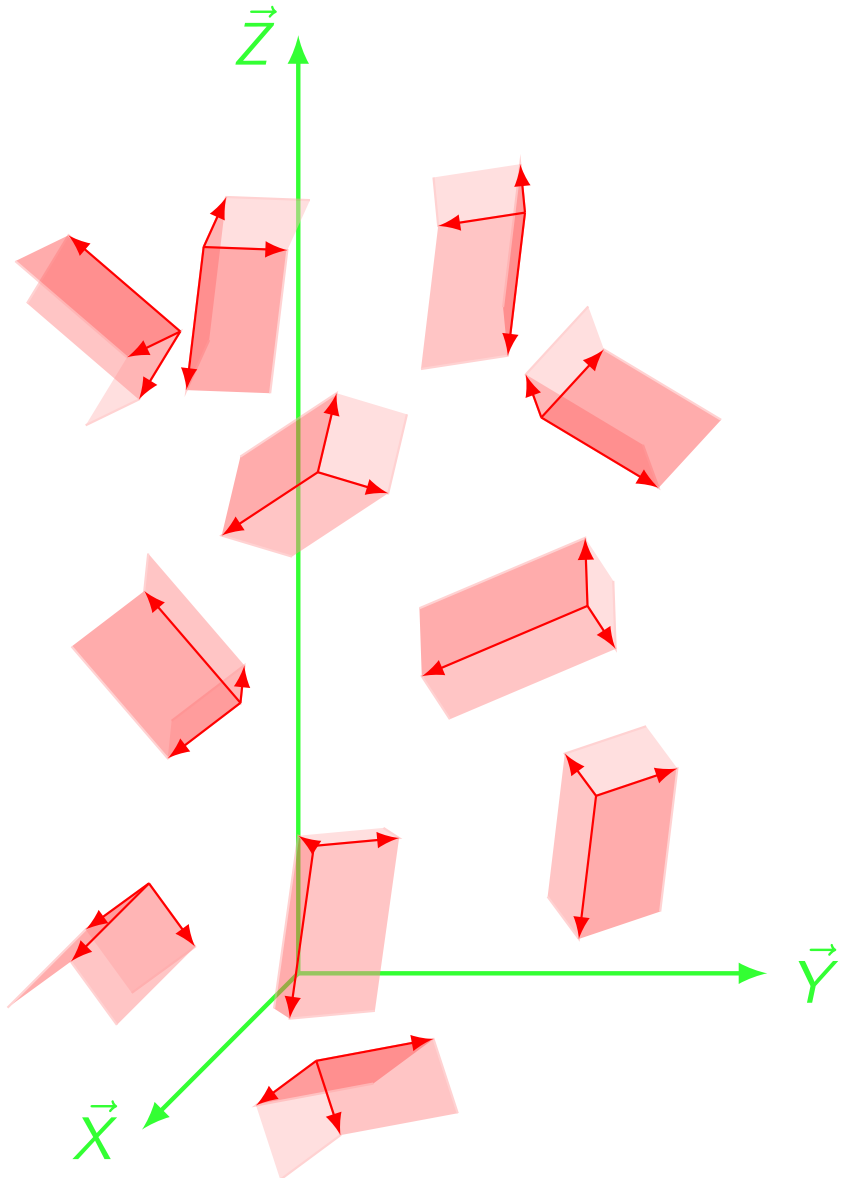
The coordinate transform $\mathbf{O}$ with

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

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.,

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

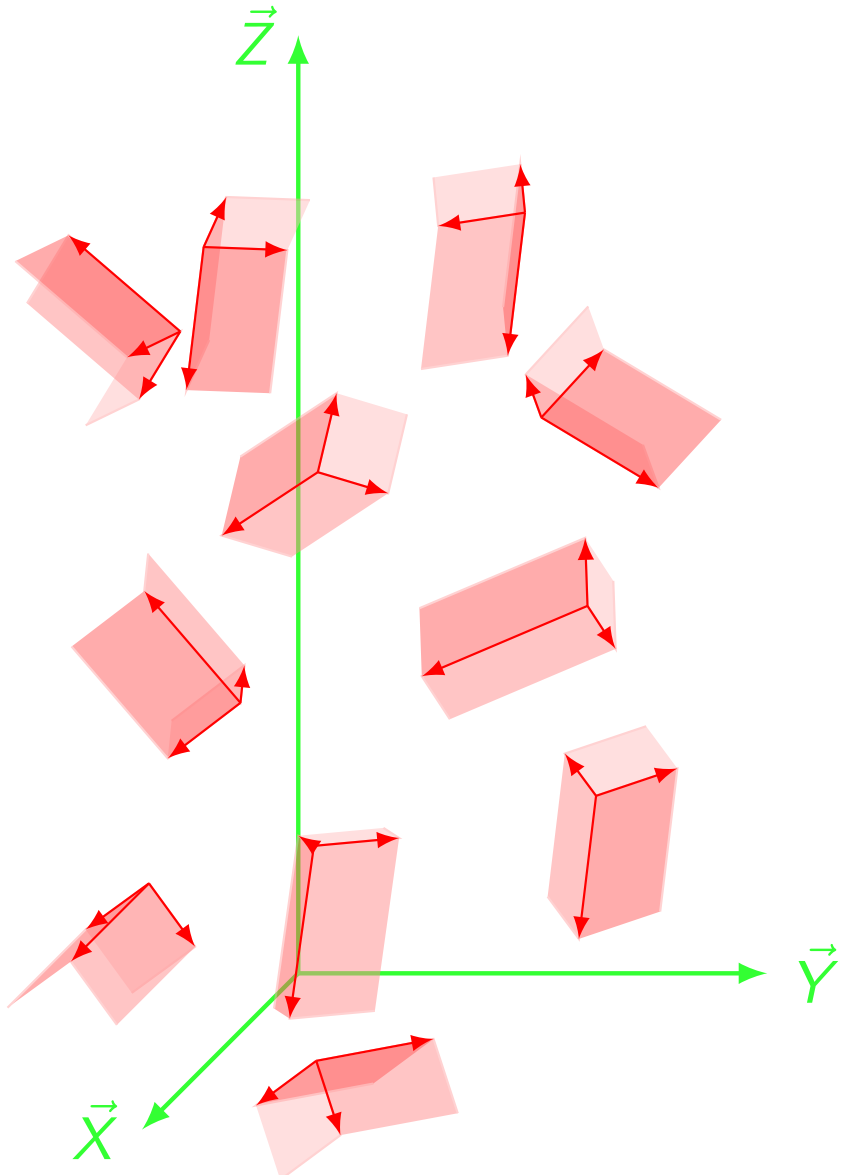
The coordinate transform $\mathbf{O}$ with

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

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.,

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

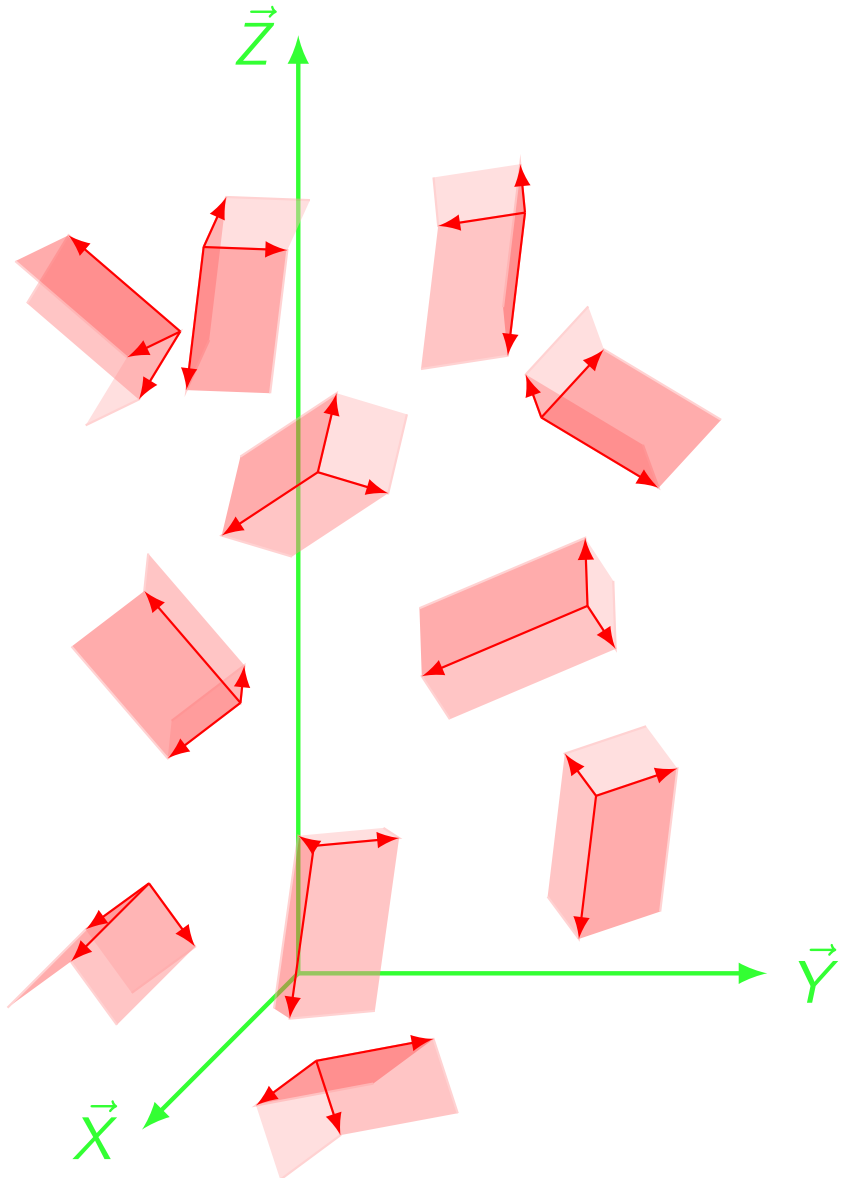
The coordinate transform $\mathbf{O}$ with

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

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.,

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

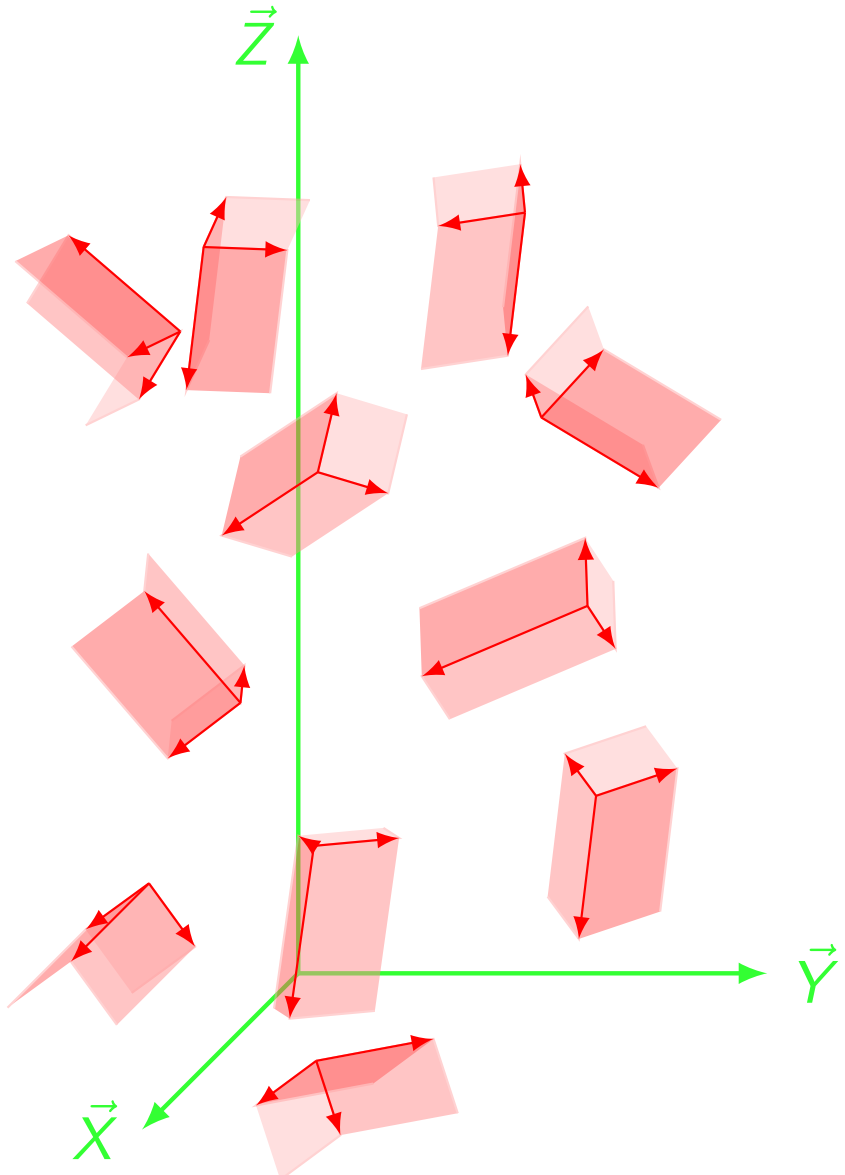
The coordinate transform $\mathbf{O}$ with

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

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}, {0 1 0}, O.CS)
plot(symmetrise(O) * h(1))
plot(O * symmetrise(h(1)))
```

```
plotPDF(O,h)
```

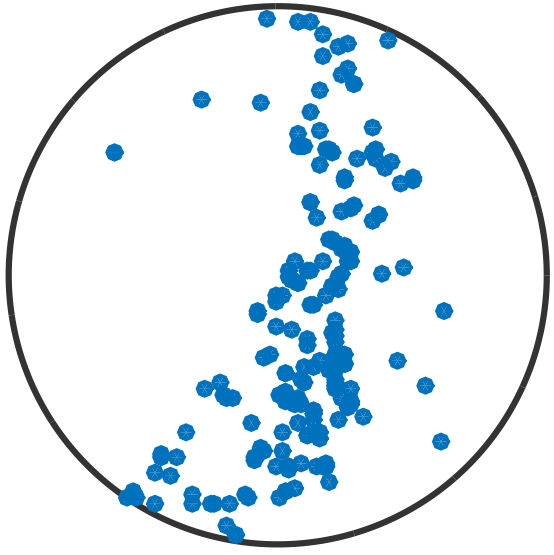
```
plotPDF(O,h,'contourf')
```

specimen directions in crystal coordinates

```
plot(symmetrise(inv(O)) * vector3d.X)
plot(symmetrise(inv(O) * vector3d.X))
```

```
v = [vector3d.X, vector3d.Y]
plotIPDF(O,v)
```

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



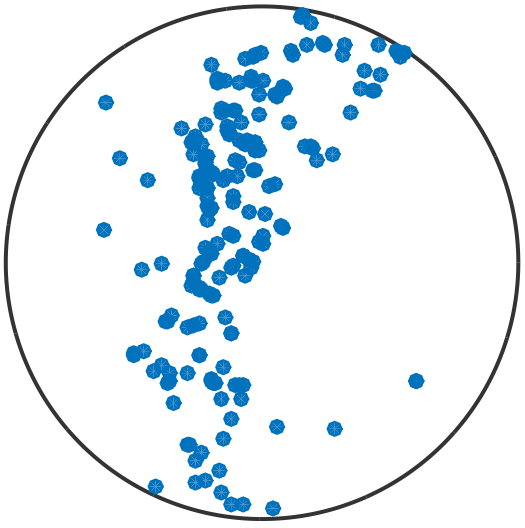
Pole Figures and Inverse Pole Figures

Lattice planes in specimen coordinates

```
h = Miller({1 0 0}, {0 1 0}, O.CS)
plot(symmetrise(O) * h(1))
plot(O * symmetrise(h(1)))
```

```
plotPDF(O,h)
```

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

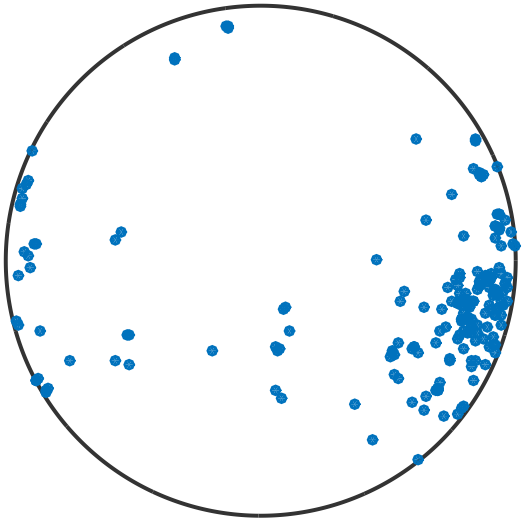


specimen directions in crystal coordinates

```
plot(symmetrise(inv(O)) * vector3d.X)
plot(symmetrise(inv(O) * vector3d.X))
```

```
v = [vector3d.X, vector3d.Y]
plotIPDF(O,v)
```

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



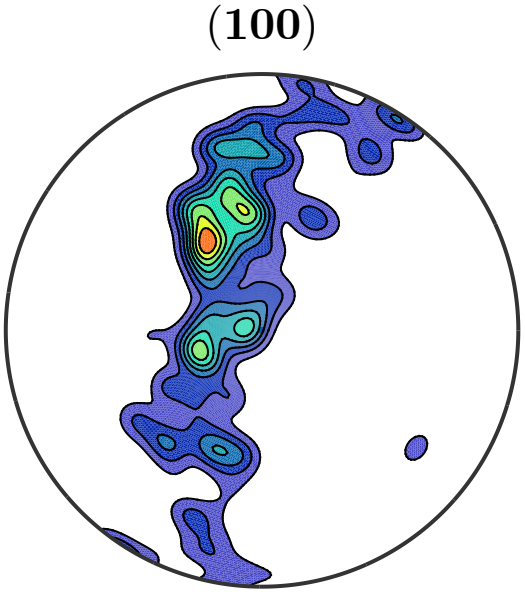
Pole Figures and Inverse Pole Figures

Lattice planes in specimen coordinates

```
h = Miller({1 0 0}, {0 1 0}, O.CS)
plot(symmetrise(O) * h(1))
plot(O * symmetrise(h(1)))
```

```
plotPDF(O,h)
```

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

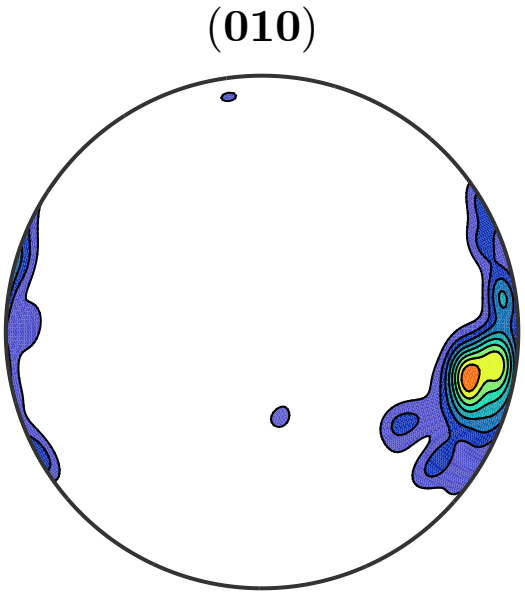


specimen directions in crystal coordinates

```
plot(symmetrise(inv(O)) * vector3d.X)
plot(symmetrise(inv(O) * vector3d.X))
```

```
v = [vector3d.X, vector3d.Y]
plotIPDF(O,v)
```

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



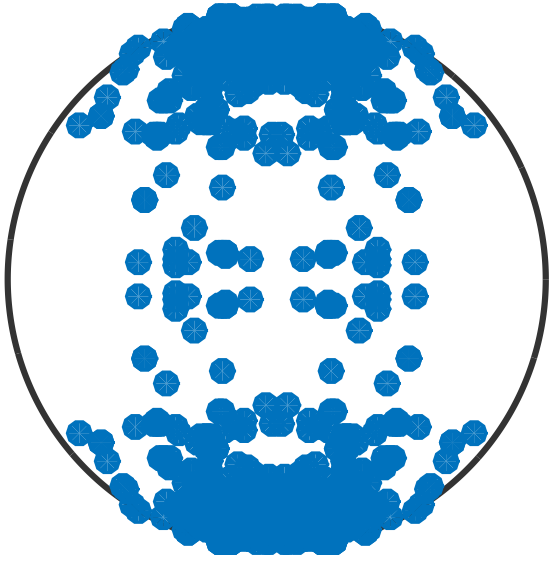
Pole Figures and Inverse Pole Figures

Lattice planes in specimen coordinates

```
h = Miller({1 0 0}, {0 1 0}, O.CS)
plot(symmetrise(O) * h(1))
plot(O * symmetrise(h(1)))
```

```
plotPDF(O,h)
```

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



specimen directions in crystal coordinates

```
plot(symmetrise(inv(O)) * vector3d.X)
plot(symmetrise(inv(O) * vector3d.X))
```

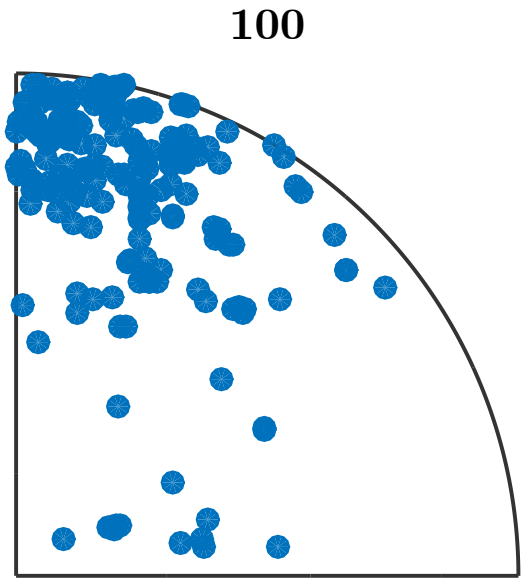
```
v = [vector3d.X, vector3d.Y]
plotIPDF(O,v)
```

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

Pole Figures and Inverse Pole Figures

Lattice planes in specimen coordinates

```
h = Miller({1 0 0}, {0 1 0}, O.CS)
plot(symmetrise(O) * h(1))
plot(O * symmetrise(h(1)))
```

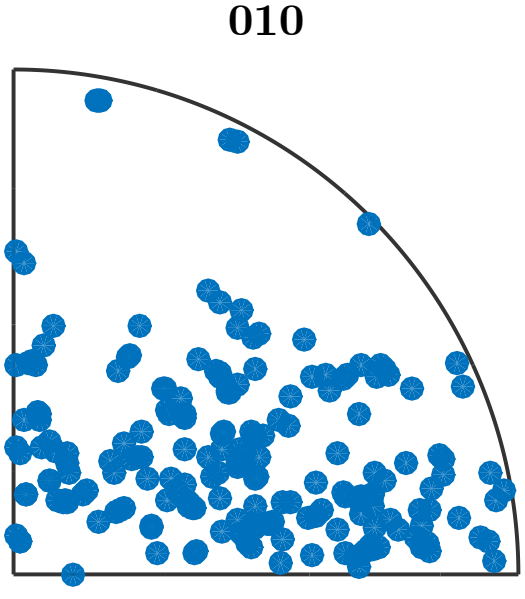



```
plotPDF(O,h)
```

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

specimen directions in crystal coordinates

```
plot(symmetrise(inv(O)) * vector3d.X)
plot(symmetrise(inv(O) * vector3d.X))
```




```
v = [vector3d.X, vector3d.Y]
plotIPDF(O,v)
```

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

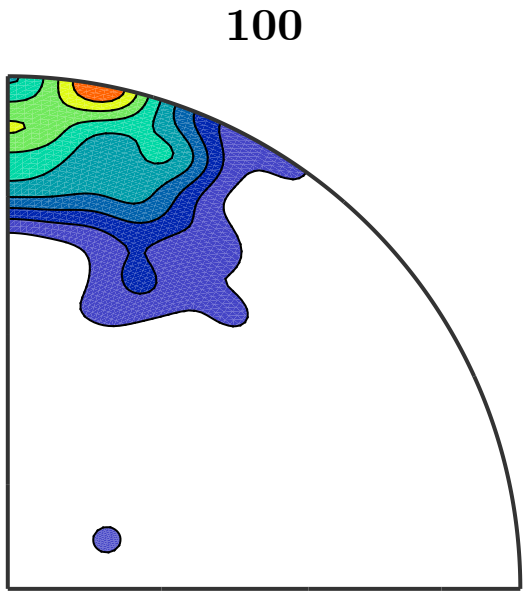
Pole Figures and Inverse Pole Figures

Lattice planes in specimen coordinates

```
h = Miller({1 0 0}, {0 1 0}, O.CS)
plot(symmetrise(O) * h(1))
plot(O * symmetrise(h(1)))
```

```
plotPDF(O,h)
```

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

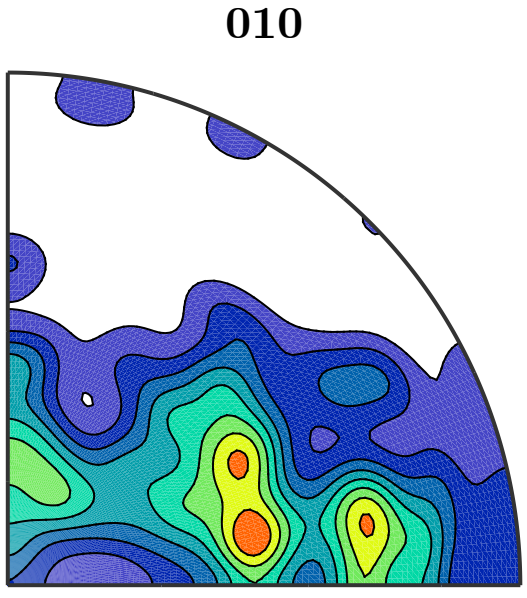


specimen directions in crystal coordinates

```
plot(symmetrise(inv(O)) * vector3d.X)
plot(symmetrise(inv(O) * vector3d.X))
```

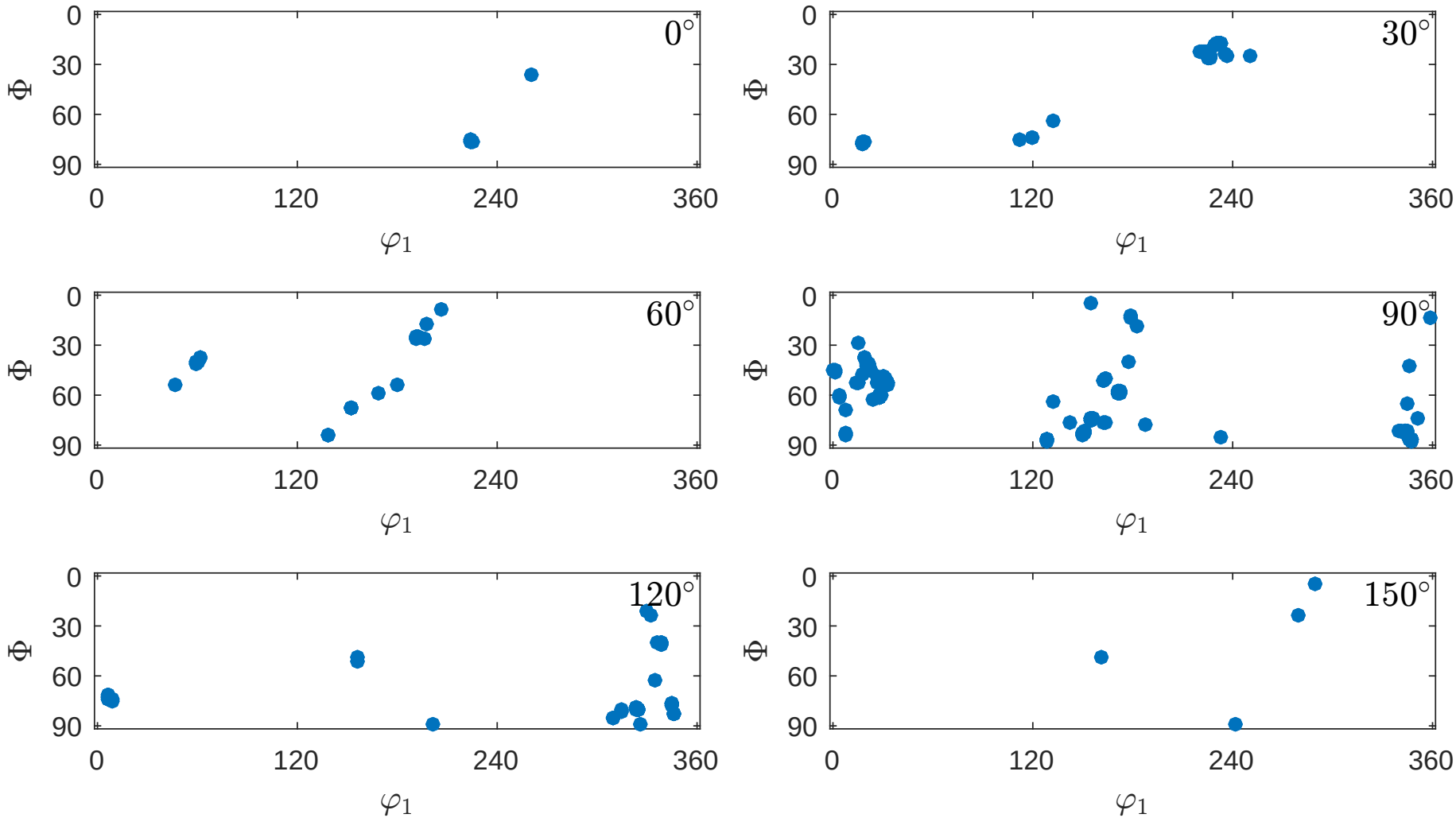
```
v = [vector3d.X, vector3d.Y]
plotIPDF(O,v)
```

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



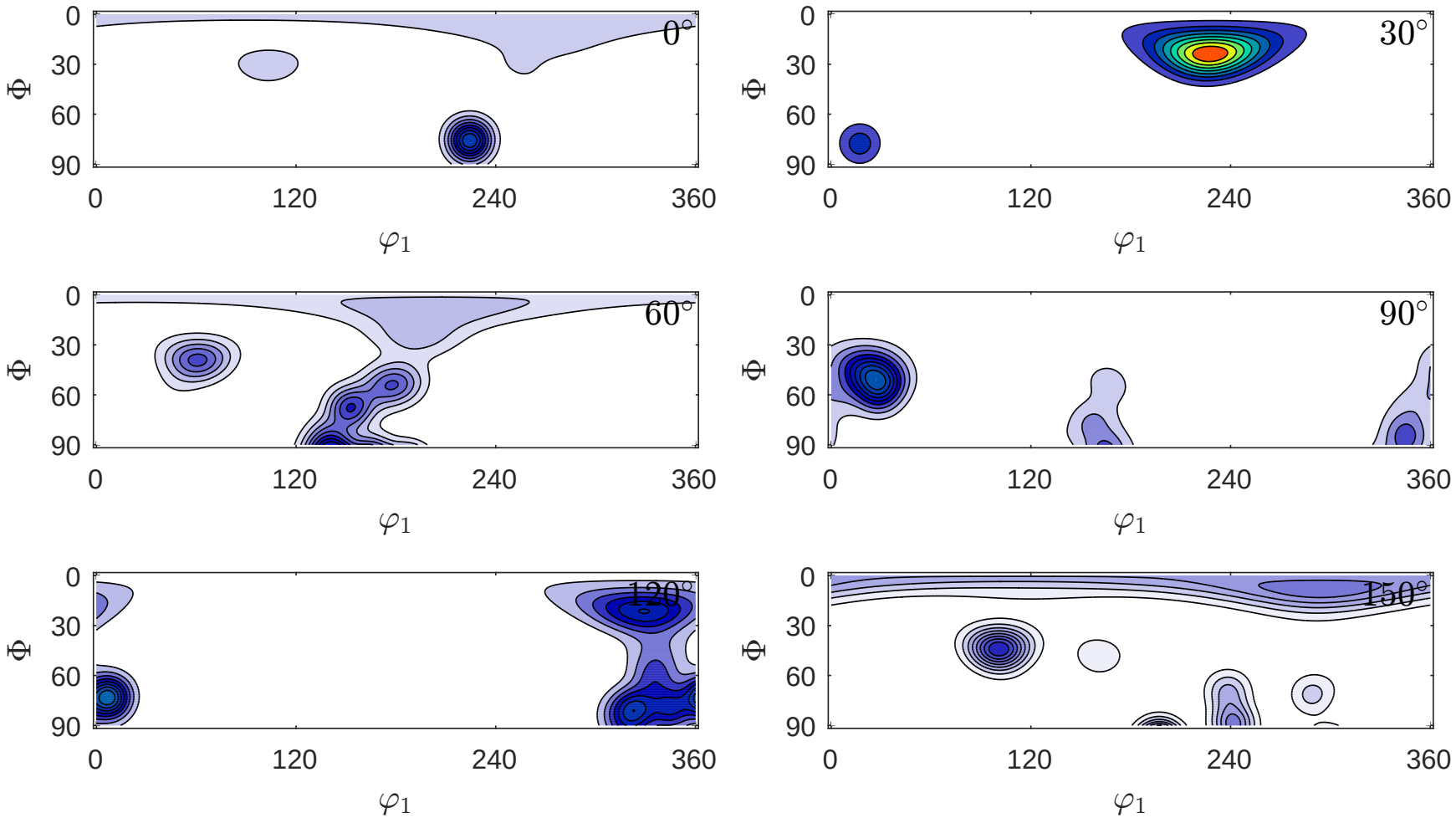
Plotting in Orientation Space

```
plotSection(O,'phi2',(0:30:150)*degree)
```



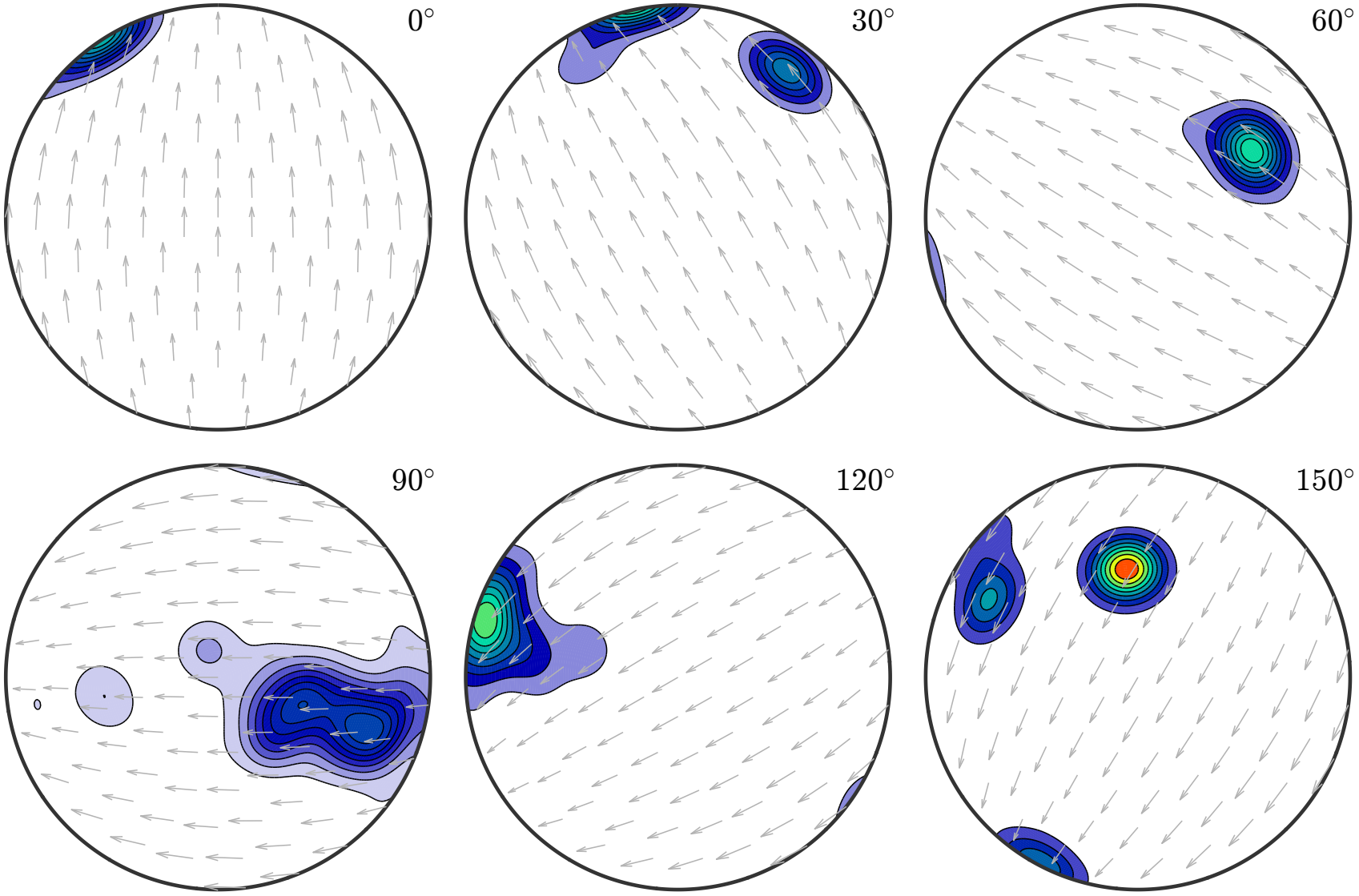
Plotting in Orientation Space

```
plotSection(O,'phi2',(0:30:150)*degree,...
                'contourf','halfwidth',10*degree)
```



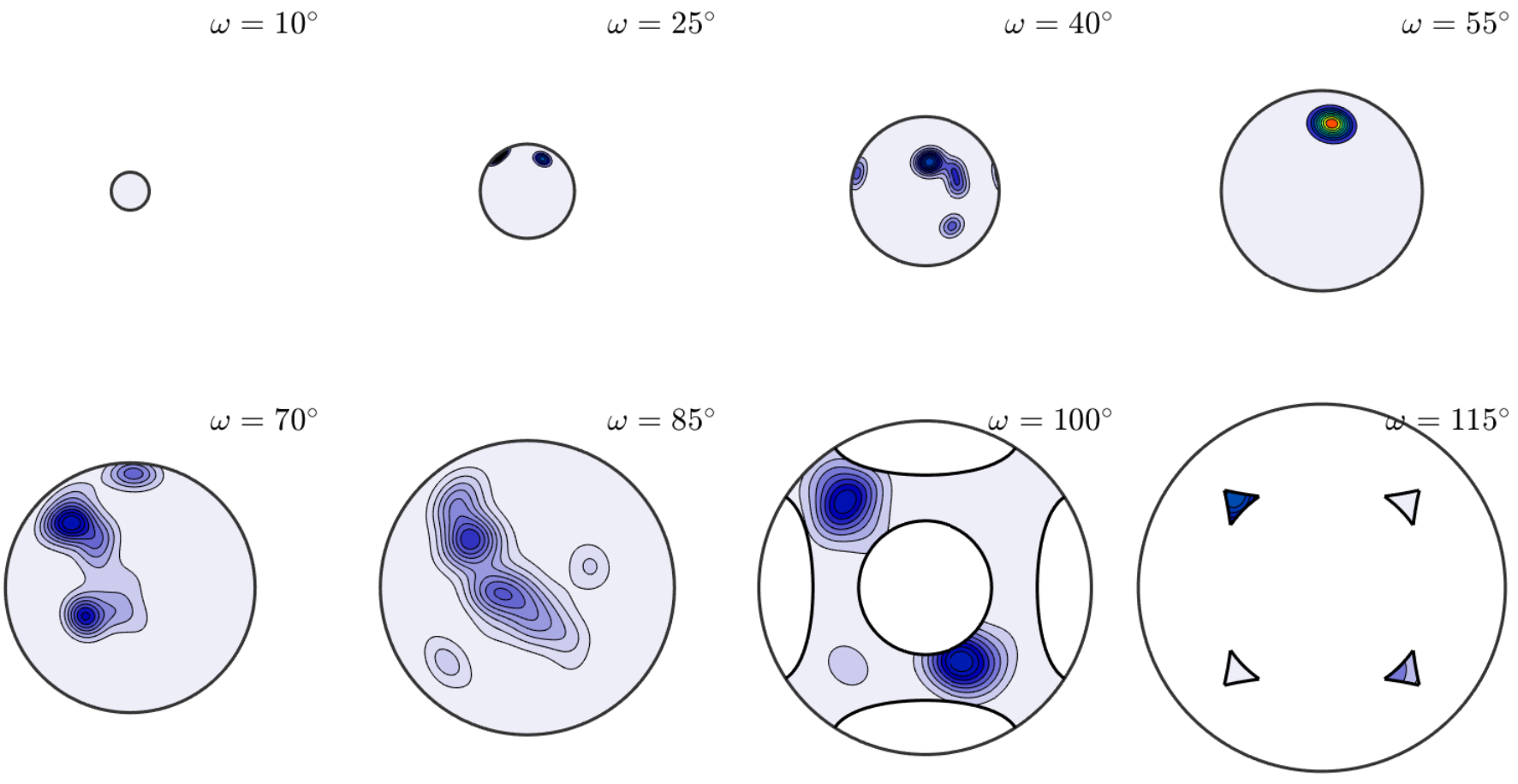
Plotting in Orientation Space

```
plotSection(O, 'sigma',(0:30:150)*degree, 'contourf')
```



Plotting in Orientation Space

```
plotSection(O, 'AxisAngle',(10:15:115)*degree,...
                'contourf')
```



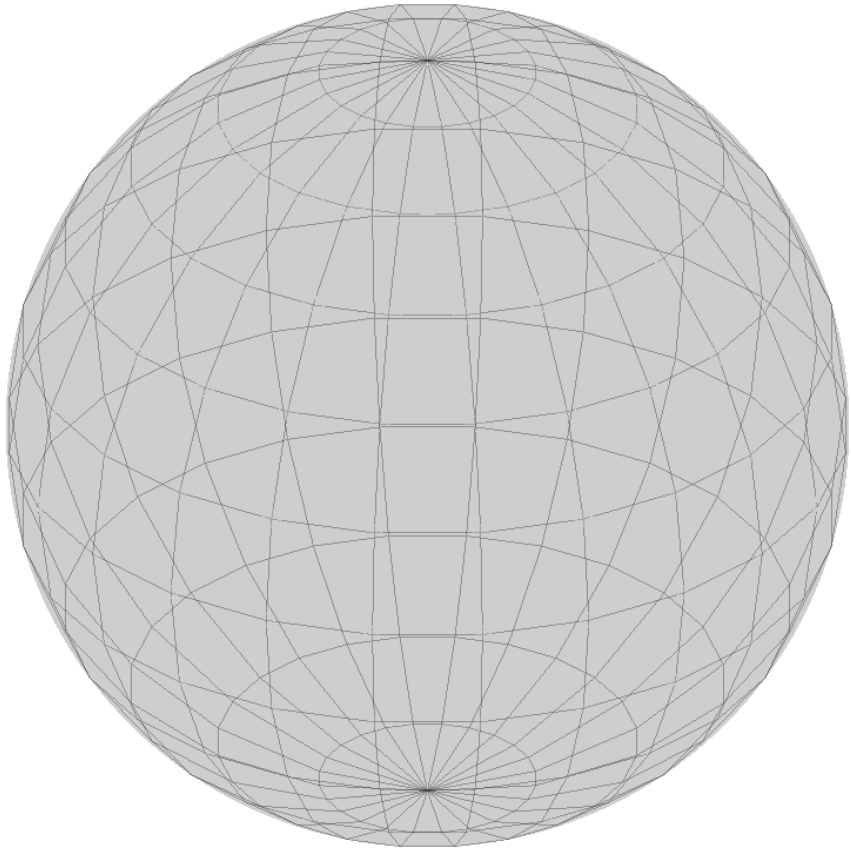
The Orientation Space

```
plot(orientationRegion)
```

```
cs = crystalSymmetry('mmm')
oR = cs.fundamentalRegion
plot(oR,'color','r')
```

```
cs = crystalSymmetry('321')
oR = cs.fundamentalRegion
plot(oR,'color','r')
```

```
cs = crystalSymmetry('432')
oR = cs.fundamentalRegion
plot(oR,'color','r')
```

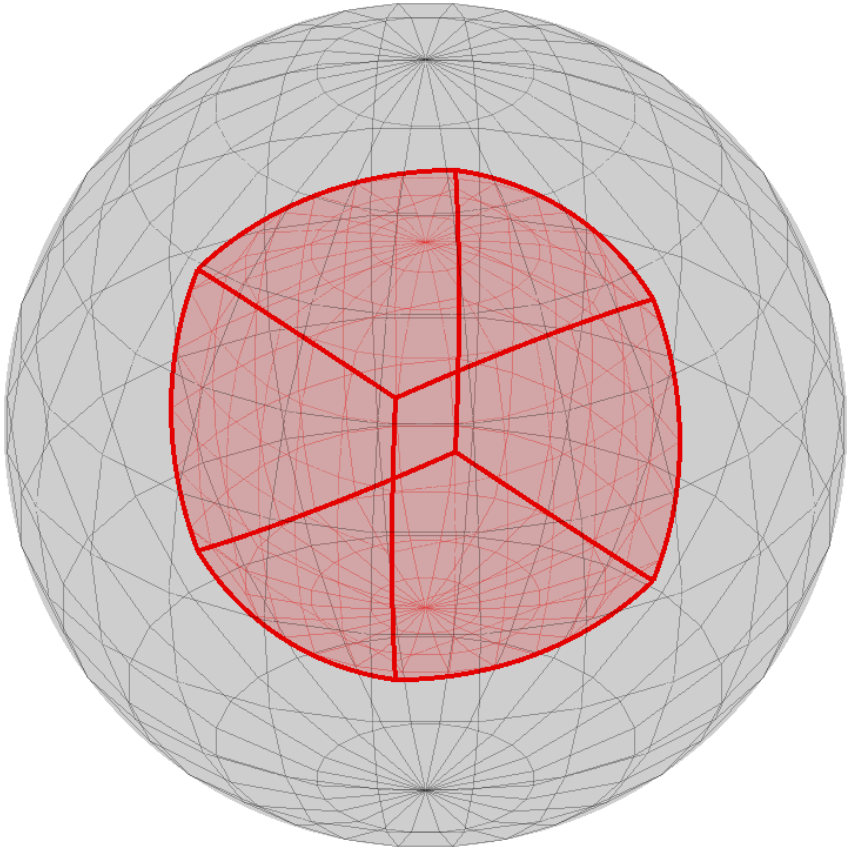


```
oR.V, oR.N, oR.checkInside, oR.axisSector(omega),
oR.maxAngle(axes), oR.minAngle,
oR.calcAxisDistribution, oR.calcAngleDistribution
```

The Orientation Space

```
plot(orientationRegion)
```

```
cs = crystalSymmetry('mmm')
oR = cs.fundamentalRegion
plot(oR,'color','r')
```



```
cs = crystalSymmetry('321')
oR = cs.fundamentalRegion
plot(oR,'color','r')
```

```
cs = crystalSymmetry('432')
oR = cs.fundamentalRegion
plot(oR,'color','r')
```

```
oR.V, oR.N, oR.checkInside, oR.axisSector(omega),
oR.maxAngle(axes), oR.minAngle,
oR.calcAxisDistribution, oR.calcAngleDistribution
```

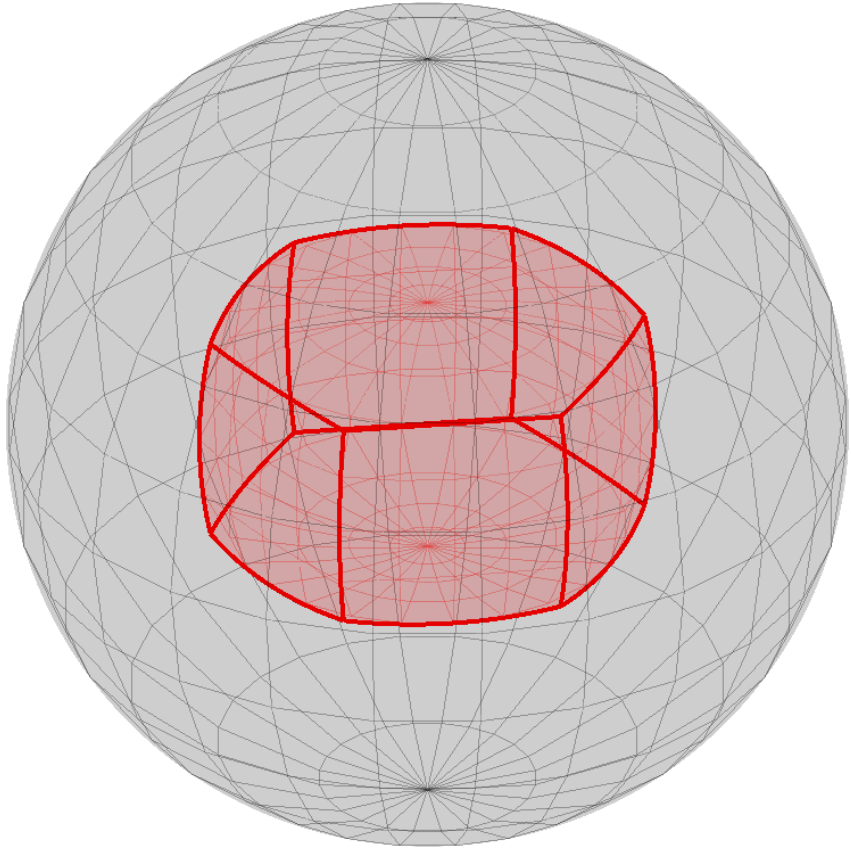
The Orientation Space

```
plot(orientationRegion)
```

```
cs = crystalSymmetry('mmm')
oR = cs.fundamentalRegion
plot(oR,'color','r')
```

```
cs = crystalSymmetry('321')
oR = cs.fundamentalRegion
plot(oR,'color','r')
```

```
cs = crystalSymmetry('432')
oR = cs.fundamentalRegion
plot(oR,'color','r')
```



```
oR.V, oR.N, oR.checkInside, oR.axisSector(omega),
oR.maxAngle(axes), oR.minAngle,
oR.calcAxisDistribution, oR.calcAngleDistribution
```

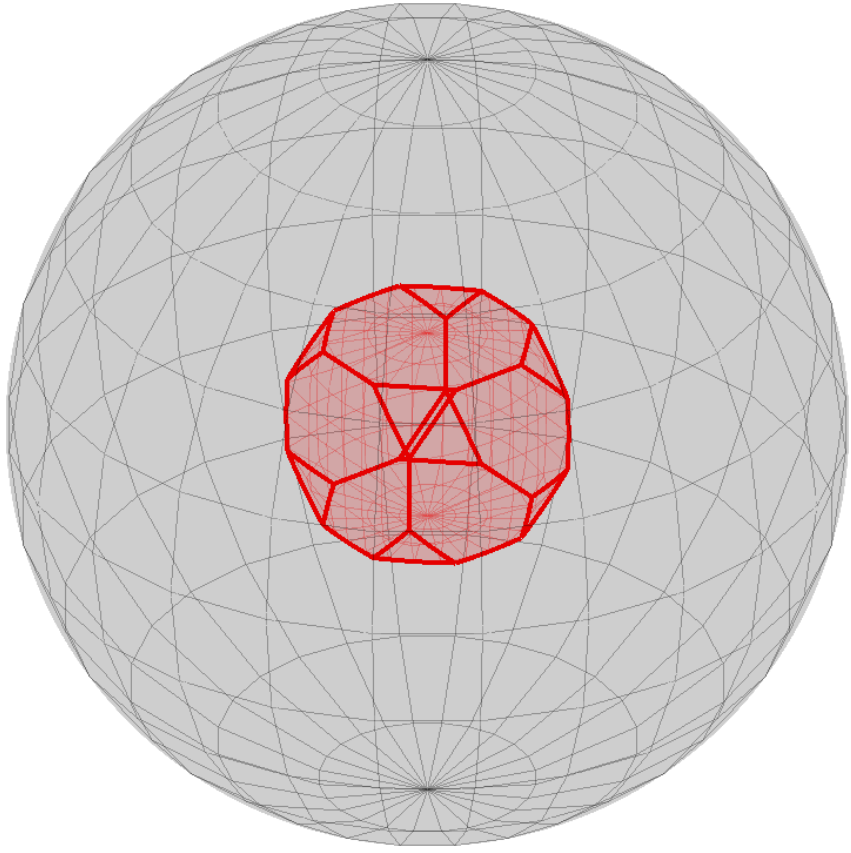
The Orientation Space

```
plot(orientationRegion)
```

```
cs = crystalSymmetry('mmm')
oR = cs.fundamentalRegion
plot(oR,'color','r')
```

```
cs = crystalSymmetry('321')
oR = cs.fundamentalRegion
plot(oR,'color','r')
```

```
cs = crystalSymmetry('432')
oR = cs.fundamentalRegion
plot(oR,'color','r')
```



```
oR.V, oR.N, oR.checkInside, oR.axisSector(omega),
oR.maxAngle(axes), oR.minAngle,
oR.calcAxisDistribution, oR.calcAngleDistribution
```

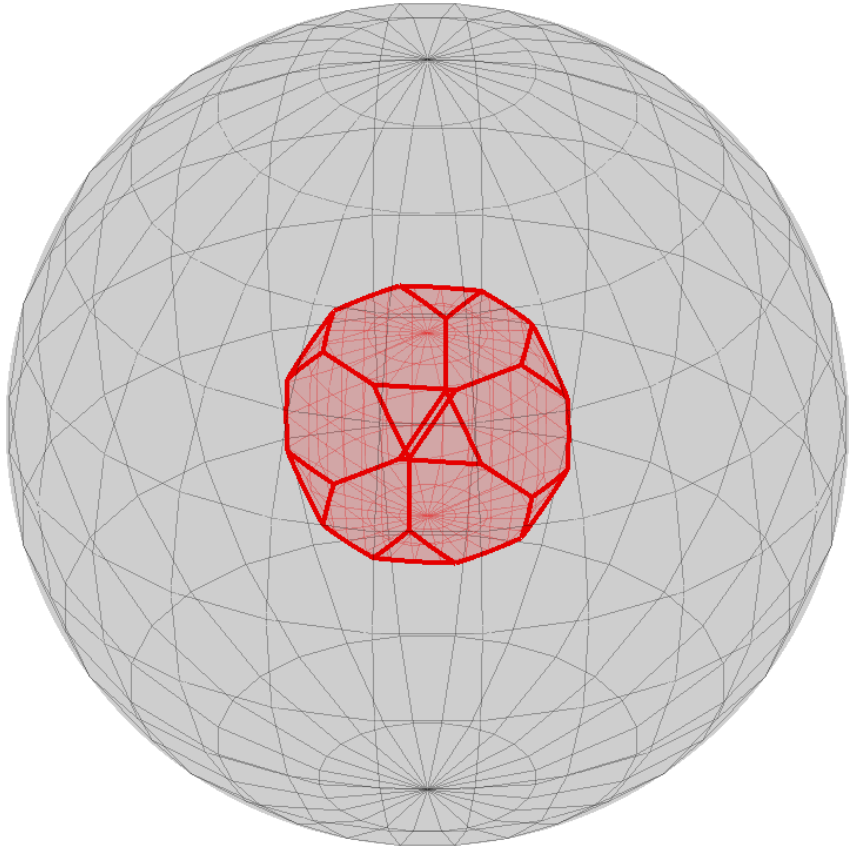
The Orientation Space

```
plot(orientationRegion)
```

```
cs = crystalSymmetry('mmm')
oR = cs.fundamentalRegion
plot(oR,'color','r')
```

```
cs = crystalSymmetry('321')
oR = cs.fundamentalRegion
plot(oR,'color','r')
```

```
cs = crystalSymmetry('432')
oR = cs.fundamentalRegion
plot(oR,'color','r')
```



```
oR.V, oR.N, oR.checkInside, oR.axisSector(omega),
oR.maxAngle(axes), oR.minAngle,
oR.calcAxisDistribution, oR.calcAngleDistribution
```

Operations on Orientations

mean orientation

```
mean(O)
```

```
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
```

mean orientation spread

```
mean(angle(O,mean(O))) ./ degree
```

volume portions

```
volume(O,mean(O),10*degree)
fibreVolume(O,Miller(1,0,0,O.CS),vector3d.X,5*degree)
```

export to ASCII file

Operations on Orientations

mean orientation

```
mean(O)
```

mean orientation spread

```
mean(angle(O,mean(O)))./degree
```

```
ans =

   47.2287
```

volume portions

```
volume(O,mean(O),10*degree)
fibreVolume(O,Miller(1,0,0,O.CS),vector3d.X,5*degree)
```

export to ASCII file

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

Operations on Orientations

mean orientation

```
mean(O)
```

mean orientation spread

```
mean(angle(O,mean(O)))./degree
```

volume portions

```
volume(O,mean(O),10*degree)
fibreVolume(O,Miller(1,0,0,O.CS),vector3d.X,5*degree)
```

export to ASCII file

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

Operations on Orientations

mean orientation

```
mean(O)
```

mean orientation spread

```
mean(angle(O,mean(O)))./degree
```

volume portions

```
volume(O,mean(O),10*degree)
fibreVolume(O,Miller(1,0,0,O.CS),vector3d.X,5*degree)
```

export to ASCII file

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