Matlab Basics and General Concepts of the MTEX Toolbox

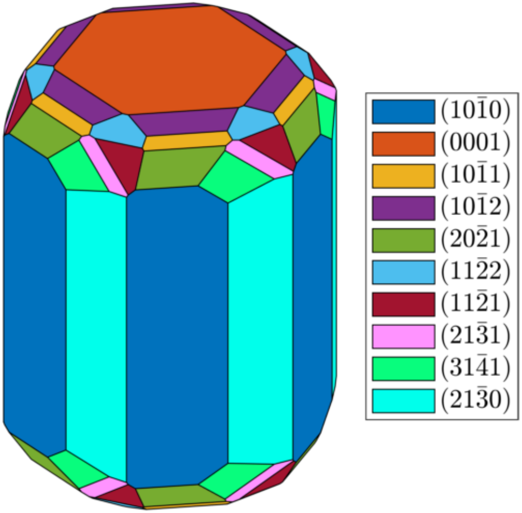
R. Hielscher

Faculty of Mathematics,
Chemnitz University of Technology, Germany

2021

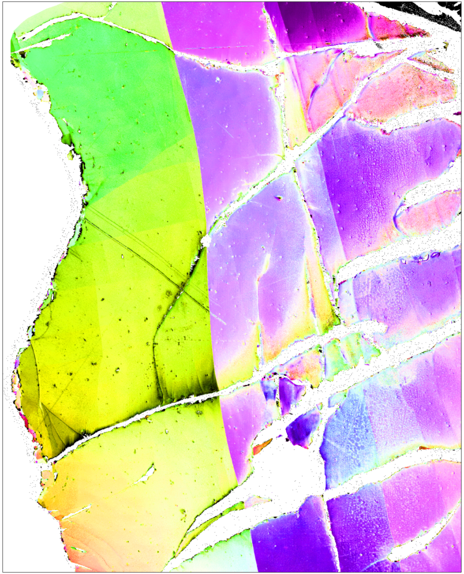
MTEX - A tool for calculating with orientation dependent properties

- crystal geometry
- EBSD data
- grains, grain boundaries
- XRD data
- orientation distribution function
- texture simulations
- tensorial properties
- elastic / plastic deformation
- misorientations / twinning
- phase transformations
- parent grain reconstruction



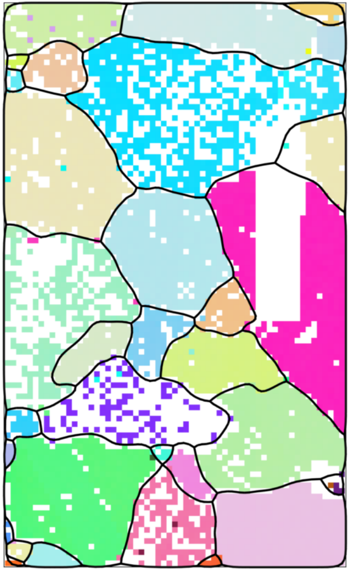
MTEX - A tool for calculating with orientation dependent properties

- crystal geometry
- EBSD data
- grains, grain boundaries
- XRD data
- orientation distribution function
- texture simulations
- tensorial properties
- elastic / plastic deformation
- misorientations / twinning
- phase transformations
- parent grain reconstruction



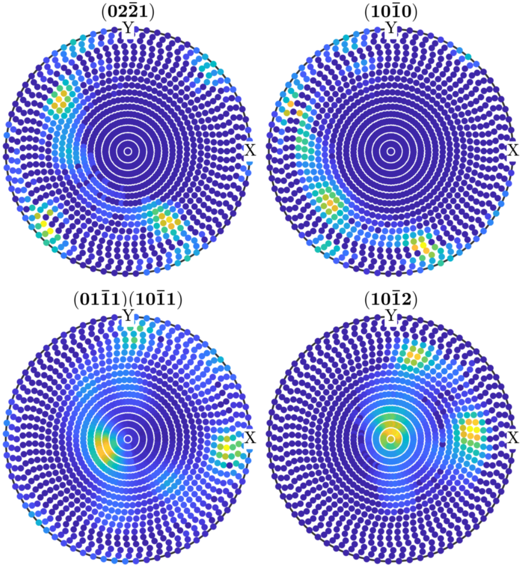
MTEX - A tool for calculating with orientation dependent properties

- crystal geometry
- EBSD data
- grains, grain boundaries
- XRD data
- orientation distribution function
- texture simulations
- tensorial properties
- elastic / plastic deformation
- misorientations / twinning
- phase transformations
- parent grain reconstruction



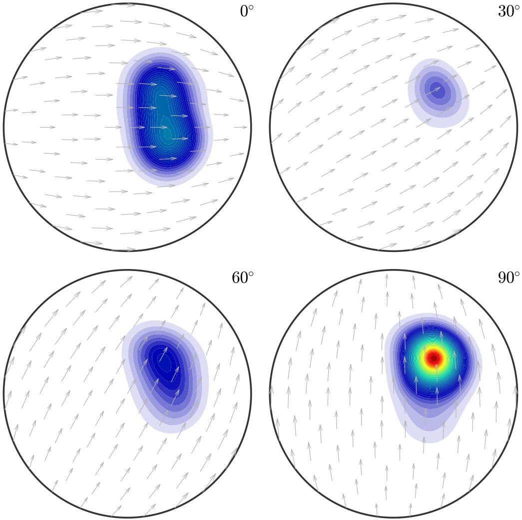
MTEX - A tool for calculating with orientation dependent properties

- crystal geometry
- EBSD data
- grains, grain boundaries
- XRD data
- orientation distribution function
- texture simulations
- tensorial properties
- elastic / plastic deformation
- misorientations / twinning
- phase transformations
- parent grain reconstruction



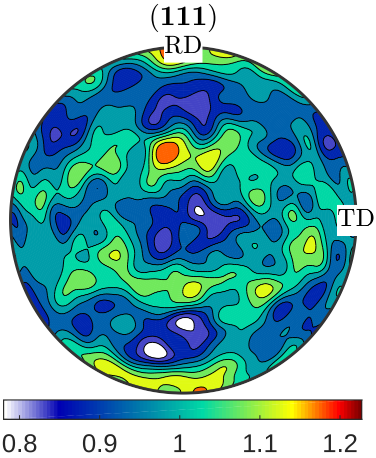
MTEX - A tool for calculating with orientation dependent properties

- crystal geometry
- EBSD data
- grains, grain boundaries
- XRD data
- orientation distribution function
- texture simulations
- tensorial properties
- elastic / plastic deformation
- misorientations / twinning
- phase transformations
- parent grain reconstruction



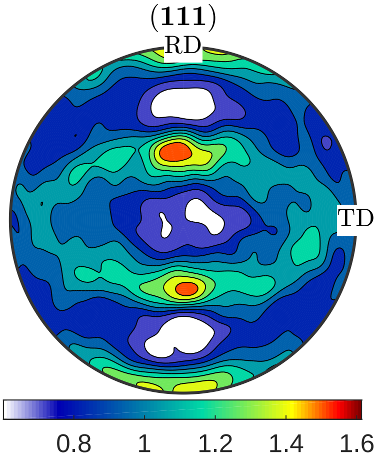
MTEX - A tool for calculating with orientation dependent properties

- crystal geometry
- EBSD data
- grains, grain boundaries
- XRD data
- orientation distribution function
- texture simulations
- tensorial properties
- elastic / plastic deformation
- misorientations / twinning
- phase transformations
- parent grain reconstruction



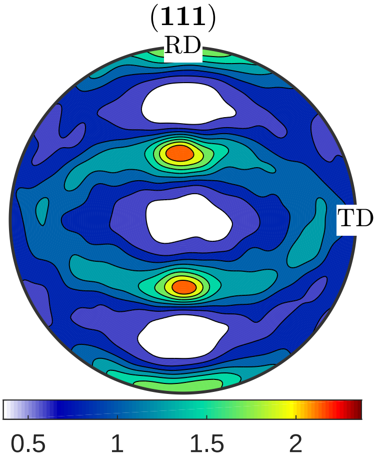
MTEX - A tool for calculating with orientation dependent properties

- crystal geometry
- EBSD data
- grains, grain boundaries
- XRD data
- orientation distribution function
- texture simulations
- tensorial properties
- elastic / plastic deformation
- misorientations / twinning
- phase transformations
- parent grain reconstruction



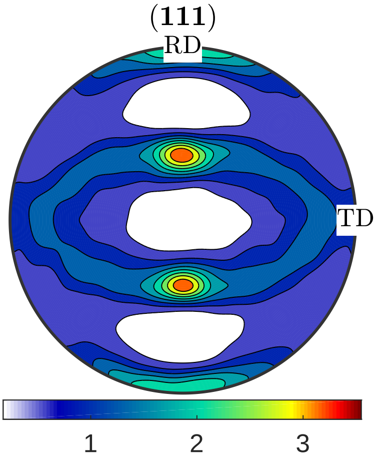
MTEX - A tool for calculating with orientation dependent properties

- crystal geometry
- EBSD data
- grains, grain boundaries
- XRD data
- orientation distribution function
- texture simulations
- tensorial properties
- elastic / plastic deformation
- misorientations / twinning
- phase transformations
- parent grain reconstruction



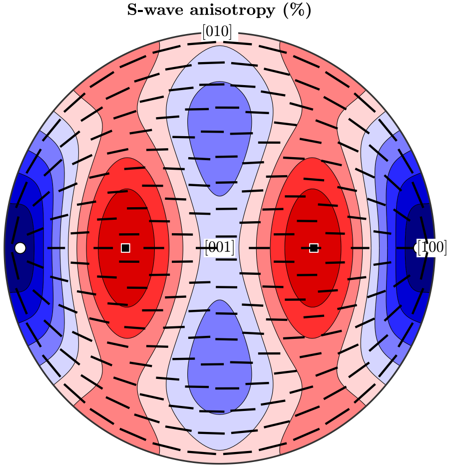
MTEX - A tool for calculating with orientation dependent properties

- crystal geometry
- EBSD data
- grains, grain boundaries
- XRD data
- orientation distribution function
- texture simulations
- tensorial properties
- elastic / plastic deformation
- misorientations / twinning
- phase transformations
- parent grain reconstruction



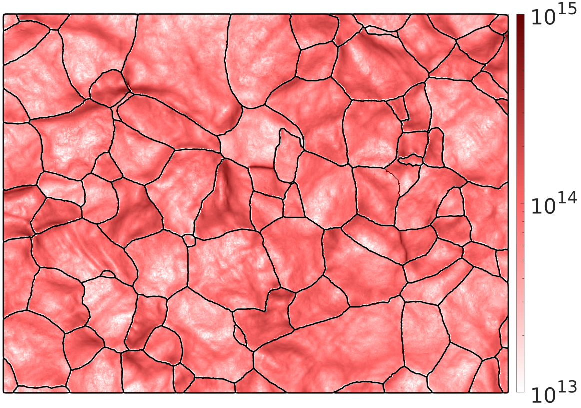
MTEX - A tool for calculating with orientation dependent properties

- crystal geometry
- EBSD data
- grains, grain boundaries
- XRD data
- orientation distribution function
- texture simulations
- tensorial properties
- elastic / plastic deformation
- misorientations / twinning
- phase transformations
- parent grain reconstruction



MTEX - A tool for calculating with orientation dependent properties

- crystal geometry
- EBSD data
- grains, grain boundaries
- XRD data
- orientation distribution function
- texture simulations
- tensorial properties
- elastic / plastic deformation
- misorientations / twinning
- phase transformations
- parent grain reconstruction



MTEX - A tool for calculating with orientation dependent properties

- crystal geometry
- EBSD data
- grains, grain boundaries
- XRD data
- orientation distribution function
- texture simulations
- tensorial properties
- elastic / plastic deformation
- misorientations / twinning
- phase transformations
- parent grain reconstruction

MTEX - A tool for calculating with orientation dependent properties

- crystal geometry
- EBSD data
- grains, grain boundaries
- XRD data
- orientation distribution function
- texture simulations
- tensorial properties
- elastic / plastic deformation
- misorientations / twinning
- phase transformations
- parent grain reconstruction

MTEX - A tool for calculating with orientation dependent properties

- crystal geometry
- EBSD data
- grains, grain boundaries
- XRD data
- orientation distribution function
- texture simulations
- tensorial properties
- elastic / plastic deformation
- misorientations / twinning
- phase transformations
- parent grain reconstruction

MTEX - A tool for calculating with orientation dependent properties

- crystal geometry
- EBSD data
- grains, grain boundaries
- XRD data
- orientation distribution function
- texture simulations
- tensorial properties
- elastic / plastic deformation
- misorientations / twinning
- phase transformations
- parent grain reconstruction

Focus: basics, generality, speed

MTEX an Open Source Toolbox

Large, well documented, tested

- 13 years of development
- 1000 functions
- 40 000 lines of code, 33 percent comments
- 14 reference paper, about 2000 references
- 2000 downloads per version
- 1000 help pages
- add-ons: MTEX GUI, MTEX2Gmsh, Stabix, CrystalAligner, phaseSegmenter

A Teaching Tool

- everything can be visualized
- everything can be manipulated
- everything can be combined with everything

Free and open software

- free to use
- free to modify
- very nice comunity

MTEX - A Matlab based scripting language

```
% load data
ebsd = EBSD.load('Emsland_plessite.ctf')

% plot data
plot(ebsd('Fe'),ebsd('Fe').orientations)

% reconstruct grains
th = 5*degree;
grains = calcGrains(ebsd,'threshold',th)

% find largest grain
[m,id] = max(grains.area)

% plot largest grain
plot(grains(id).boundary,'linewidth',2)
```

MTEX - A Matlab based scripting language

```
% load data
ebsd = EBSD.load('Emsland_plessite.ctf')

% plot data
plot(ebsd('Fe'),ebsd('Fe').orientations)

% reconstruct grains
th = 5*degree;
grains = calcGrains(ebsd,'threshold',th)

% find largest grain
[m,id] = max(grains.area)

% plot largest grain
plot(grains(id).boundary,'linewidth',2)
```

Why scripts?

- reproducible results
- easy to document
- templates for common tasks
- extensively customizable
- batch processing of many data sets
- repeated calculations with different parameters

Best practice

- comment your scripts
- short scripts
- function for repeated tasks
- avoid loops

MTEX Resources

- documentation
- function reference
- examples
- user scripts
- discussion forum

Matlab

centered around matrices

Three dimensional vectors

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)
```

Three dimensional vectors

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
```

Three dimensional vectors

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

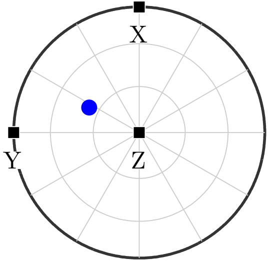
$$\vec{r} = x \cdot \vec{X} + y \cdot \vec{Y} + z \cdot \vec{Z}$$

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)
```



Three dimensional vectors

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

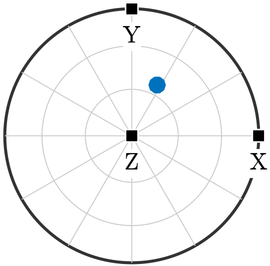
$$\vec{r} = x \cdot \vec{X} + y \cdot \vec{Y} + z \cdot \vec{Z}$$

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

predefined vectors

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


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

```
theta = 90 * degree; rho = 45 * degree;
r = vector3d.byPolar(theta,rho)
```

In MTEX all angles are in radiant!

combine vectors

```
r = [vector3d.X, vector3d.Y, vector3d(1,1,1)]
```

importing vectors

```
r = vector3d.load('file','ColumnNames',{'x','y','z'})
```

random vectors

```
r = vector3d.rand(100)
```

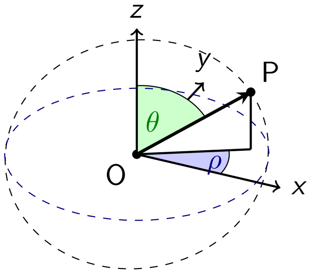
Defining vectors

predefined vectors

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

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

```
theta = 90 * degree; rho = 45 * degree;
r = vector3d.byPolar(theta,rho)
```



In MTEX all angles are in radiant!

combine vectors

```
r = [vector3d.X, vector3d.Y, vector3d(1,1,1)]
```

importing vectors

```
r = vector3d.load('file','ColumnNames',{'x','y','z'})
```

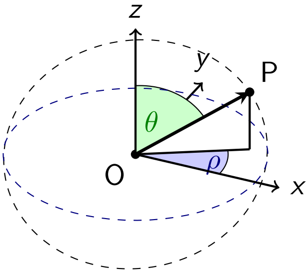
random vectors

```
r = vector3d.rand(100)
```

Defining vectors

predefined vectors

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



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

```
theta = 90 * degree; rho = 45 * degree;
r = vector3d.byPolar(theta,rho)
```

In MTEX all angles are in radiant!

combine vectors

```
r = [vector3d.X, vector3d.Y, vector3d(1,1,1)]
```

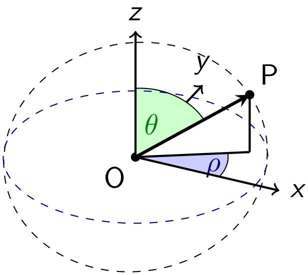
```
r = vector3d (show methods, plot)
  size: 1 x 3
  x y z
  1 0 0
  0 1 0
  1 1 1
```

importing vectors

Defining vectors

predefined vectors

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



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

```
theta = 90 * degree; rho = 45 * degree;
r = vector3d.byPolar(theta,rho)
```

In MTEX all angles are in radiant!

combine vectors

```
r = [vector3d.X, vector3d.Y, vector3d(1,1,1)]
```

importing vectors

```
r = vector3d.load('file','ColumnNames',{'x','y','z'})
```

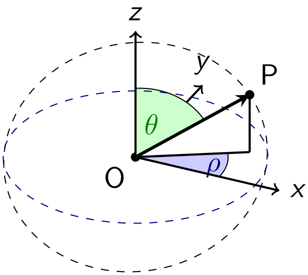
```
r = vector3d (show methods, plot)
  size: 200 x 1
```

random vectors

Defining vectors

predefined vectors

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



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

```
theta = 90 * degree; rho = 45 * degree;
r = vector3d.byPolar(theta,rho)
```

In MTEX all angles are in radiant!

combine vectors

```
r = [vector3d.X, vector3d.Y, vector3d(1,1,1)]
```

importing vectors

```
r = vector3d.load('file','ColumnNames',{'x','y','z'})
```

random vectors

```
r = vector3d.rand(100)
```

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
normalize(v)    % scale to norm 1
orth(v)         % arbitrary orthogonal vector
```

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
normalize(v)    % scale to norm 1
orth(v)         % arbitrary orthogonal vector
```

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
normalize(v)    % scale to norm 1
orth(v)         % arbitrary orthogonal vector
```

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

```
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)
```

```
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, tensors, 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)
```

The above techniques applies also to lists of rotations, orientations, tensors, 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])
```

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
```

The above techniques applies also to lists of rotations, orientations, tensors, 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])
```

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

```
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
```

```
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

```
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) = []
```

```
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
```

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

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
```

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

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

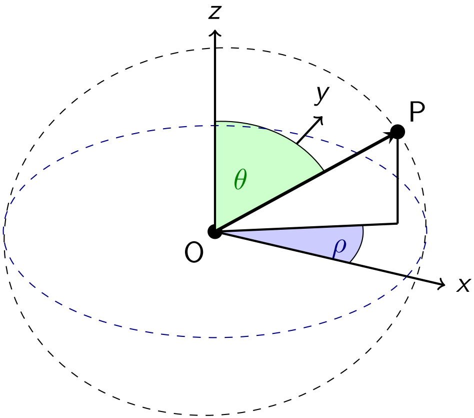
```
r.x = 0
```

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

Spherical Projections

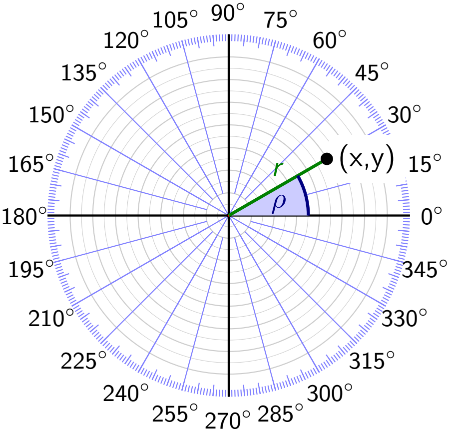
spherical polar coordinates

$$(x, y, z) = (\cos\rho \sin\theta, \sin\rho \sin\theta, \cos\theta)$$

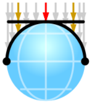
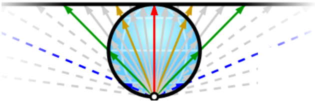
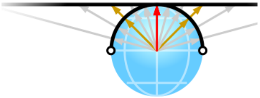


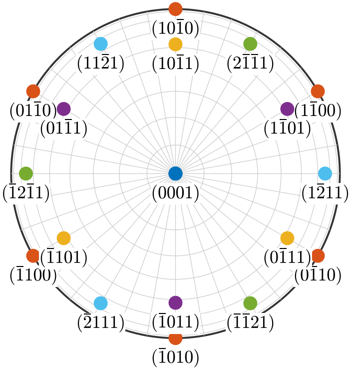
polar coordinates in the plane

$$(x, y) = (r \cos\rho, r \sin\rho)$$

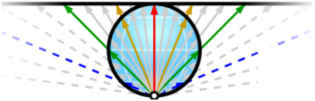
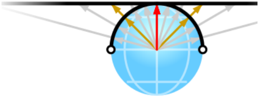


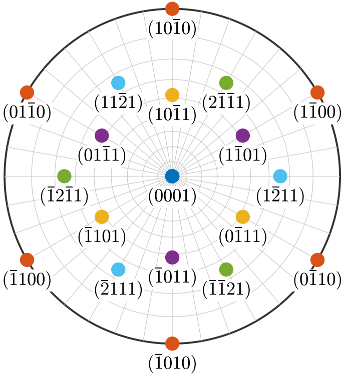
Spherical Projections

name	formula	schema
orthographic	$r = \sin\theta$	
equal angle (stereographic)	$r = \tan\frac{\theta}{2}$	
gnomonic	$r = \tan\theta$	
equal area (Schmidt)	$\sqrt{2(1-\cos\theta)}$	
equal distant	$r = \theta$	

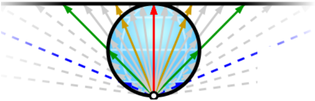
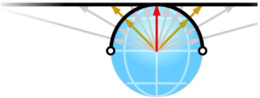


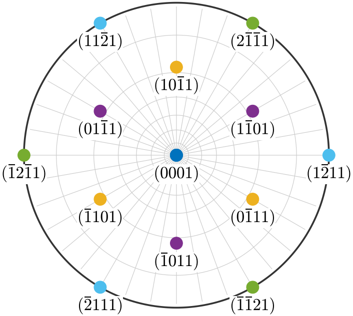
Spherical Projections

name	formula	schema
orthographic	$r = \sin\theta$	
equal angle (stereographic)	$r = \tan\frac{\theta}{2}$	
gnonomic	$r = \tan\theta$	
equal area (Schmidt)	$\sqrt{2(1-\cos\theta)}$	
equal distant	$r = \theta$	

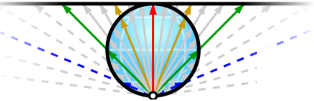
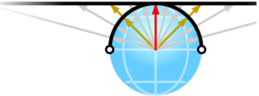


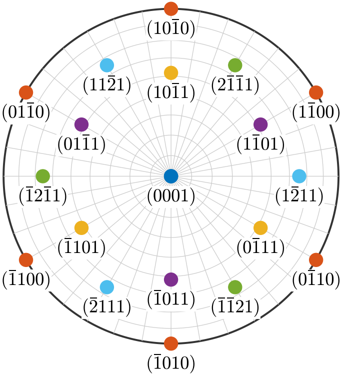
Spherical Projections

name	formula	schema
orthographic	$r = \sin\theta$	
equal angle (stereographic)	$r = \tan\frac{\theta}{2}$	
gnomonic	$r = \tan\theta$	
equal area (Schmidt)	$\sqrt{2(1-\cos\theta)}$	
equal distant	$r = \theta$	

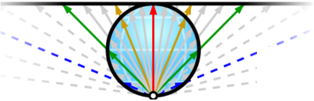
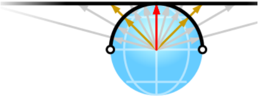


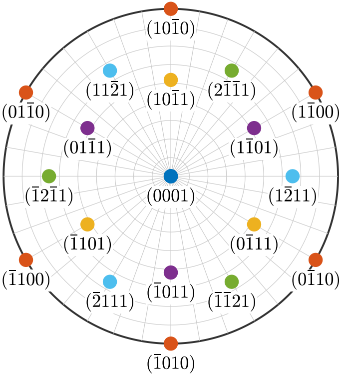
Spherical Projections

name	formula	schema
orthographic	$r = \sin\theta$	
equal angle (stereographic)	$r = \tan\frac{\theta}{2}$	
gnomonic	$r = \tan\theta$	
equal area (Schmidt)	$\sqrt{2(1-\cos\theta)}$	
equal distant	$r = \theta$	



Spherical Projections

name	formula	schema
orthographic	$r = \sin\theta$	
equal angle (stereographic)	$r = \tan\frac{\theta}{2}$	
gnonomic	$r = \tan\theta$	
equal area (Schmidt)	$\sqrt{2(1-\cos\theta)}$	
equal distant	$r = \theta$	



Spherical Plots in MTEX

spherical projections: `earea, edist, eangle, 3d`

spherical region: `upper, lower, complete, fundamentalRegion`

plot alignment: `plotx2east, plotx2north, plotzIntoPlane, plotzOutOfPlane`

`marker`: `s, d, o, v`
`markerSize, markerEdgeColor, markerFaceColor, linewidth, MarkerEdgeAlpha, MarkerFaceAlpha`

combined plots: `hold on, hold off, add2all`

multiple plots: `nextAxis, newMTEXFigure, gcm`

labels: `label, fontSize, backgroundcolor,`

Spherical Plots in MTEX

spherical projections: `earea`, `edist`, `eangle`, `3d`

spherical region: `upper`, `lower`, `complete`, `fundamentalRegion`

plot alignment: `plotx2east`, `plotx2north`, `plotzIntoPlane`, `plotzOutOfPlane`

marker: `s`, `d`, `o`, `v`
`markerSize`, `markerEdgeColor`, `markerFaceColor`, `linewidth`, `MarkerEdgeAlpha`, `MarkerFaceAlpha`

combined plots: `hold on`, `hold off`, `add2all`

multiple plots: `nextAxis`, `newMTEXFigure`, `gcm`

labels: `label`, `fontSize`, `backgroundcolor`,

Spherical Plots in MTEX

spherical projections: `earea`, `edist`, `eangle`, `3d`

spherical region: `upper`, `lower`, `complete`, `fundamentalRegion`

plot alignment: `plotx2east`, `plotx2north`, `plotzIntoPlane`, `plotzOutOfPlane`

marker: `s`, `d`, `o`, `v`
`markerSize`, `markerEdgeColor`, `markerFaceColor`, `linewidth`, `MarkerEdgeAlpha`, `MarkerFaceAlpha`

combined plots: `hold on`, `hold off`, `add2all`

multiple plots: `nextAxis`, `newMTEXFigure`, `gcm`

labels: `label`, `fontSize`, `backgroundcolor`,

Spherical Plots in MTEX

spherical projections: `earea, edist, eangle, 3d`

spherical region: `upper, lower, complete, fundamentalRegion`

plot alignment: `plotx2east, plotx2north, plotzIntoPlane, plotzOutOfPlane`

`marker`: `s, d, o, v`
`markerSize, markerEdgeColor, markerFaceColor, linewidth, MarkerEdgeAlpha, MarkerFaceAlpha`

combined plots: `hold on, hold off, add2all`

multiple plots: `nextAxis, newMTEXFigure, gcm`

labels: `label, fontSize, backgroundcolor,`

Spherical Plots in MTEX

spherical projections: earea, edist, eangle, 3d

spherical region: upper, lower, complete, fundamentalRegion

plot alignment: plotx2east, plotx2north, plotzIntoPlane, plotzOutOfPlane

marker: s, d, o, v
markerSize, markerEdgeColor, markerFaceColor, linewidth, MarkerEdgeAlpha, MarkerFaceAlpha

combined plots: hold on, hold off, add2all

multiple plots: nextAxis, newMTEXFigure, gcm

labels: label, fontSize, backgroundcolor,

Spherical Plots in MTEX

spherical projections: `earea, edist, eangle, 3d`

spherical region: `upper, lower, complete, fundamentalRegion`

plot alignment: `plotx2east, plotx2north, plotzIntoPlane, plotzOutOfPlane`

`marker`: `s, d, o, v`
`markerSize, markerEdgeColor, markerFaceColor, linewidth, MarkerEdgeAlpha, MarkerFaceAlpha`

combined plots: `hold on, hold off, add2all`

multiple plots: `nextAxis, newMTEXFigure, gcm`

labels: `label, fontSize, backgroundcolor,`

Spherical Plots in MTEX

spherical projections: `earea, edist, eangle, 3d`

spherical region: `upper, lower, complete, fundamentalRegion`

plot alignment: `plotx2east, plotx2north, plotzIntoPlane, plotzOutOfPlane`

`marker`: `s, d, o, v`
`markerSize, markerEdgeColor, markerFaceColor, linewidth, MarkerEdgeAlpha, MarkerFaceAlpha`

combined plots: `hold on, hold off, add2all`

multiple plots: `nextAxis, newMTEXFigure, gcm`

labels: `label, fontSize, backgroundcolor,`

Data Plots

colorize vectors by value

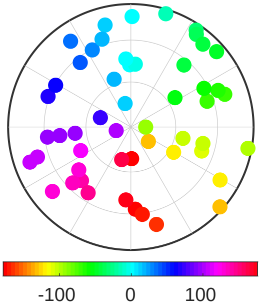
```
v = vector3d.rand(100)
scatter(v,v.rho./degree)
mtexColorbar southoutside
mtexColorMap hsv
```

colorize by RGB triples

```
key = HSVDirectionKey
scatter(v,key.direction2color(v))
```

visualize directions

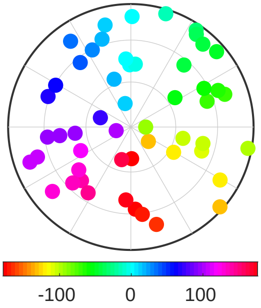
```
quiver(v,orth(v)) % a vector field
```



Data Plots

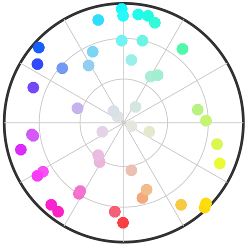
colorize vectors by value

```
v = vector3d.rand(100)
scatter(v,v.rho./degree)
mtexColorbar southoutside
mtexColorMap hsv
```



colorize by RGB triples

```
key = HSVDirectionKey
scatter(v,key.direction2color(v))
```



visualize directions

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

Data Plots

colorize vectors by value

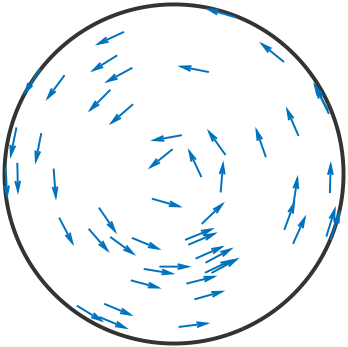
```
v = vector3d.rand(100)
scatter(v,v.rho./degree)
mtexColorbar southoutside
mtexColorMap hsv
```

colorize by RGB triples

```
key = HSVDirectionKey
scatter(v,key.direction2color(v))
```

visualize directions

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



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')
```

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

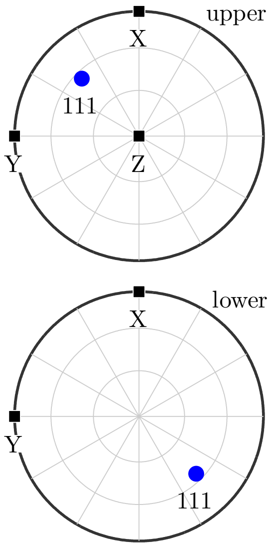
Then `r` and `-r` represent the same axis

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

The angle to an axis is always less then 90°

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

Changing option **antipodal**



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

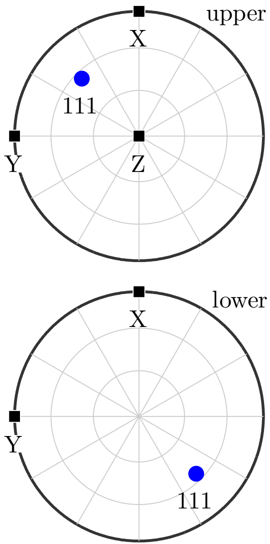
```
eq(r, -r)
```

```
1
```

The angle to an axis is always less then 90°

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

Changing option **antipodal**



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

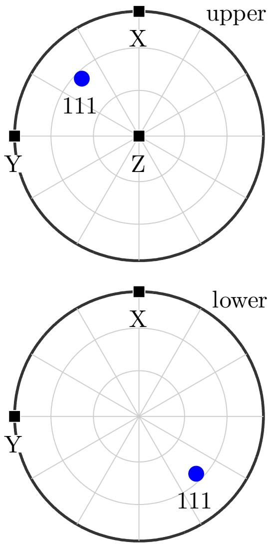
```
eq(r, -r)
```

The angle to an axis is always less then 90°

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

```
  54.7
```

Changing option **antipodal**



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

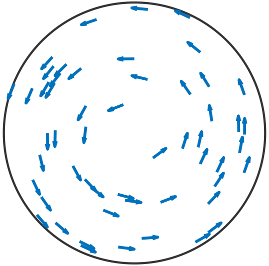
```
eq(r, -r)
```

The angle to an axis is always less then 90°

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

Changing option **antipodal**

```
r = vector3d.rand(100)
o = v.orth;
quiver(v,o)
```



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
```

Changing option **antipodal**

```
r = vector3d.rand(100)
o = v.orth;
o.antipodal = true;
quiver(v,o)
```