Tensor Calculations

R. Hielscher

Faculty of Mathematics,
Chemnitz University of Technology, Germany

MTEX Workshop 2018

Table of Content

What is a Tensor?

Tensors are used to described linear interactions between physical properties.

rank zero tensor scalar property, e.g. temperature

rank one tensor directional depended property, e.g. wave velocity

rank two tensors relationship between two vector fields, e.g. stress, strain, conductivity

rank three tensor relationship between a one and a two rank tensor, e.g. piezoelectricity

rank four tensor relationship between two two rank tensor, e.g. elasticity,

A tensor T of rank $s + t$ maps a tensor A of rank s onto a tensor B of rank t by the formula

$$B_{k_1,\ldots,k_t} = T_{k_1,k_2,\ldots,k_t,j_1,\ldots,j_s} A_{j_1,\ldots,j_s}$$

What is a Tensor?

Tensors are used to described linear interactions between physical properties.

rank zero tensor scalar property, e.g. temperature

rank one tensor directional depended property, e.g. wave velocity

rank two tensors relationship between two vector fields, e.g. stress, strain, conductivity

rank three tensor relationship between a one and a two rank tensor, e.g. piezoelectricity

rank four tensor relationship between two two rank tensor, e.g. elasticity,

A tensor T of rank $s + t$ maps a tensor A of rank s onto a tensor B of rank t by the formula

$$B_{k_1,\ldots,k_t} = T_{k_1,k_2,\ldots,k_t,j_1,\ldots,j_s} A_{j_1,\ldots,j_s}$$

What is a Tensor?

Tensors are used to described linear interactions between physical properties.

rank zero tensor scalar property, e.g. temperature

rank one tensor directional depended property, e.g. wave velocity

rank two tensors relationship between two vector fields, e.g. stress, strain, conductivity

rank three tensor relationship between a one and a two rank tensor, e.g. piezoelectricity

rank four tensor relationship between two two rank tensor, e.g. elasticity,

A tensor T of rank $s + t$ maps a tensor A of rank s onto a tensor B of rank t by the formula

$$B_{k_1,\ldots,k_t} = T_{k_1,k_2,\ldots,k_t,j_1,\ldots,j_s} A_{j_1,\ldots,j_s}$$

A Simple Example

```
M = [[1.45  0.00  0.19];...
     [0.00  2.11  0.00];...
     [0.19  0.00  1.79]];

sigma = stressTensor(M, 'unit', 'MPa');
```

```
sigma = stressTensor (show methods, plot)
  unit: MPa
  rank: 2 (3 x 3)

  1.45     0  0.19
     0  2.11     0
  0.19     0  1.79
```

A Simple Example

```
M = [[1.45  0.00  0.19];...
     [0.00  2.11  0.00];...
     [0.19  0.00  1.79]];

sigma = stressTensor(M, 'unit', 'MPa');
```

```
n = vector3d.X % normal direction
```

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

A Simple Example

```
M = [[1.45  0.00  0.19];...
     [0.00  2.11  0.00];...
     [0.19  0.00  1.79]];

sigma = stressTensor(M, 'unit', 'MPa');
```

```
n = vector3d.X % normal direction
```

the stress vector $T^{\vec{n}}$ of plane $\vec{n} = \{1, 0, 0\}$, is computed by $T_j^{\vec{n}} = \sigma_{ij}\vec{n}_i$.

A Simple Example

```
M = [[1.45  0.00  0.19];...
     [0.00  2.11  0.00];...
     [0.19  0.00  1.79]];

sigma = stressTensor(M, 'unit', 'MPa');
```

```
n = vector3d.X % normal direction
```

the stress vector $T^{\vec{n}}$ of plane $\vec{n} = \{1, 0, 0\}$, is computed by $T_j^{\vec{n}} = \sigma_{ij}\vec{n}_i$.

```
T = EinsteinSum(sigma,[-1  1],n,-1)
```

```
T = tensor (show methods, plot)
  unit: MPa
  rank: 1 (3)

 1.45
    0
 0.19
```

Einstein Summation

The scalar magnitudes of the normal stress σ_N and the shear stress σ_S are given as

$$\sigma_N = T_i^{\vec{n}} \vec{n}_i = \sigma_{ij} \vec{n}_i \vec{n}_j \quad \text{and} \quad \sigma_S = \sqrt{T_i^{\vec{n}} T_i^{\vec{n}} - \sigma_N^2}.$$

```
sigmaN = double(EinsteinSum(T,-1,n,-1))
sigmaS = sqrt(double(EinsteinSum(T,-1,T,-1))...
         - sigmaN^2)
```

```
sigmaN =

    1.4500

sigmaS =

    0.1900
```

Einstein Summation

The scalar magnitudes of the normal stress σ_N and the shear stress σ_S are given as

$$\sigma_N = T_i^{\vec{n}} \vec{n}_i = \sigma_{ij} \vec{n}_i \vec{n}_j \quad \text{and} \quad \sigma_S = \sqrt{T_i^{\vec{n}} T_i^{\vec{n}} - \sigma_N^2}.$$

```
sigmaN = double(EinsteinSum(T,-1,n,-1))
sigmaS = sqrt(double(EinsteinSum(T,-1,T,-1))...
         - sigmaN^2)
```

```
sigmaN =

    1.4500

sigmaS =

    0.1900
```

Visualization

For a second order tensor σ_{ij} its directional magnitude $R(\vec{x})$ is defined as

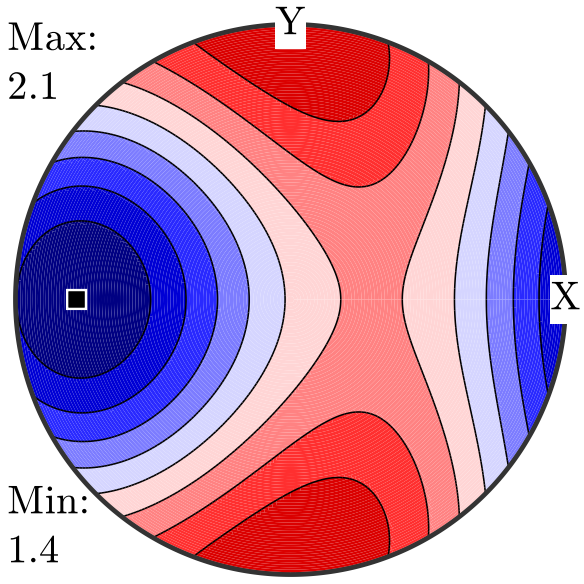
$$R(\vec{x}) = \sigma_{ij}\vec{x}_i\vec{x}_j.$$

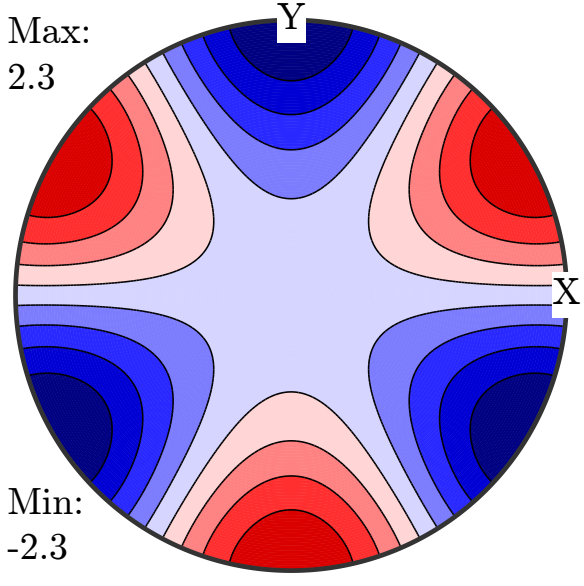
```
R = EinsteinSum(sigma,[-1 -2],...
  x,-1,x,-2)
```

```
R = directionalMagnitude(sigma)
```

```
[value,pos] = min(R)
```

```
plot(sigma.directionalMagnitude)
annotate(pos)
plot(sigma,'minmax')
mtexColorMap blue2red
```





Visualization

For a second order tensor σ_{ij} its directional magnitude $R(\vec{x})$ is defined as

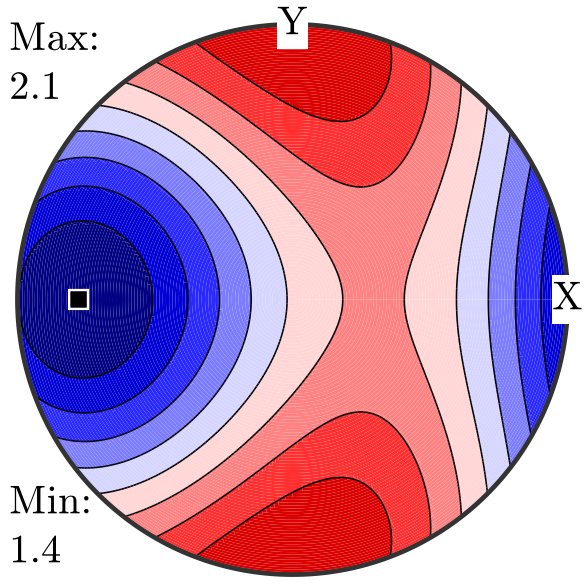
$$R(\vec{x}) = \sigma_{ij}\vec{x}_i\vec{x}_j.$$

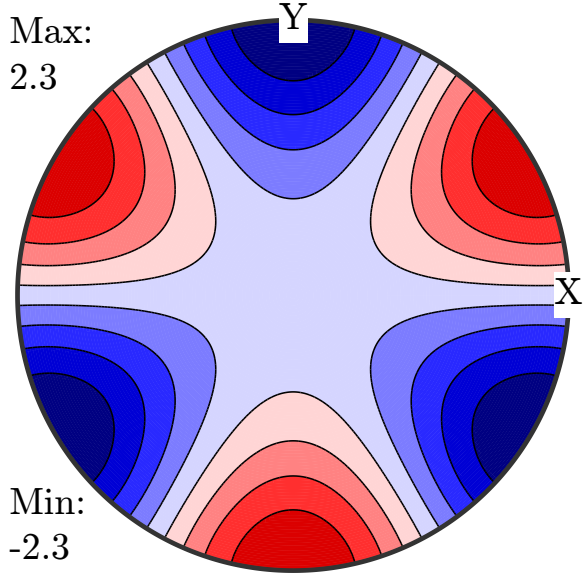
```
R = EinsteinSum(sigma,[-1 -2],...
    x,-1,x,-2)
```

```
R = directionalMagnitude(sigma)
```

```
[value,pos] = min(R)
```

```
plot(sigma.directionalMagnitude)
annotate(pos)
plot(sigma,'minmax')
mtexColorMap blue2red
```





Visualization

For a second order tensor σ_{ij} its directional magnitude $R(\vec{x})$ is defined as

$$R(\vec{x}) = \sigma_{ij}\vec{x}_i\vec{x}_j.$$

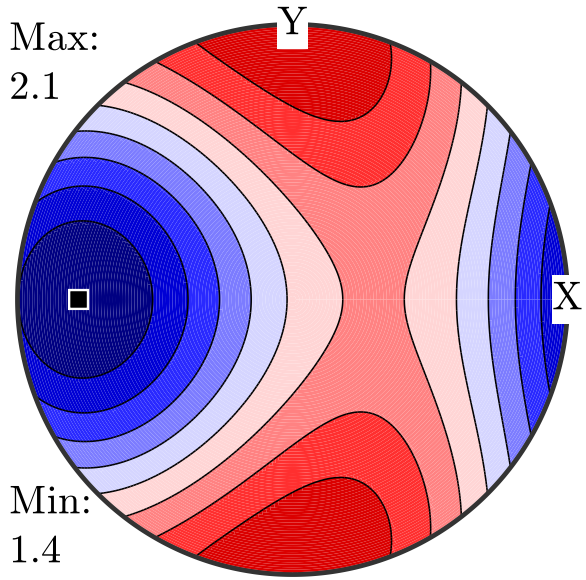
```
R = EinsteinSum(sigma,[-1 -2],...
    x,-1,x,-2)
```

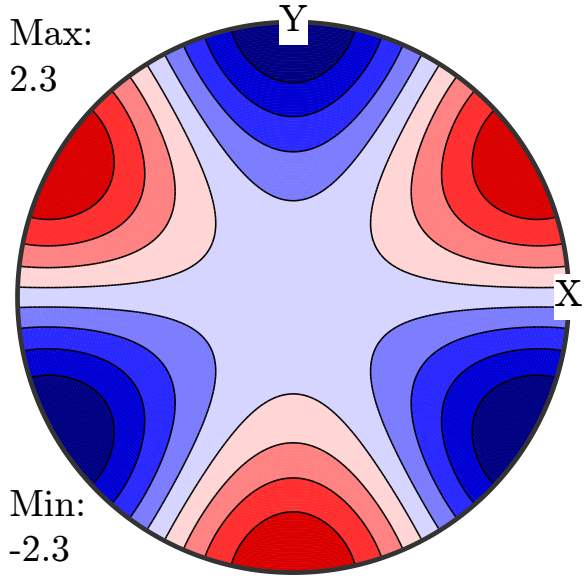
```
R = directionalMagnitude(sigma,x)
```

```
R = directionalMagnitude(sigma)
```

```
[value,pos] = min(R)
```

```
plot(sigma.directionalMagnitude)
annotate(pos)
plot(sigma,'minmax')
```





Visualization

For a second order tensor σ_{ij} its directional magnitude $R(\vec{x})$ is defined as

$$R(\vec{x}) = \sigma_{ij}\vec{x}_i\vec{x}_j.$$

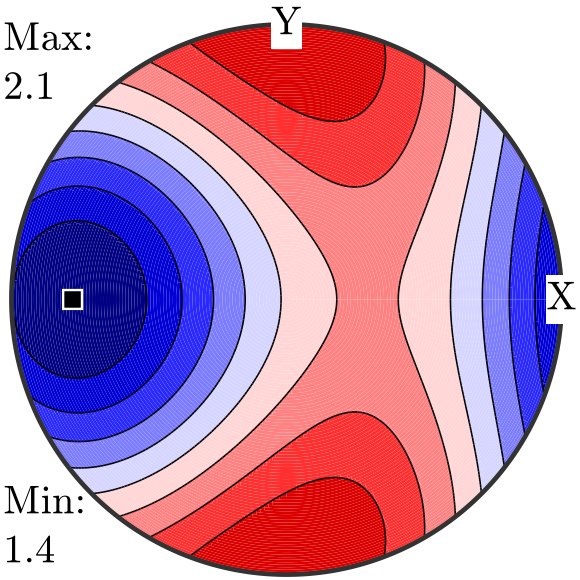
```
R = EinsteinSum(sigma,[-1 -2],...
    x,-1,x,-2)
```

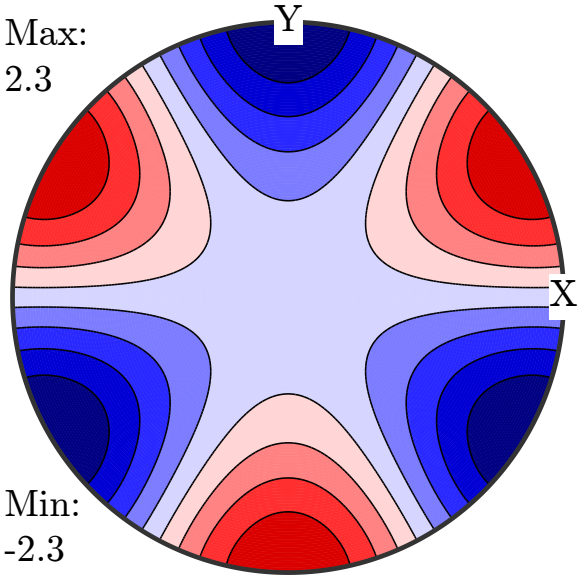
```
R = directionalMagnitude(sigma)
```

```
R = S2FunHarmonic (show methods, plot)
 bandwidth: 2
 antipodal: true
```

```
[value, pos] = min(R)
```

```
plot(sigma.directionalMagnitude)
annotate(pos)
plot(sigma,'minmax')
```





Visualization

For a second order tensor σ_{ij} its directional magnitude $R(\vec{x})$ is defined as

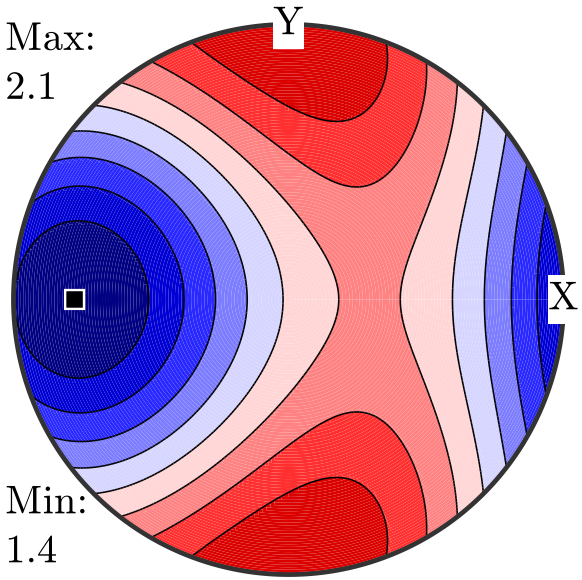
$$R(\vec{x}) = \sigma_{ij}\vec{x}_i\vec{x}_j.$$

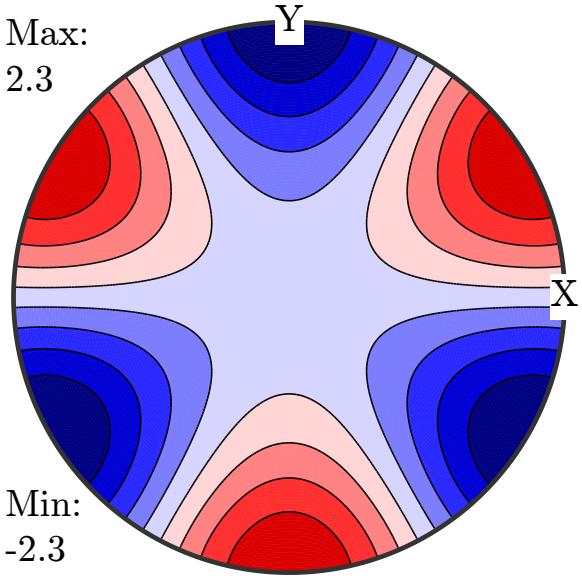
```
R = EinsteinSum(sigma,[-1 -2],...
    x,-1,x,-2)
```

```
R = directionalMagnitude(sigma)
```

```
[value,pos] = min(R)
```

```
value =
    1.3652

pos = vector3d
 size: 1 x 1
          x          y          z
  -0.918733          0   0.394879
```





Visualization

For a second order tensor σ_{ij} its directional magnitude $R(\vec{x})$ is defined as

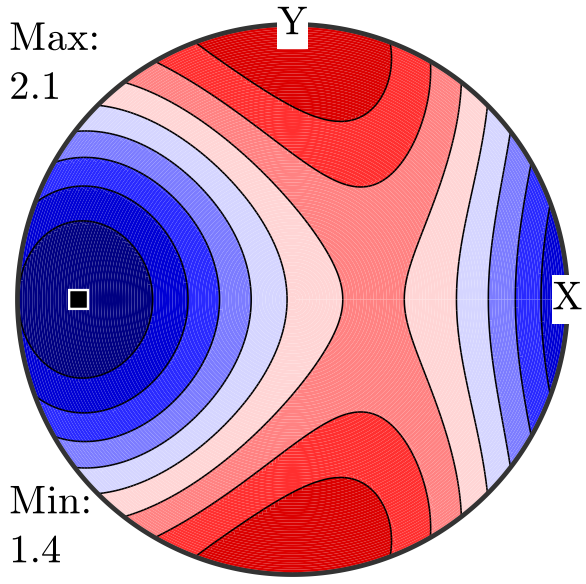
$$R(\vec{x}) = \sigma_{ij}\vec{x}_i\vec{x}_j.$$

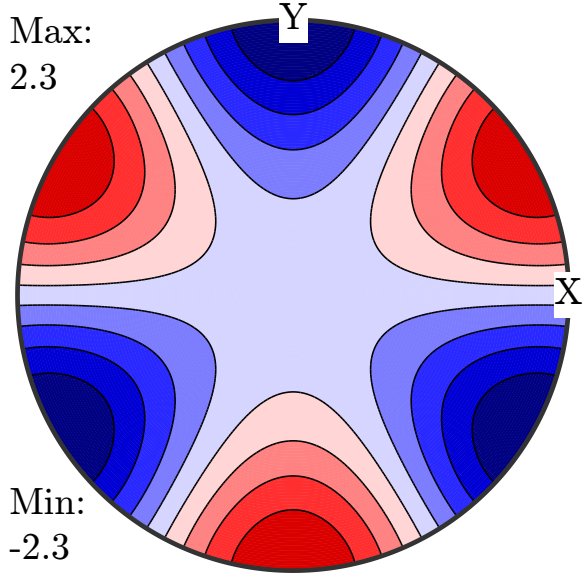
```
R = EinsteinSum(sigma,[-1 -2],...
  x,-1,x,-2)
```

```
R = directionalMagnitude(sigma)
```

```
[value,pos] = min(R)
```

```
plot(sigma.directionalMagnitude)
annotate(pos)
plot(sigma,'minmax')
mtexColorMap blue2red
```





Field Tensors vs. Matter Tensors

field tensors:

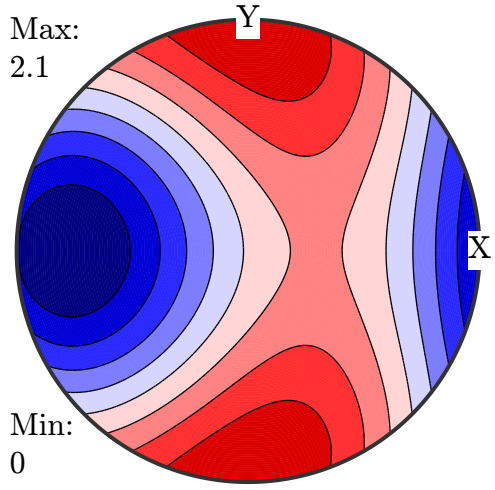
- in specimen coordinates
- describe applied forces like: stress, electric field

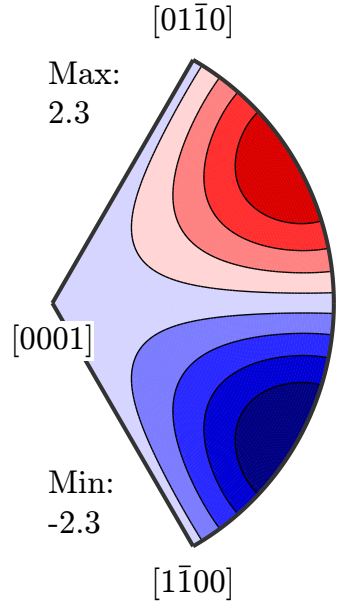
```
sigma = stressTensor(M);
```

matter tensors:

- in crystal coordinates
- describe physical properties like: electrical or thermal conductivity, magnetic permeability

```
CS = loadCif('Quartz')
P = piezoElectricityTensor(M,CS)
```





Field Tensors vs. Matter Tensors

field tensors:

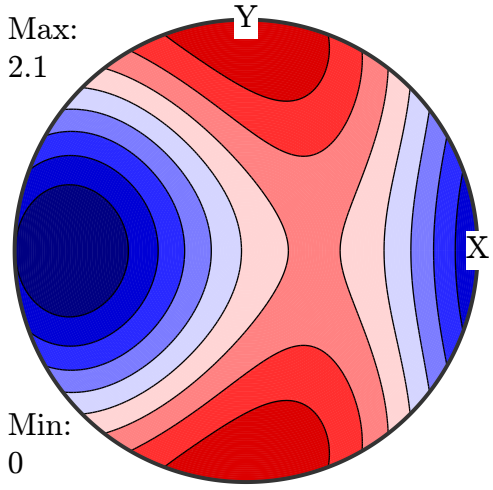
- in specimen coordinates
- describe applied forces like: stress, electric field

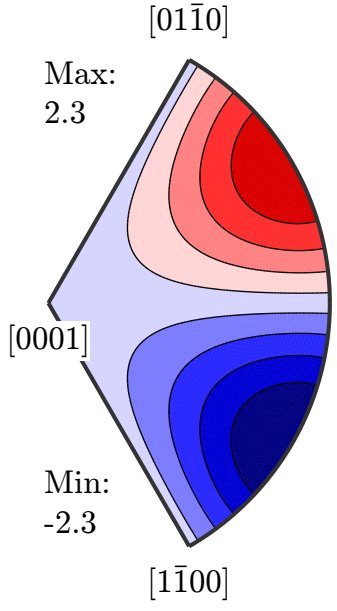
```
sigma = stressTensor(M);
```

matter tensors:

- in crystal coordinates
- describe physical properties like: electrical or thermal conductivity, magnetic permeability

```
CS = loadCif('Quartz')
P = piezoElectricityTensor(M,CS)
```





Rotating Tensors

Consider the piezoelectricity tensor

```
P=piezoElectricityTensor(M,CS)
```

```
P = piezoElectricityTensor (show methods, plot)
  unit                : C/N
  rank                : 3 (3 x 3 x 3)
  mineral             : Quartz (321, X||a*, Y||b, Z||c)

  tensor in compact matrix form:
     0      0      0 -0.67      0    4.6
   2.3   -2.3      0      0   0.67      0
     0      0      0      0      0      0
```

Remember orientations transforms crystal into specimen coordinates

```
ori = orientation('Euler',10*degree,20*degree,0,CS)
ori * P
```

Contrary, an inverse orientation transforms specimen coordinates into crystal coordinates.

```
inv(ori) * sigma
```

Rotating Tensors

Consider the piezoelectricity tensor

```
P=piezoElectricityTensor(M,CS)
```

Remember orientations transforms crystal into specimen coordinates

```
ori = orientation('Euler',10*degree,20*degree,0,CS)
ori * P
```

```
ans = piezoElectricityTensor (show methods, plot)
  unit                : C/N
  rank                : 3 (3 x 3 x 3)

  tensor in compact matrix form:
  -1.08   1.25  -0.17  -0.01   1.47   3.72
   1.92  -1.63  -0.29  -1.29   1.10   1.86
   0.76  -0.66  -0.09  -0.46   0.30   0.43
```

Contrary, an inverse orientation transforms specimen coordinates into crystal coordinates.

```
inv(ori) * sigma
```

Rotating Tensors

Consider the piezoelectricity tensor

```
P=piezoElectricityTensor(M,CS)
```

Remember orientations transforms crystal into specimen coordinates

```
ori = orientation('Euler',10*degree,20*degree,0,CS)
ori * P
```

Contrary, an inverse orientation transforms specimen coordinates into crystal coordinates.

```
inv(ori) * sigma
```

```
ans = stressTensor (show methods, plot)
  unit    : MPa
  rank    : 2 (3 x 3)
  mineral: Quartz (321, X||a*, Y||b, Z||c)

  1.47   0.17   0.14
  0.17   2.03  -0.12
  0.14  -0.12   1.85
```

Average Tensors

The average tensorial property of a specimen is the mean of matter tensors rotated according to each grain orientation o_m, $m = 1, \dots, M$.

The Voigt and the Reuss averages of a tensor T are defines as

$$\langle T \rangle^{\text{Voigt}} = \sum_{m=1}^{M} o_m \cdot T, \quad \langle T \rangle^{\text{Reuss}} = \left[\sum_{m=1}^{M} o_m \cdot T^{-1} \right]^{-1}.$$

For EBSD data this is computed by

```
[TVoigt, TReus, THill] = calcTensor(ebsd,T)
```

and for an ODF by

```
[TVoigt, TReus, THill] = calcTensor(odf,T)
```

Average Tensors

The average tensorial property of a specimen is the mean of matter tensors rotated according to each grain orientation o_m, $m = 1, \dots, M$.

The Voigt and the Reuss averages of a tensor T are defines as

$$\langle T \rangle^{\text{Voigt}} = \sum_{m=1}^{M} o_m \cdot T, \quad \langle T \rangle^{\text{Reuss}} = \left[\sum_{m=1}^{M} o_m \cdot T^{-1} \right]^{-1}.$$

For EBSD data this is computed by

```
[TVoigt, TReus, THill] = calcTensor(ebsd,T)
```

and for an ODF by

```
[TVoigt, TReus, THill] = calcTensor(odf,T)
```

Average Tensors

The average tensorial property of a specimen is the mean of matter tensors rotated according to each grain orientation o_m, $m = 1, \dots, M$.

The Voigt and the Reuss averages of a tensor T are defines as

$$\langle T \rangle^{\text{Voigt}} = \sum_{m=1}^{M} o_m \cdot T, \quad \langle T \rangle^{\text{Reuss}} = \left[\sum_{m=1}^{M} o_m \cdot T^{-1} \right]^{-1}.$$

For EBSD data this is computed by

```
[TVoigt, TReus, THill] = calcTensor(ebsd,T)
```

and for an ODF by

```
[TVoigt, TReus, THill] = calcTensor(odf,T)
```

Average Tensors

The average tensorial property of a specimen is the mean of matter tensors rotated according to each grain orientation o_m, $m = 1, \dots, M$.

The Voigt and the Reuss averages of a tensor T are defines as

$$\langle T \rangle^{\text{Voigt}} = \sum_{m=1}^{M} o_m \cdot T, \quad \langle T \rangle^{\text{Reuss}} = \left[\sum_{m=1}^{M} o_m \cdot T^{-1} \right]^{-1}.$$

For EBSD data this is computed by

```
[TVoigt, TReus, THill] = calcTensor(ebsd,T)
```

and for an ODF by

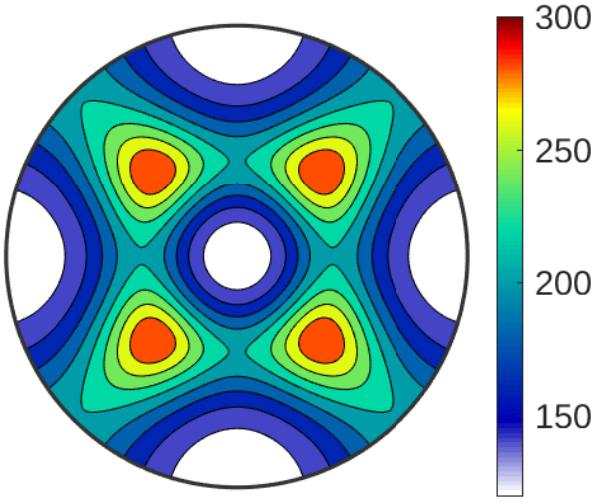
```
[TVoigt, TReus, THill] = calcTensor(odf,T)
```

Elastic Deformation

Imort some stiffness tensor

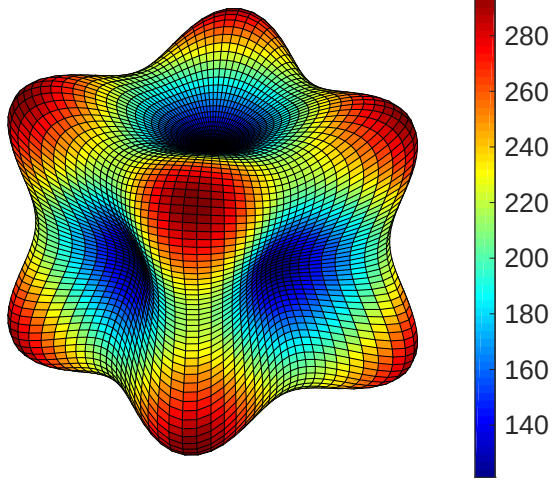
```
cs = loadCIF('Nickel')
C = stiffnessTensor.load(...
  'IN739LC.GPa',cs)
```

```
C = stiffnessTensor (show methods)
  unit   : GPa
  rank   : 4 (3 x 3 x 3 x 3)
  mineral: Ni (432)

  tensor in Voigt matrix representation:
 235.16 147.67 147.67      0      0      0
 147.67 235.16 147.67      0      0      0
 147.67 147.67 235.16      0      0      0
      0      0      0 122.53      0      0
      0      0      0      0 122.53      0
      0      0      0      0      0 122.53
```



The inverse of the stiffness is the compliance

```
S = inv(C)
```



Consider unaxial stress σ

Elastic Deformation

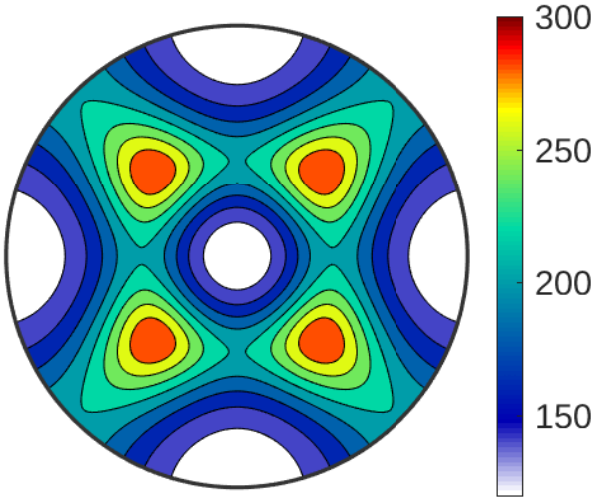
Imort some stiffness tensor

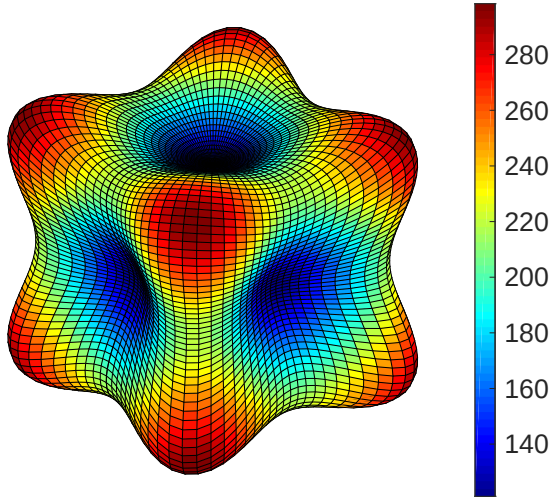
```
cs = loadCIF('Nickel')
C = stiffnessTensor.load(...
    'IN739LC.GPa',cs)
```

The inverse of the stiffness is the compliance

```
S = inv(C)
```

```
S = complianceTensor
  unit              : 1 / GPa
  rank              : 4 (3 x 3 x 3 x 3)
  doubleConvention: true
  mineral           : Ni (432)

  tensor in Voigt matrix represent.: *10^-4
  82.48 -31.82 -31.82      0      0      0
-31.82  82.48 -31.82      0      0      0
-31.82 -31.82  82.48      0      0      0
     0      0      0  81.61      0      0
     0      0      0      0  81.61      0
     0      0      0      0      0  81.61
```





Consider uniaxial stress σ

Elastic Deformation

Imort some stiffness tensor

```
cs = loadCIF('Nickel')
C = stiffnessTensor.load(...
    'IN739LC.GPa',cs)
```

The inverse of the stiffness is the compliance

```
S = inv(C)
```

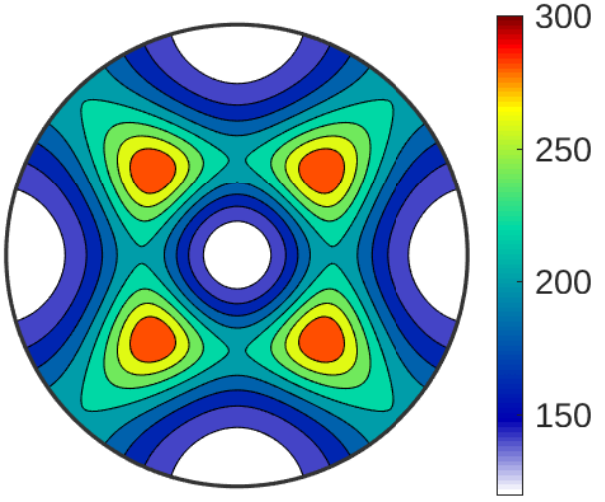
Consider unaxial stress σ

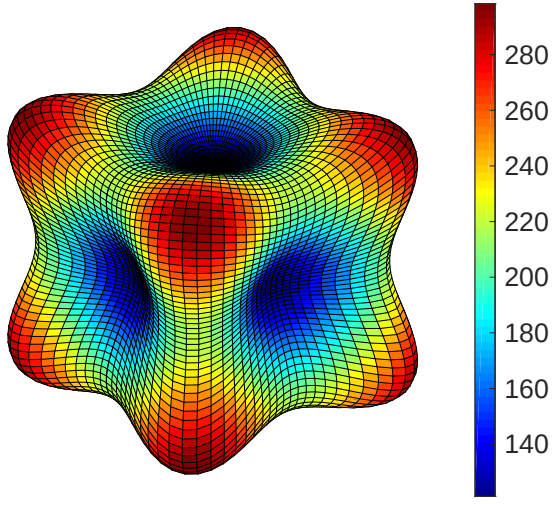
```
d = vector3d.Z;
sigma = stressTensor.uniaxial(d)
```

```
sigma = stressTensor (show methods, plot)
  rank: 2 (3 x 3)

 0 0 0
 0 0 0
 0 0 1
```

strain $\varepsilon_{ij} = S_{klij}\sigma_{kl}$

```
eps = (ori * S) * sigma
```





Elastic Deformation

Imort some stiffness tensor

```
cs = loadCIF('Nickel')
C = stiffnessTensor.load(...
    'IN739LC.GPa',cs)
```

The inverse of the stiffness is the compliance

```
S = inv(C)
```

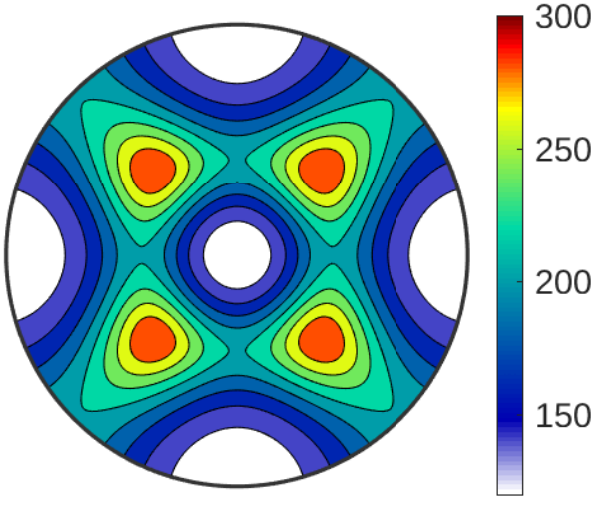
Consider unaxial stress σ

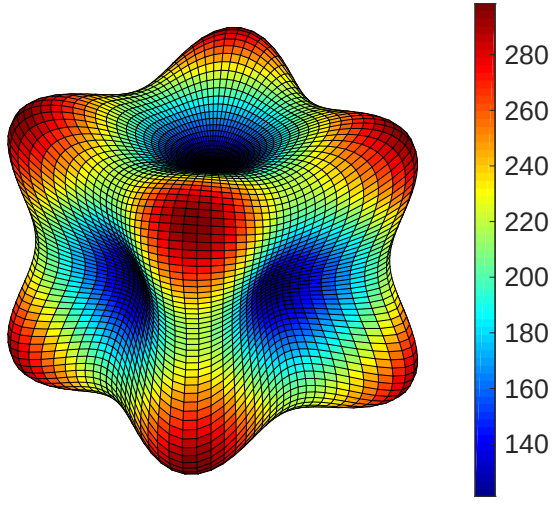
```
d = vector3d.Z;
sigma = stressTensor.uniaxial(d)
```

strain $\varepsilon_{ij} = S_{klij}\sigma_{kl}$

```
eps = (ori * S) * sigma
```

```
eps = strainTensor (show methods, plot)
  rank: 2 (3 x 3)

 147.67       0       0
      0  147.67       0
      0       0  235.16
```





Elasticity Tensors

derived elastic properties

```
beta = C.volumeCompressibility
beta = C.linearCompressibility
E    = C.YoungsModulus
G    = C.shearModulus(h,u)
nu   = C.PoissonRatio(x,y)
T    = C.ChristoffelTensor(n)
```

```
nu = C.PoissonRatio([],xvector);
contourf(nu,'minmax')
mtexColorMap blue2red
```

```
[value,pos] = max(nu)
```

```
annotate(pos,'label','max')
```

Elasticity Tensors

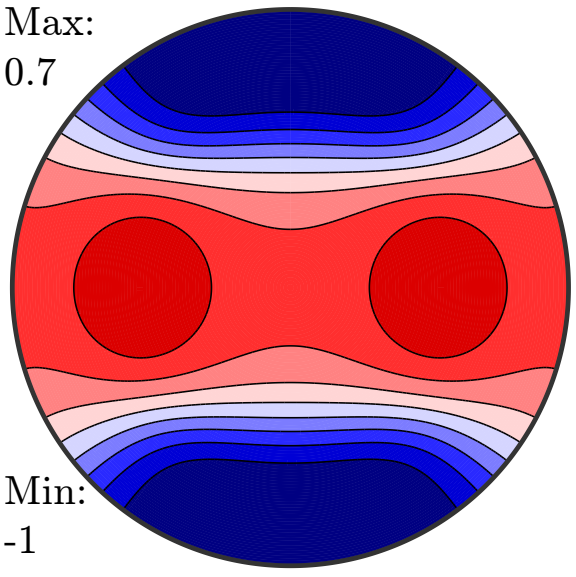
derived elastic properties

```
beta = C.volumeCompressibility
beta = C.linearCompressibility
E    = C.YoungsModulus
G    = C.shearModulus(h,u)
nu   = C.PoissonRatio(x,y)
T    = C.ChristoffelTensor(n)
```

```
nu = C.PoissonRatio([],xvector);
contourf(nu,'minmax')
mtexColorMap blue2red
```

```
[value,pos] = max(nu)
```

```
annotate(pos,'label','max')
```



Elasticity Tensors

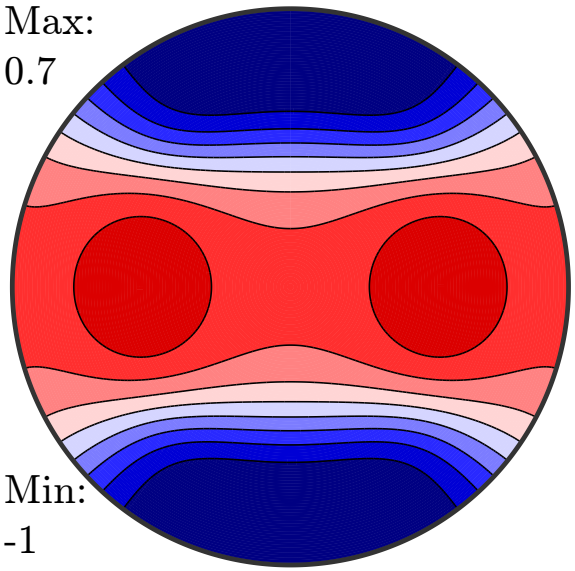
derived elastic properties

```
beta = C.volumeCompressibility
beta = C.linearCompressibility
E    = C.YoungsModulus
G    = C.shearModulus(h,u)
nu   = C.PoissonRatio(x,y)
T    = C.ChristoffelTensor(n)
```

```
nu = C.PoissonRatio([],xvector);
contourf(nu,'minmax')
mtexColorMap blue2red
```

```
[value,pos] = max(nu)
```

```
value =
    0.6956
pos = vector3d
 size: 1 x 1
          x          y          z
  -0.918733          0   0.394879
```



Elasticity Tensors

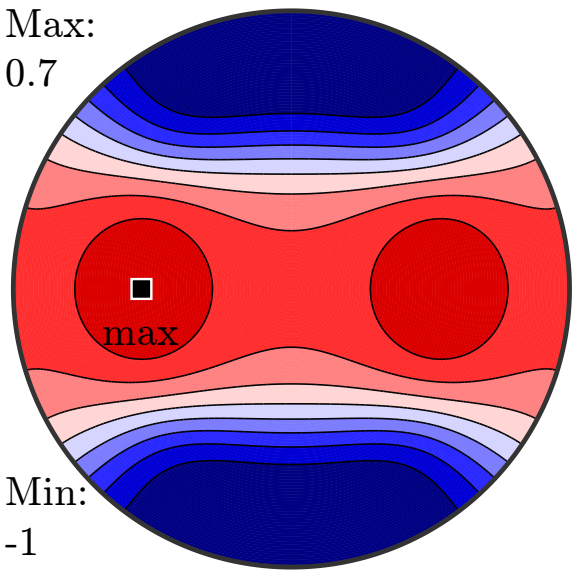
derived elastic properties

```
beta = C.volumeCompressibility
beta = C.linearCompressibility
E    = C.YoungsModulus
G    = C.shearModulus(h,u)
nu   = C.PoissonRatio(x,y)
T    = C.ChristoffelTensor(n)
```

```
nu = C.PoissonRatio([],xvector);
contourf(nu,'minmax')
mtexColorMap blue2red
```

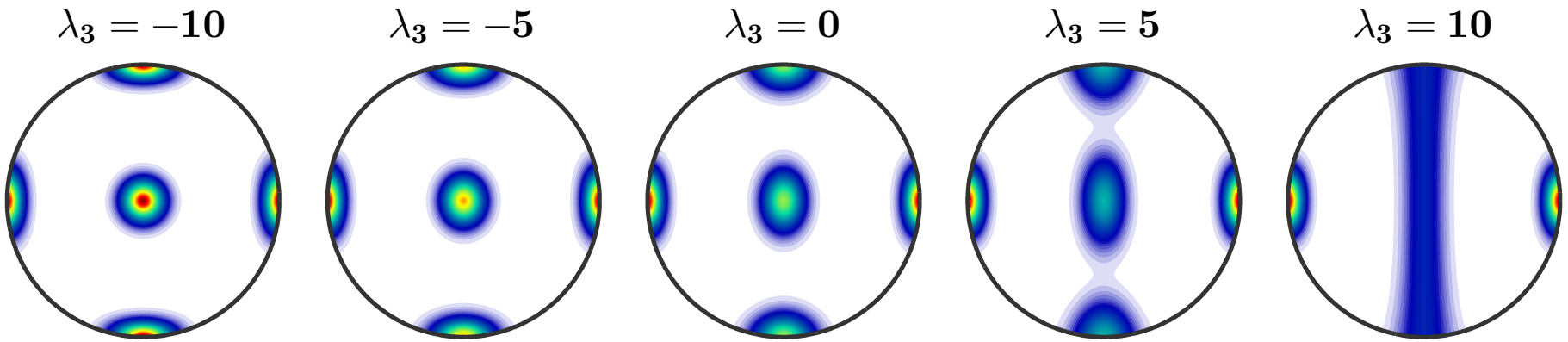
```
[value,pos] = max(nu)
```

```
annotate(pos,'label','max')
```



Evolution of Elasticity

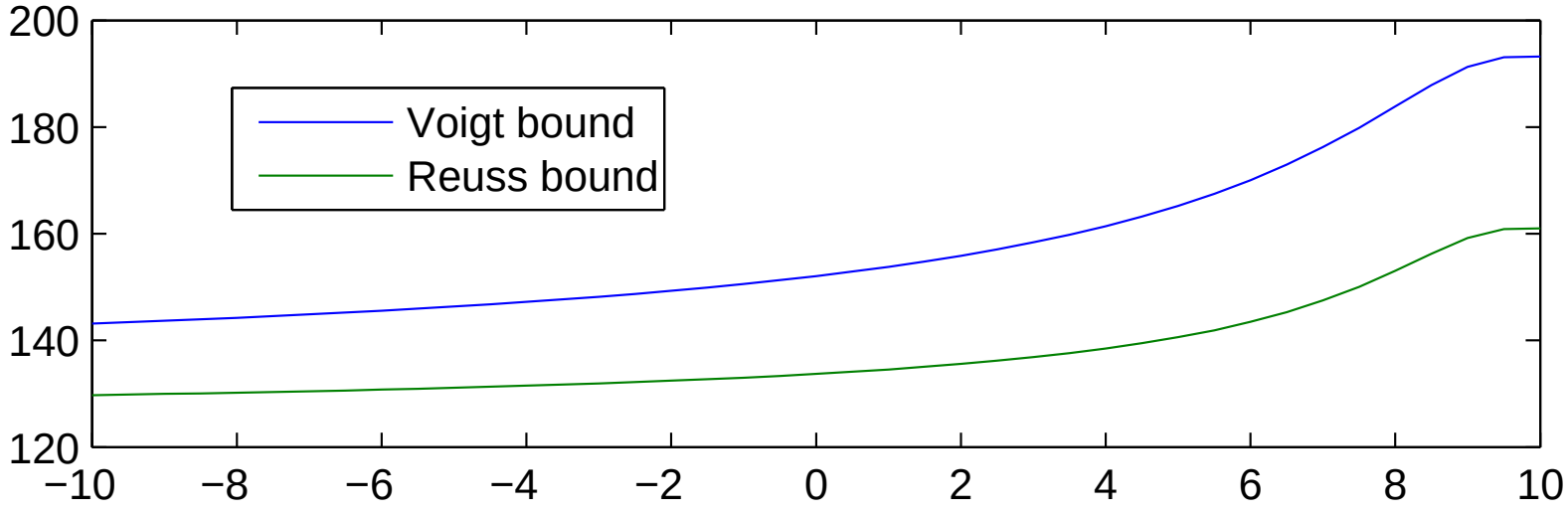
```
for k = 1:5
  odf{k} = BinghamODF([-10,-10,5*k-15,10],cs);
end
```



Evolution of Elasticity

```
for k = 1:5
  odf{k} = BinghamODF([-10,-10,5*k-15,10],cs);
end
```

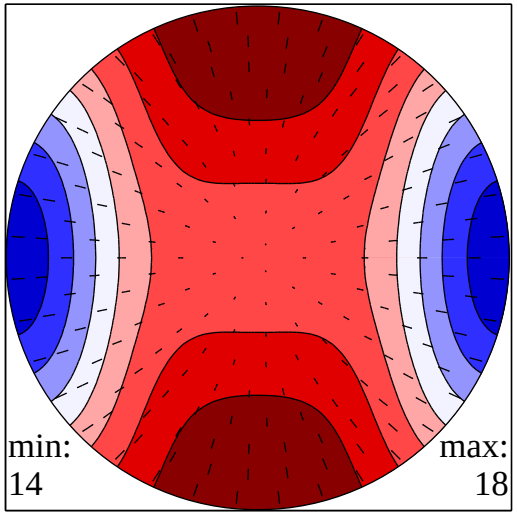
```
for k = 1:length(odf)
  [C_v,C_r] = calcTensor(odf{k},C);
  psr_v(k) = C_v.YoungsModulus(vector3d.Z);
  psr_r(k) = C_r.YoungsModulus(vector3d.Z);
end
```



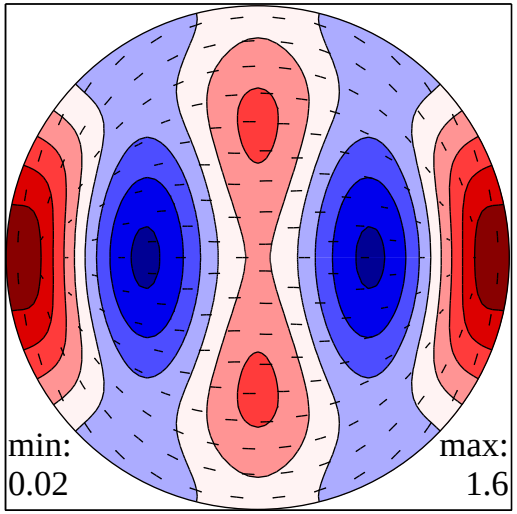
Wave Velocities

```
[vp,vs1,vs2,pp,ps1,ps2] = velocity(C)
```

```
plot(vp,'minmax')

hold on
plot(pp)
hold off
```



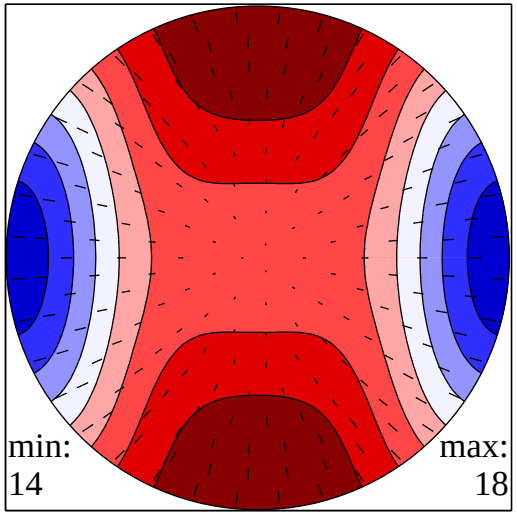

```
nextAxis % generate a new axis

plot(vs1-vs2,'minmax')

hold on
plot(ps1)
hold off
```



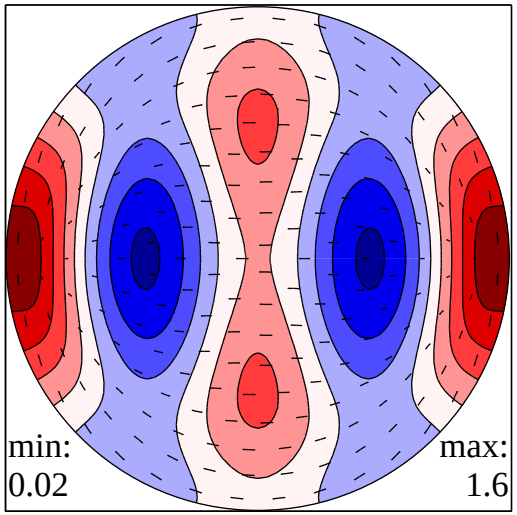
Wave Velocities

```
[vp,vs1,vs2,pp,ps1,ps2] = velocity(C)
```

```
plot(vp,'minmax')

hold on
plot(pp)
hold off
```




```
nextAxis % generate a new axis

plot(vs1-vs2,'minmax')

hold on
plot(ps1)
hold off
```

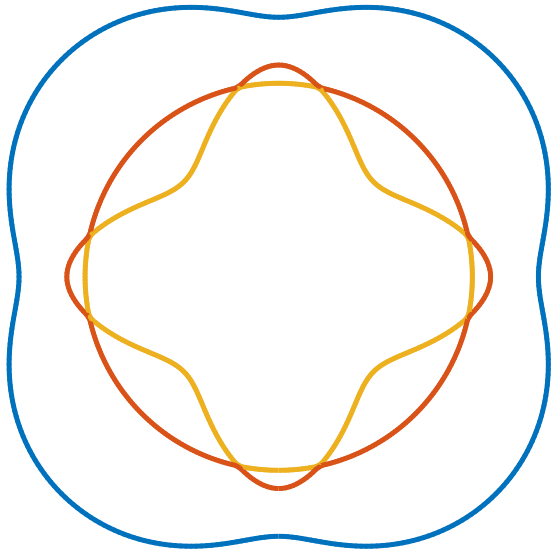


Slowness Surface

```
[vp,vs1,vs2,pp,ps1,ps2] = velocity(C)
```

The wavefront in the plane perpendicular to **z**

```
pN = vector3d.Z
plotSection(vp,pN)
hold on
plotSection(vs1,pN)
plotSection(vs2,pN)
hold off
```



The slowness surface perpendicular to **z**

```
plotSection(1./vp,pN)
hold on
plotSection(1./vs1,pN)
plotSection(1./vs2,pN)
hold off
```

Slowness Surface

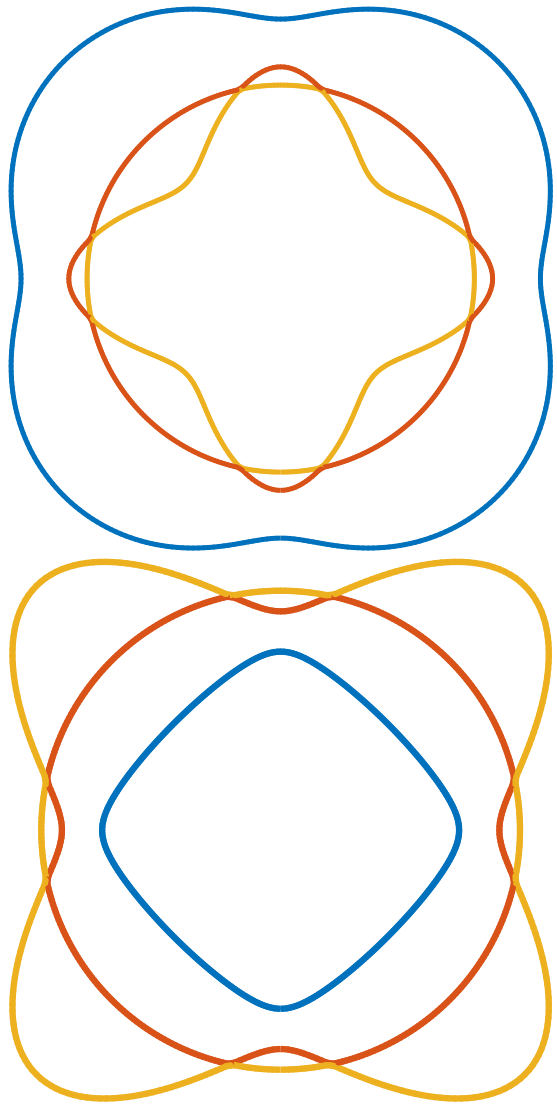
```
[vp,vs1,vs2,pp,ps1,ps2] = velocity(C)
```

The wavefront in the plane perpendicular to **z**

```
pN = vector3d.Z
plotSection(vp,pN)
hold on
plotSection(vs1,pN)
plotSection(vs2,pN)
hold off
```

The slowness surface perpendicular to **z**

```
plotSection(1./vp,pN)
hold on
plotSection(1./vs1,pN)
plotSection(1./vs2,pN)
hold off
```



Vertical and Horizontal Velocities

Vertical and horizontal velocities

```
id = angle(ps1,pN) <= 89.9*degree;
vsv = id .* vs1 + (1-id) .* vs2;
vsh = id .* vs2 + (1-id) .* vs1;
```

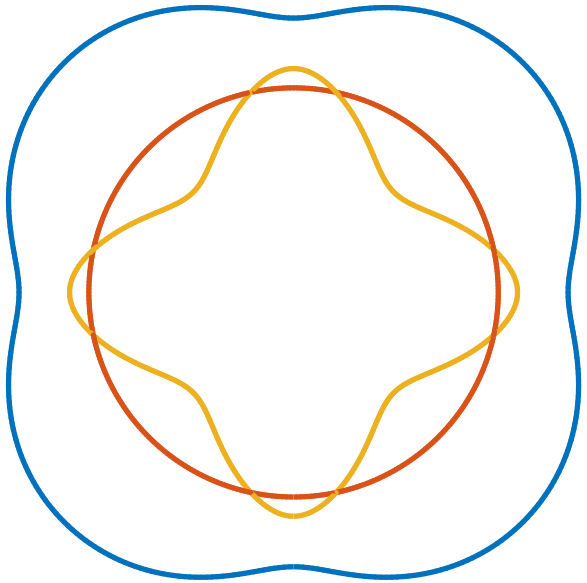
Vertical and Horizontal Velocities

Vertical and horizontal velocities

```
id = angle(ps1,pN) <= 89.9*degree;
vsv = id .* vs1 + (1-id) .* vs2;
vsh = id .* vs2 + (1-id) .* vs1;
```

The wavefront in the plane perpendicular to **z**

```
plotSection(vp,pN)
hold on
plotSection(vsv,pN)
plotSection(vsh,pN)
```



Vertical and Horizontal Velocities

Vertical and horizontal velocities

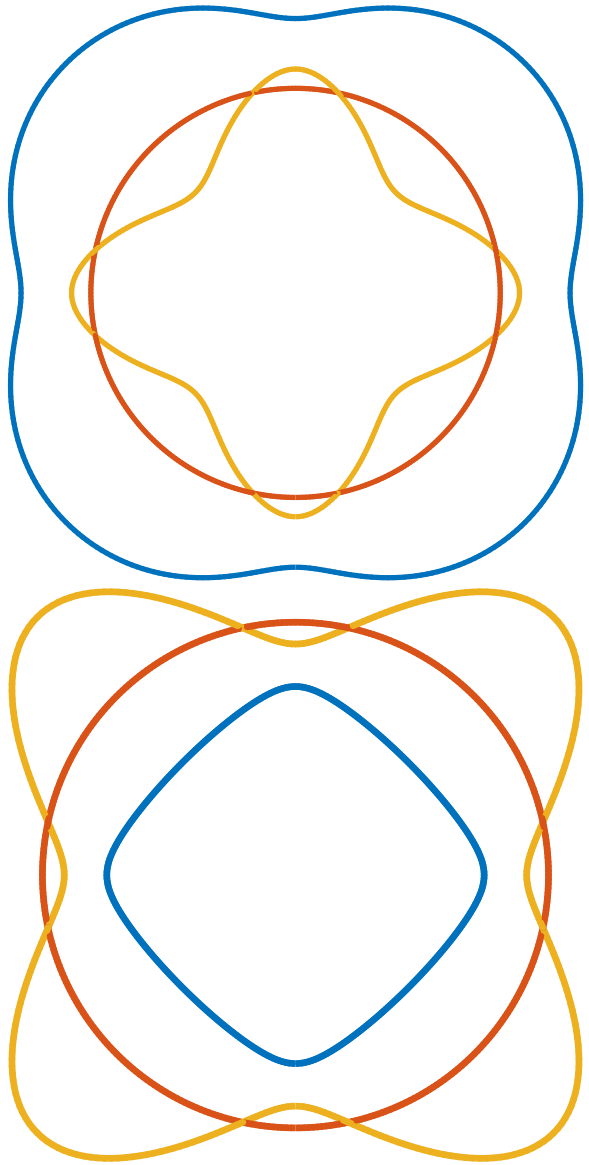
```
id = angle(ps1,pN) <= 89.9*degree;
vsv = id .* vs1 + (1-id) .* vs2;
vsh = id .* vs2 + (1-id) .* vs1;
```

The wavefront in the plane perpendicular to **z**

```
plotSection(vp,pN)
hold on
plotSection(vsv,pN)
plotSection(vsh,pN)
```

The slowness surface perpendicular to **z**

```
plotSection(1./vp,pN)
hold on
plotSection(1./vsv,pN)
plotSection(1./vsh,pN)
```



Energy Velocity Vector

Definition

$$E_i = \frac{v}{\rho} C_{ijkl} p_j p_l x_k$$

- C stiffness tensor
- x propagation direction
- v wave velocity in x
- p polarization in x
- ρ density

Energy Velocity Vector

Definition

$$E_i = \frac{v}{\rho} C_{ijkl} p_j p_l x_k$$

```
Evp = energyVector(C,[],vp,pp)
Esv = energyVector(C,[],vsv,psv)
Esh = energyVector(C,[],vsh,psh)
```

- C stiffness tensor
- x propagation direction
- v wave velocity in x
- p polarization in x
- ρ density

Energy Velocity Vector

Definition

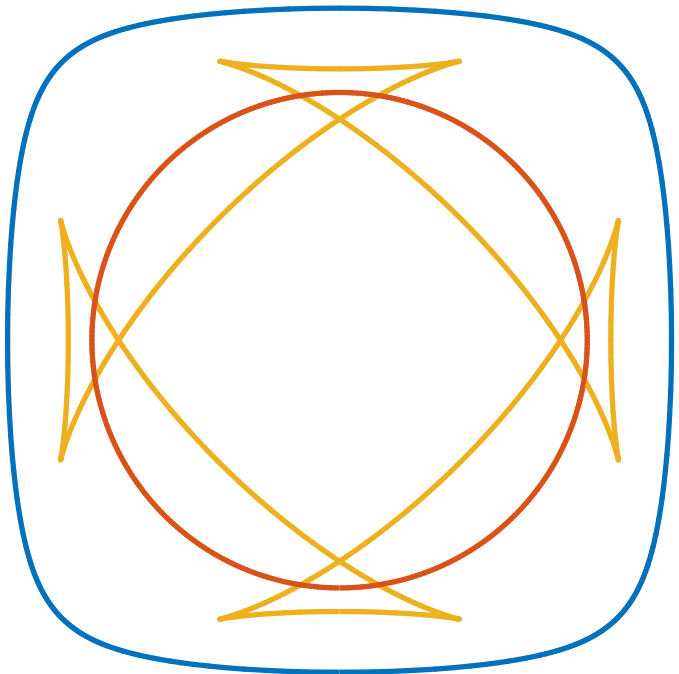
$$E_i = \frac{v}{\rho} C_{ijkl} p_j p_l x_k$$

- C stiffness tensor
- x propagation direction
- v wave velocity in x
- p polarization in x
- ρ density

```
Evp = energyVector(C,[],vp,pp)
Esv = energyVector(C,[],vsv,psv)
Esh = energyVector(C,[],vsh,psh)
```

The energy vector perpendicular to **z**

```
plotSection(Evp,pN)
hold on
plotSection(Esv,pN)
plotSection(Esh,pN)
hold off
```



Schmid Factor

Consider the slip system

```
d = Miller(0,-1,1,cs,'uvw');   % slip direction
n = Miller(1,1,1,cs,'hkl');    % slip plane normal
```

and the extension direction

```
r = vector3d(0,0,1)            % extension direction
```

Then the Schmid factor in direction **r** is given by

$$SF = \cos\theta \cos\rho$$

In **MTEX** this can be computed by

```
SF = dot(n,r) * dot(b,r)
```

Schmid Factor

Consider the slip system

```
d = Miller(0,-1,1,cs,'uvw');   % slip direction
n = Miller(1,1,1,cs,'hkl');    % slip plane normal
```

and the extension direction

```
r = vector3d(0,0,1)            % extension direction
```

Then the Schmid factor in direction **r** is given by

$$SF = \cos\theta \cos\rho$$

In **MTEX** this can be computed by

```
SF = dot(n,r) * dot(b,r)
```

Schmid Factor

Consider the slip system

```
d = Miller(0,-1,1,cs,'uvw');   % slip direction
n = Miller(1,1,1,cs,'hkl');    % slip plane normal
```

and the extension direction

```
r = vector3d(0,0,1)            % extension direction
```

Then the Schmid factor in direction **r** is given by

$$SF = \cos\theta \cos\rho$$

In MTEX this can be computed by

```
SF = dot(n,r) * dot(b,r)
```

Schmid Factor

Consider the slip system

```
d = Miller(0,-1,1,cs,'uvw');   % slip direction
n = Miller(1,1,1,cs,'hkl');    % slip plane normal
```

and the extension direction

```
r = vector3d(0,0,1)            % extension direction
```

Then the Schmid factor in direction **r** is given by

$$SF = \cos\theta \cos\rho$$

In **MTEX** this can be computed by

```
SF = dot(n,r) * dot(b,r)
```

Slip Systems

```
sS = slipSystem(d,n)
```

```
sS = slipSystem
 symmetry: 432
 CRSS: 1
 size: 1 x 1
  u    v    w | h    k    l
  0   -1    1    1    1    1
```

```
sS = slipSystem.fcc(cs)
```

```
sS.symmetrise('antipodal')
```

```
sS.SchmidFactor(vector3d.Z)
```

```
plot(sS.SchmidFactor)
```

```
M = [1 0 1;0 0 0;1 0 1];
sigma = stressTensor(M)
sS.SchmidFactor(sigma)
```

Slip Systems

```
sS = slipSystem(d,n)
```

```
sS = slipSystem.fcc(cs)
```

```
sS = slipSystem
 symmetry: 432
 CRSS: 1
 size: 1 x 1
  u    v    w | h    k    l
  0    1   -1    1    1    1
```

```
sS.symmetrise('antipodal')
```

```
sS.SchmidFactor(vector3d.Z)
```

```
plot(sS.SchmidFactor)
```

```
M = [1 0 1;0 0 0;1 0 1];
sigma = stressTensor(M)
sS.SchmidFactor(sigma)
```

Slip Systems

```
sS = slipSystem(d,n)
```

```
sS = slipSystem.fcc(cs)
```

```
sS.symmetrise('antipodal')
```

```
sS = slipSystem
 symmetry: 432
 CRSS: 1
 size: 12 x 1
   u    v    w | h    k    l
   0   -1    1    1    1    1
   1    0   -1    1    1    1
  -1    1    0    1    1    1
  -1    1    0    1    1   -1
  -1    0   -1    1    1   -1
   0   -1   -1    1    1   -1
   0   -1    1   -1    1    1
  -1    0   -1   -1    1    1
  -1   -1    0   -1    1    1
   1    0   -1    1   -1    1
  -1   -1    0    1   -1    1
   0   -1   -1    1   -1    1
```

Slip Systems

```
sS = slipSystem(d,n)
```

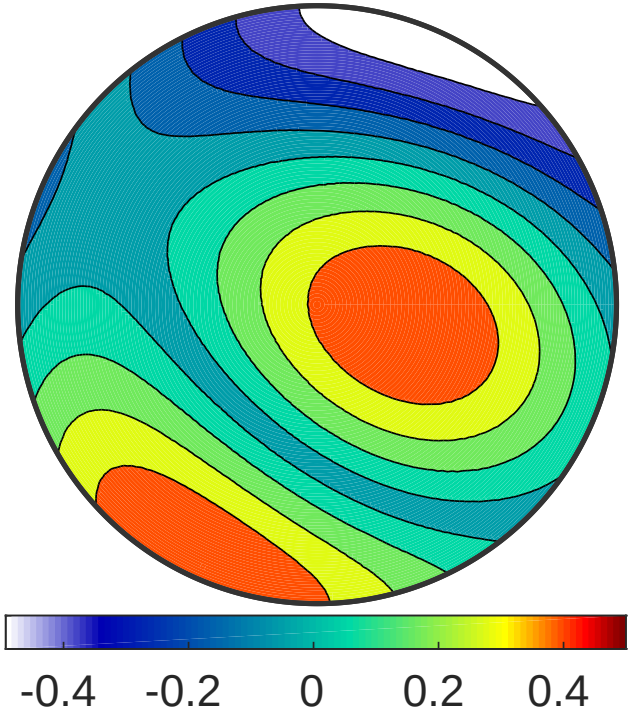
```
sS = slipSystem.fcc(cs)
```

```
sS.symmetrise('antipodal')
```

```
sS.SchmidFactor(vector3d.Z)
```

```
plot(sS.SchmidFactor)
```

```
M = [1 0 1;0 0 0;1 0 1];
sigma = stressTensor(M)
sS.SchmidFactor(sigma)
```



Slip Systems

```
sS = slipSystem(d,n)
```

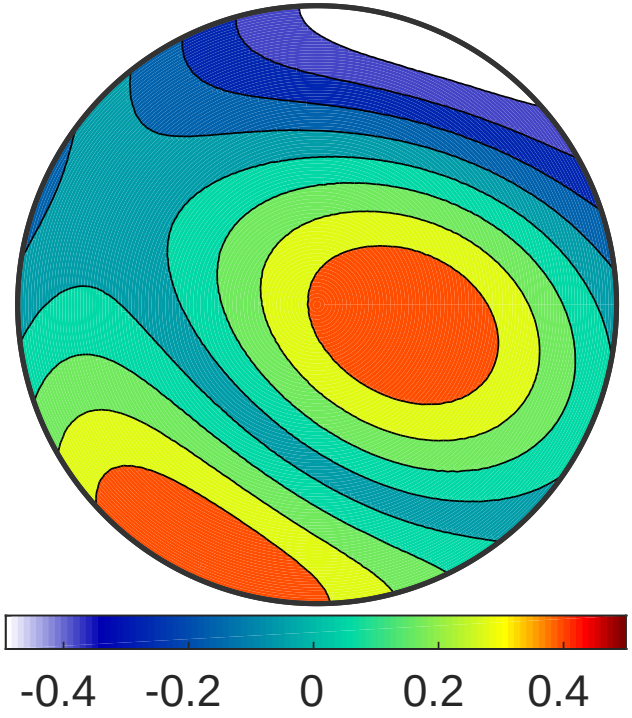
```
sS = slipSystem.fcc(cs)
```

```
sS.symmetrise('antipodal')
```

```
sS.SchmidFactor(vector3d.Z)
```

```
plot(sS.SchmidFactor)
```

```
M = [1 0 1;0 0 0;1 0 1];
sigma = stressTensor(M)
sS.SchmidFactor(sigma)
```



Slip Systems

```
sS = slipSystem(d,n)
```

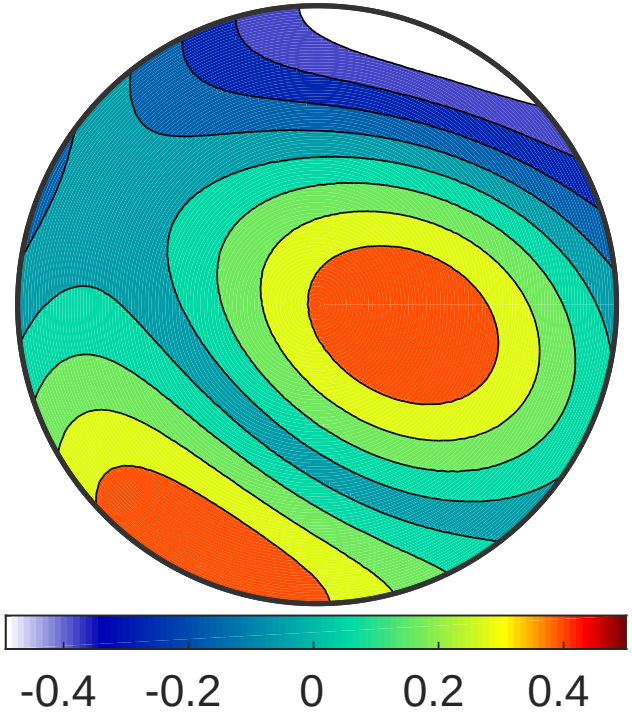
```
sS = slipSystem.fcc(cs)
```

```
sS.symmetrise('antipodal')
```

```
sS.SchmidFactor(vector3d.Z)
```

```
plot(sS.SchmidFactor)
```

```
M = [1 0 1;0 0 0;1 0 1];
sigma = stressTensor(M)
sS.SchmidFactor(sigma)
```



Maximum Schmid Factor

the Schmid factor for a list of tension directions

```
sS = sS.symmetrise('antipodal')
r = plotS2Grid('upper')
SF = sS.SchmidFactor(r)
```

SF is `length(r)` × `length(sS)` matrix
maximum Schmidfactor over the second dimension

```
[maxSF,id] = max(abs(SF),[],2)
contourf(r,maxSF)
```

the active slip system

```
contourf(r,id,'contours',12)
hold on
quiver(r,sS(id).n,'Color','r');
quiver(r,sS(id).b,'Color','g');
hold off
```

Maximum Schmid Factor

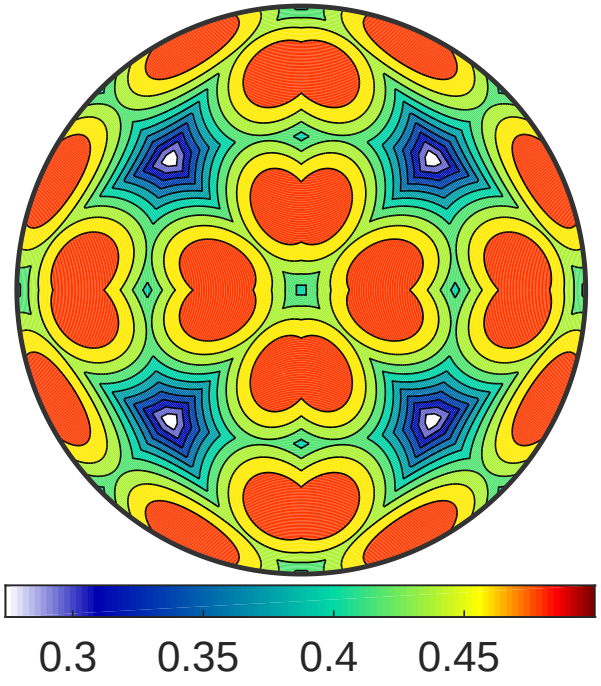
the Schmid factor for a list of tension directions

```
sS = sS.symmetrise('antipodal')
r = plotS2Grid('upper')
SF = sS.SchmidFactor(r)
```

SF is `length(r)` $\times$ `length(sS)` matrix

maximum Schmidfactor over the second dimension

```
[maxSF,id] = max(abs(SF),[],2)
contourf(r,maxSF)
```



the active slip system

```
contourf(r,id,'contours',12)
hold on
quiver(r,sS(id).n,'Color','r');
quiver(r,sS(id).b,'Color','g');
hold off
```

Maximum Schmid Factor

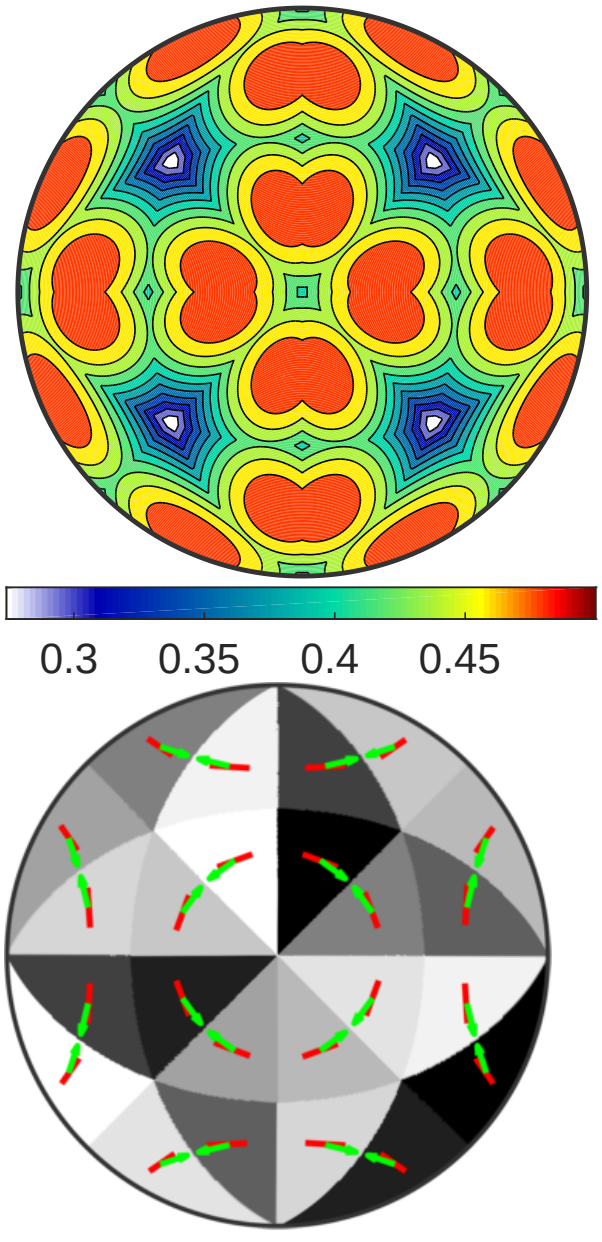
the Schmid factor for a list of tension directions

```
sS = sS.symmetrise('antipodal')
r = plotS2Grid('upper')
SF = sS.SchmidFactor(r)
```

SF is `length(r)` × `length(sS)` matrix

maximum Schmidfactor over the second dimension

```
[maxSF,id] = max(abs(SF),[],2)
contourf(r,maxSF)
```



the active slip system

```
contourf(r,id,'contours',12)
hold on
quiver(r,sS(id).n,'Color','r');
quiver(r,sS(id).b,'Color','g');
hold off
```

Stress Based Analysis - Route 1

```
sS = symmetrise(slipSystem.fcc(ebsd.CS))
```

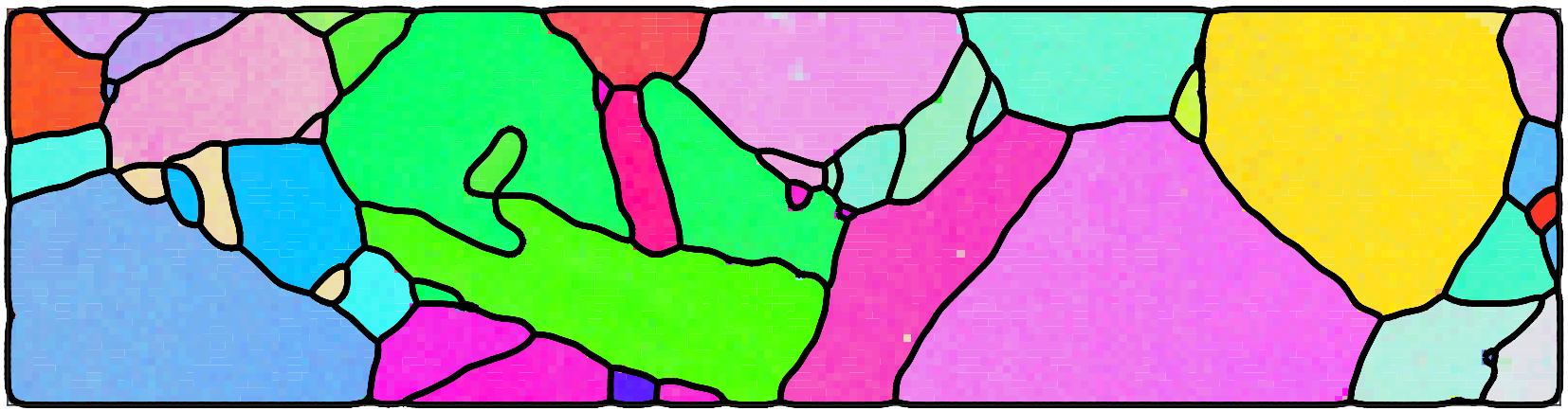
```
sS = slipSystem (show methods, plot)
 mineral: iron (m-3m)
 size: 24 x 1
   u    v    w | h    k    l
   0    1   -1    1    1    1
  -1    0    1    1    1    1
   1   -1    0    1    1    1
   0   -1    1    1    1    1
   1    0   -1    1    1    1
  -1    1    0    1    1    1
   1   -1    0    1    1   -1
   1    0    1    1    1   -1
   0    1    1    1    1   -1
  -1    0   -1    1    1   -1
   0   -1   -1    1    1   -1
  -1    1    0    1    1   -1
   0    1   -1   -1    1    1
   1    0    1   -1    1    1
   1    1    0   -1    1    1
  -1    0   -1   -1    1    1
  -1   -1    0   -1    1    1
   0   -1    1   -1    1    1
```

Stress Based Analysis - Route 1

```
sS = symmetrise(slipSystem.fcc(ebsd.CS))
```

```
sSGrain = grains.meanOrientation * sS
```

```
sSGrain = slipSystem (show methods, plot)
 size: 71 x 24
```

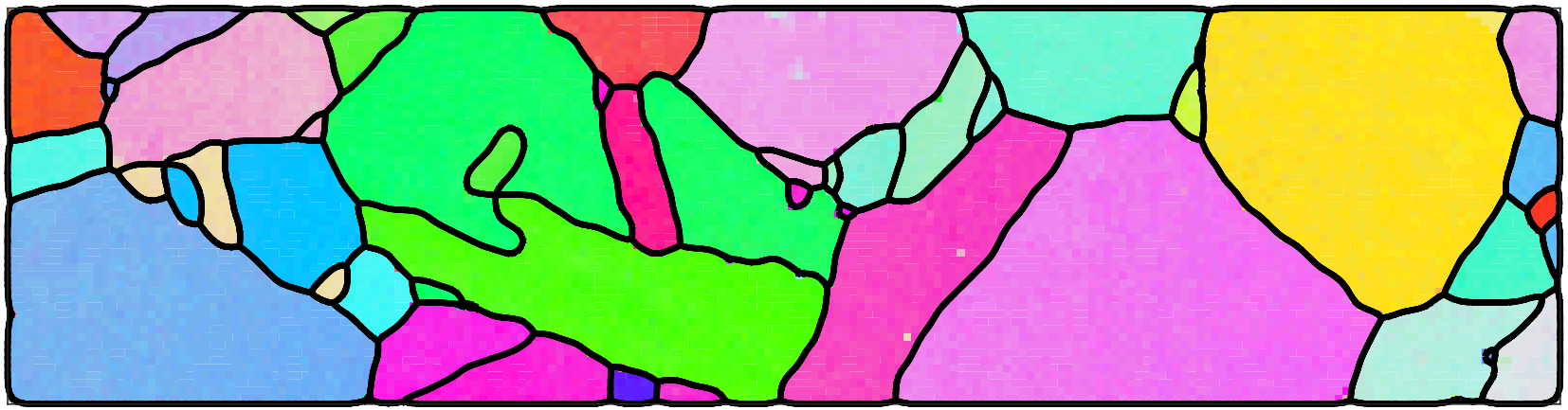


Stress Based Analysis - Route 1

```
sS = symmetrise(slipSystem.fcc(ebsd.CS))
```

```
sSGrain = grains.meanOrientation * sS
```

```
SF = sSGrain.SchmidFactor(sigma);
```



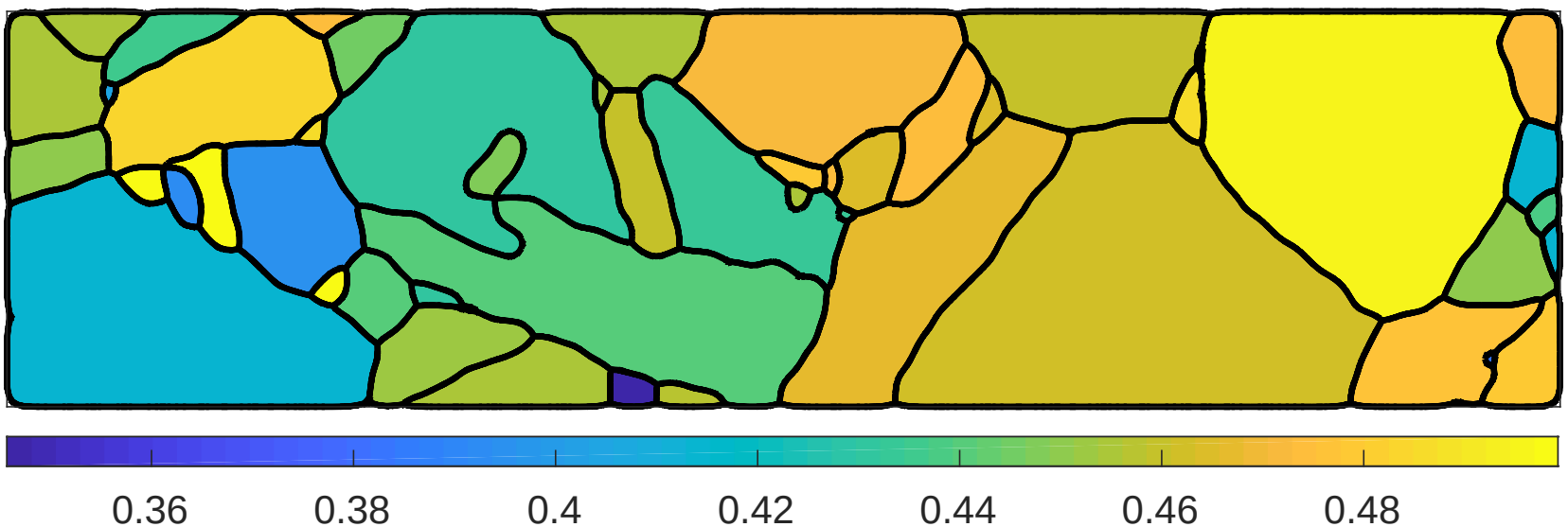
Stress Based Analysis - Route 1

```
sS = symmetrise(slipSystem.fcc(ebsd.CS))
```

```
sSGrain = grains.meanOrientation * sS
```

```
SF = sSGrain.SchmidFactor(sigma);
```

```
[maxSF,active] = max(SF,[],2);
plot(grains,maxSF)
```



Stress Based Analysis - Route 1

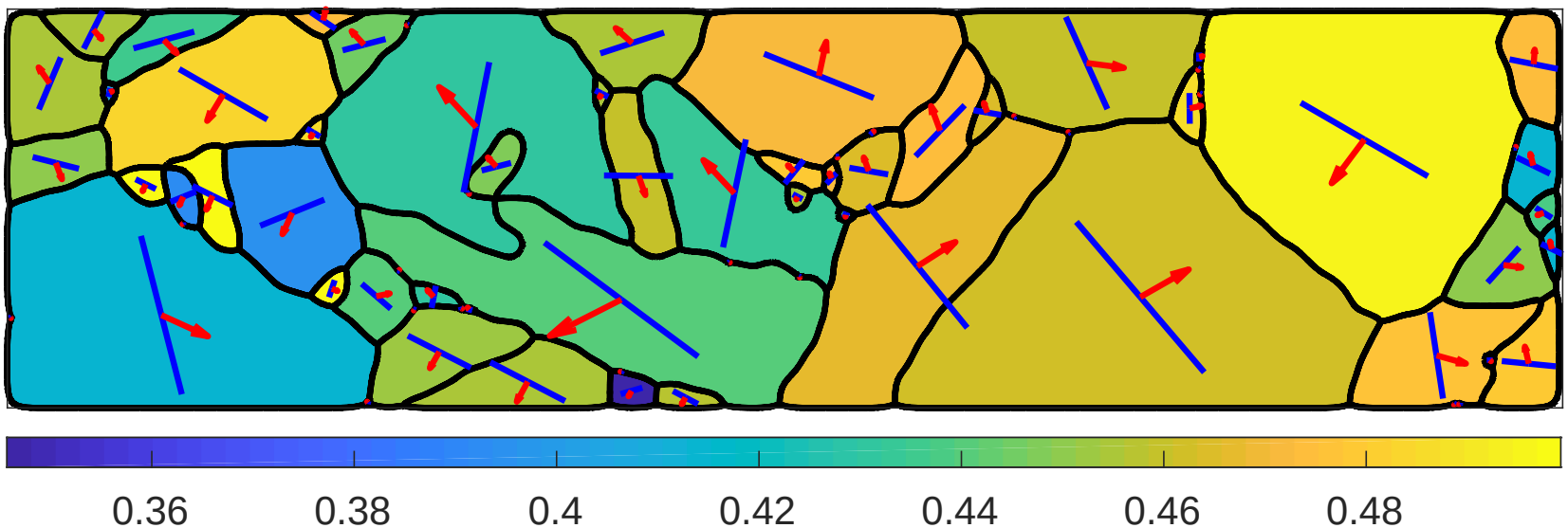
```
sS = symmetrise(slipSystem.fcc(ebsd.CS))
```

```
sSGrain = grains.meanOrientation * sS
```

```
SF = sSGrain.SchmidFactor(sigma);
```

```
[maxSF,active] = max(SF,[],2);
plot(grains,maxSF)
```

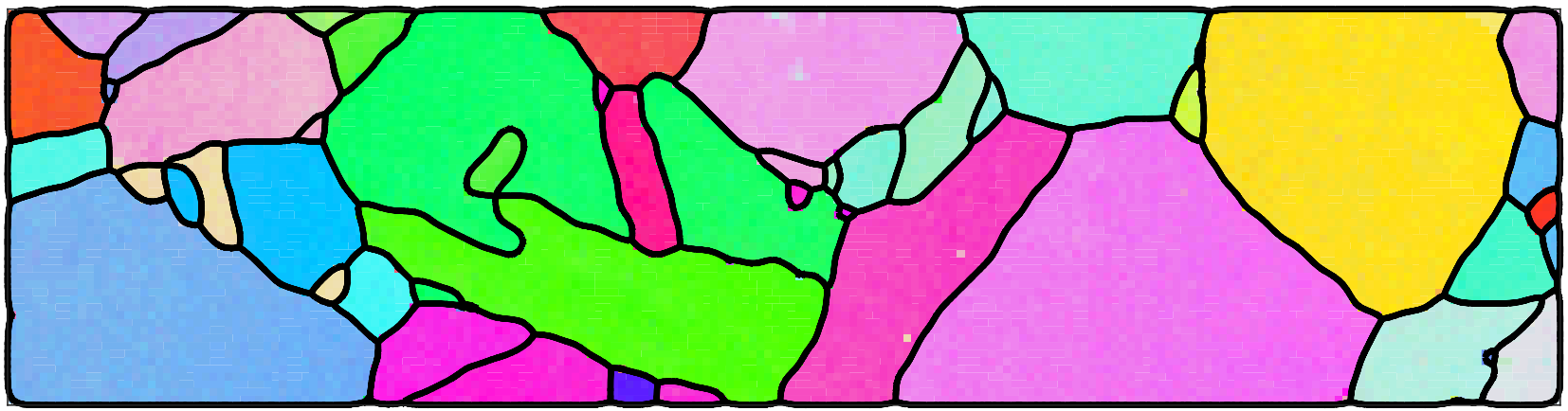
```
sSactive = grains.meanOrientation .* sS(active);
quiver(grains,sSactive.trace,'color','b')
quiver(grains,sSactive.b,'color','r')
```



Stress Based Analysis - Route 2

```
sigma = stressTensor.uniaxial(vector3d.X)
```

```
sigma = stressTensor (show methods, plot)
  rank: 2 (3 x 3)

 1 0 0
 0 0 0
 0 0 0
```

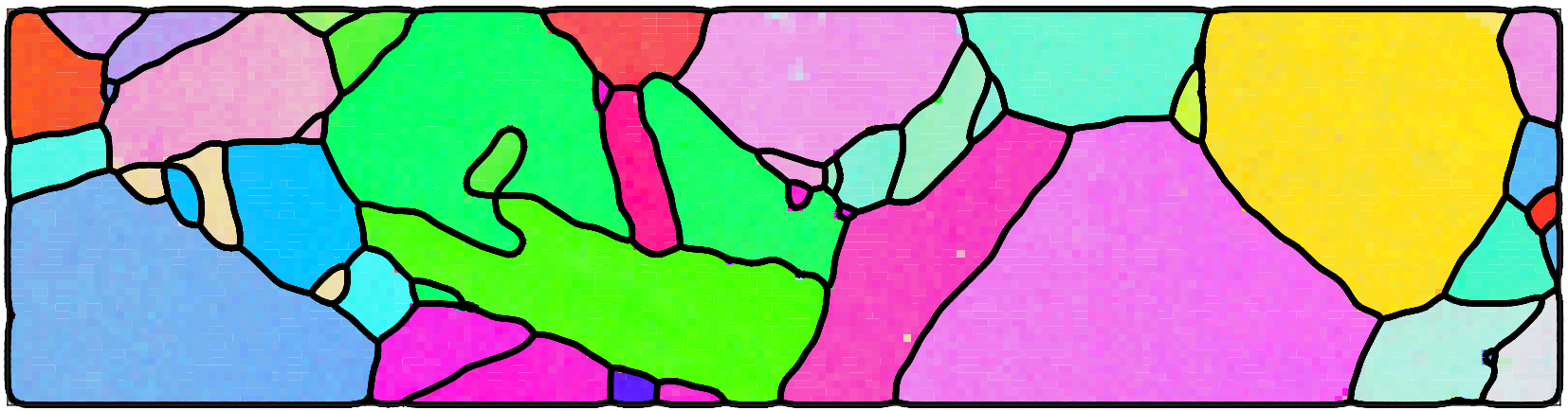


Stress Based Analysis - Route 2

```
sigma = stressTensor.uniaxial(vector3d.X)
```

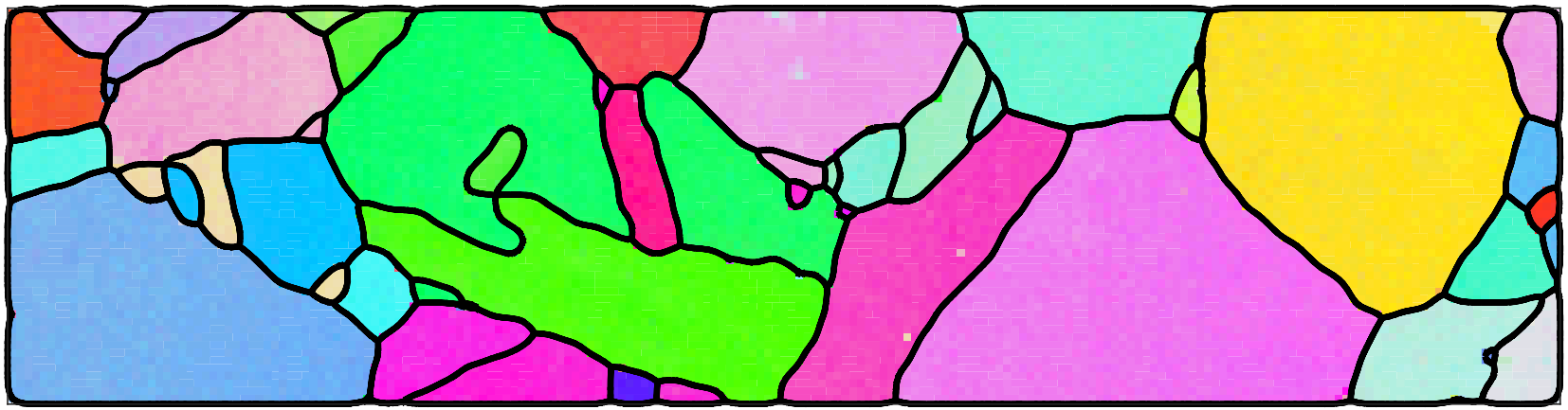
```
sigmaCrystal = inv(grains.meanOrientation) * sigma
```

```
sigmaCrystal = stressTensor (show methods, plot)
  size    : 71 x 1
  rank    : 2 (3 x 3)
  mineral: iron (m-3m)
```



Stress Based Analysis - Route 2

```
sigma = stressTensor.uniaxial(vector3d.X)
sigmaCrystal = inv(grains.meanOrientation) * sigma
SF = sS.SchmidFactor(sigmaCrystal);
```



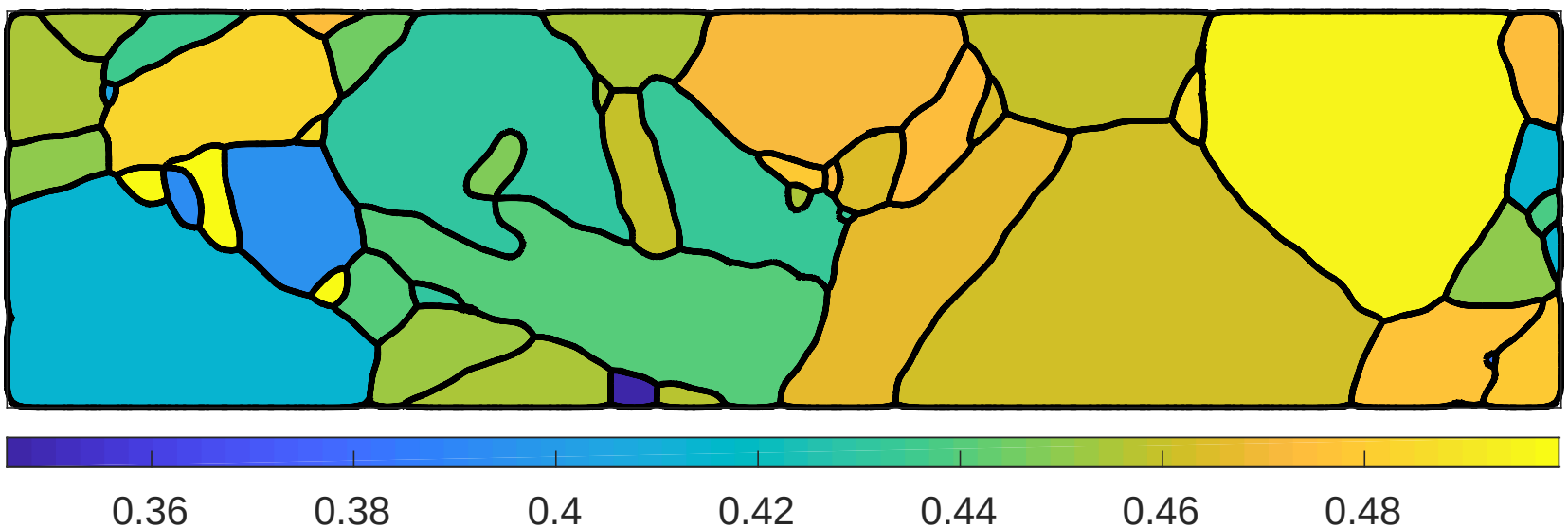
Stress Based Analysis - Route 2

```
sigma = stressTensor.uniaxial(vector3d.X)
```

```
sigmaCrystal = inv(grains.meanOrientation) * sigma
```

```
SF = sS.SchmidFactor(sigmaCrystal);
```

```
[maxSF,active] = max(abs(SF),[],2);
plot(grains,maxSF)
```



Stress Based Analysis - Route 2

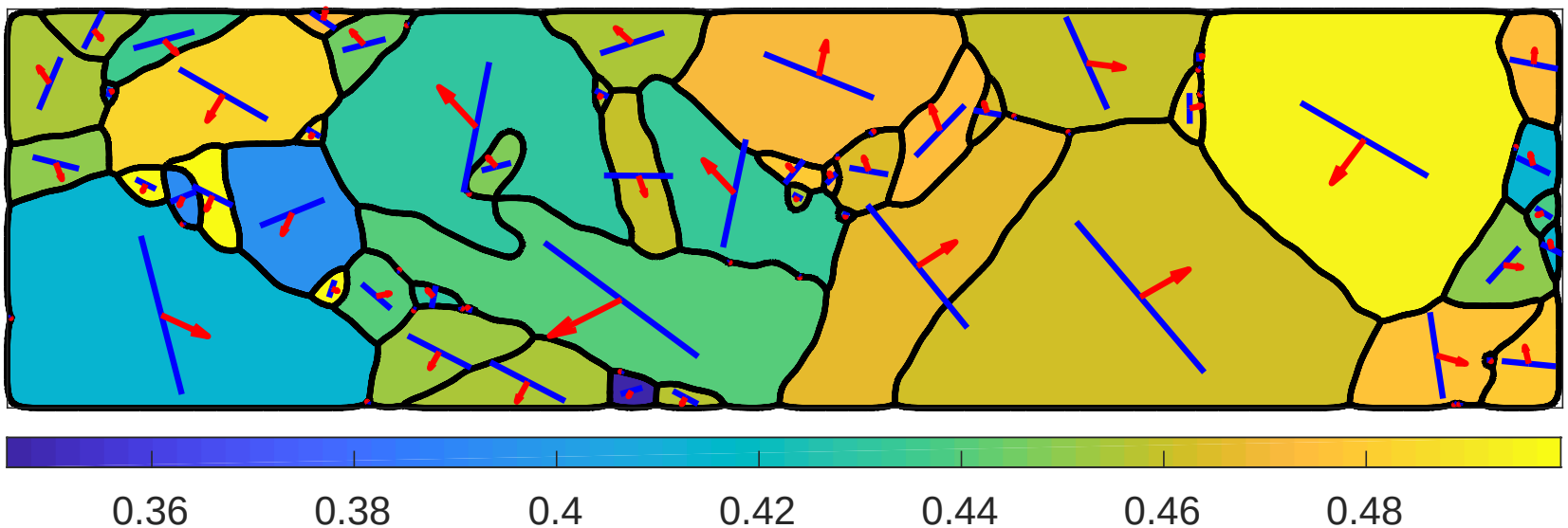
```
sigma = stressTensor.uniaxial(vector3d.X)
```

```
sigmaCrystal = inv(grains.meanOrientation) * sigma
```

```
SF = sS.SchmidFactor(sigmaCrystal);
```

```
[maxSF,active] = max(abs(SF),[],2);
plot(grains,maxSF)
```

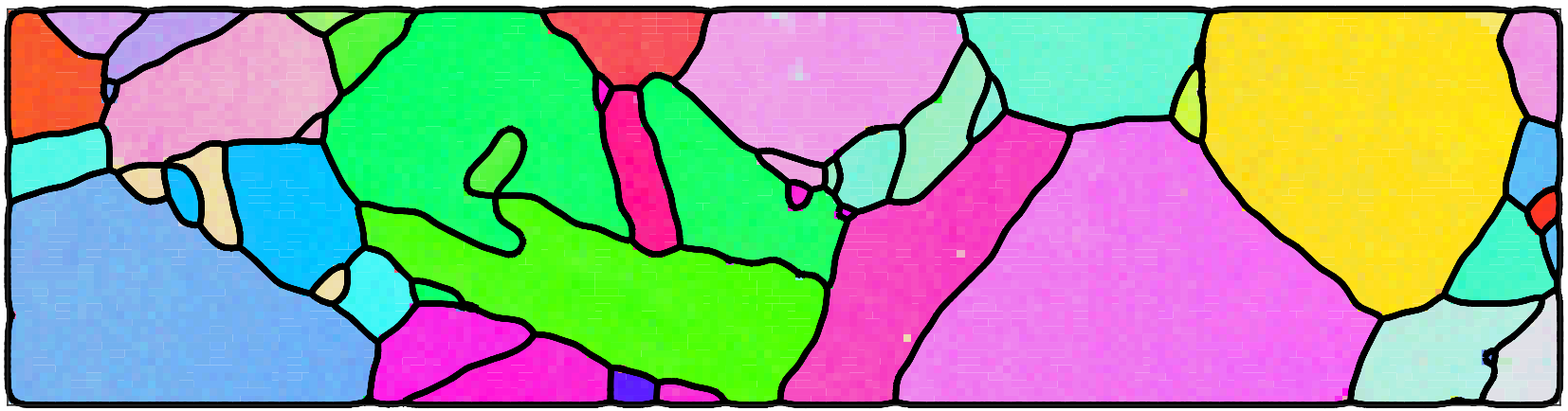
```
sSactive = grains.meanOrientation .* sS(active);
quiver(grains,sSactive.trace,'color','b')
quiver(grains,sSactive.b,'color','r')
```



Strain Based Analysis

```
eps = strainTensor(diag([1,0,-1]))
```

```
sigma = strainTensor (show methods, plot)
  rank: 2 (3 x 3)

 1  0  0
 0  0  0
 0  0 -1
```

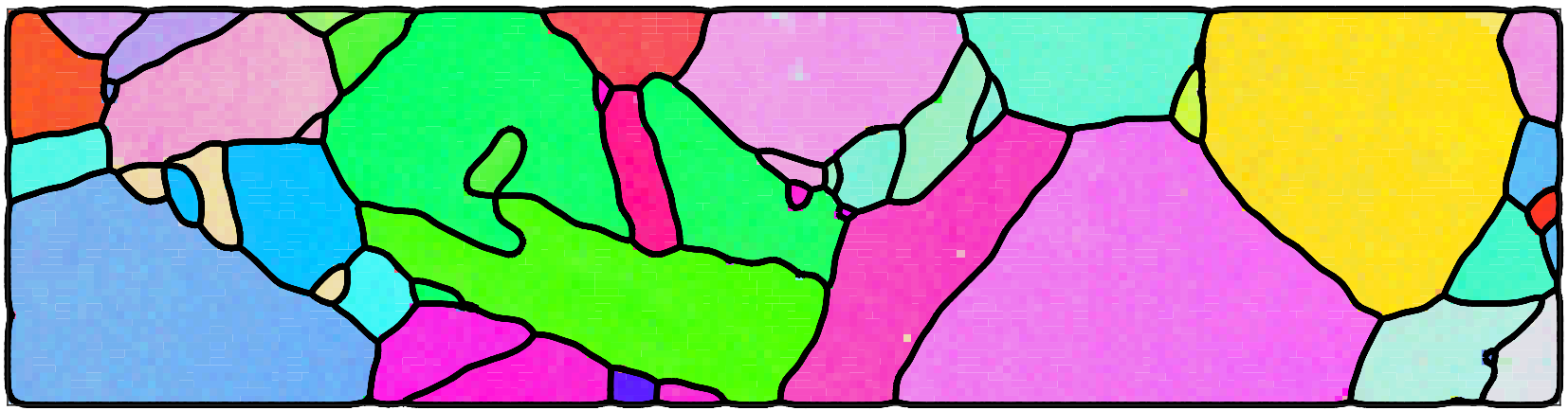


Strain Based Analysis

```
eps = strainTensor(diag([1,0,-1]))
```

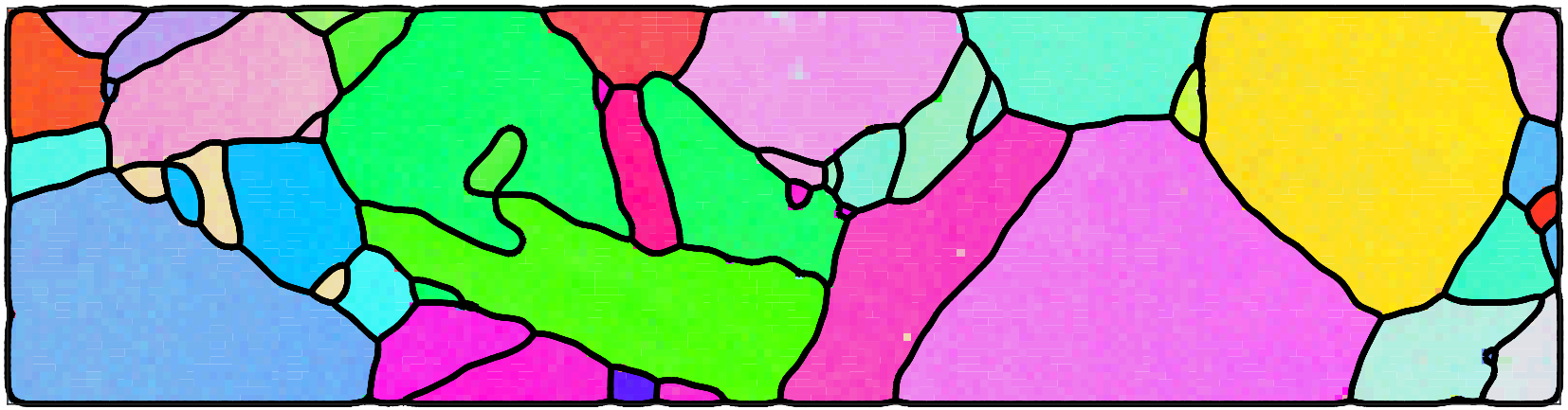
```
epsCrystal = inv(grains.meanOrientation) * eps
```

```
sigmaCrystal = strainTensor (show methods, plot)
  size    : 71 x 1
  rank    : 2 (3 x 3)
  mineral: iron (m-3m)
```



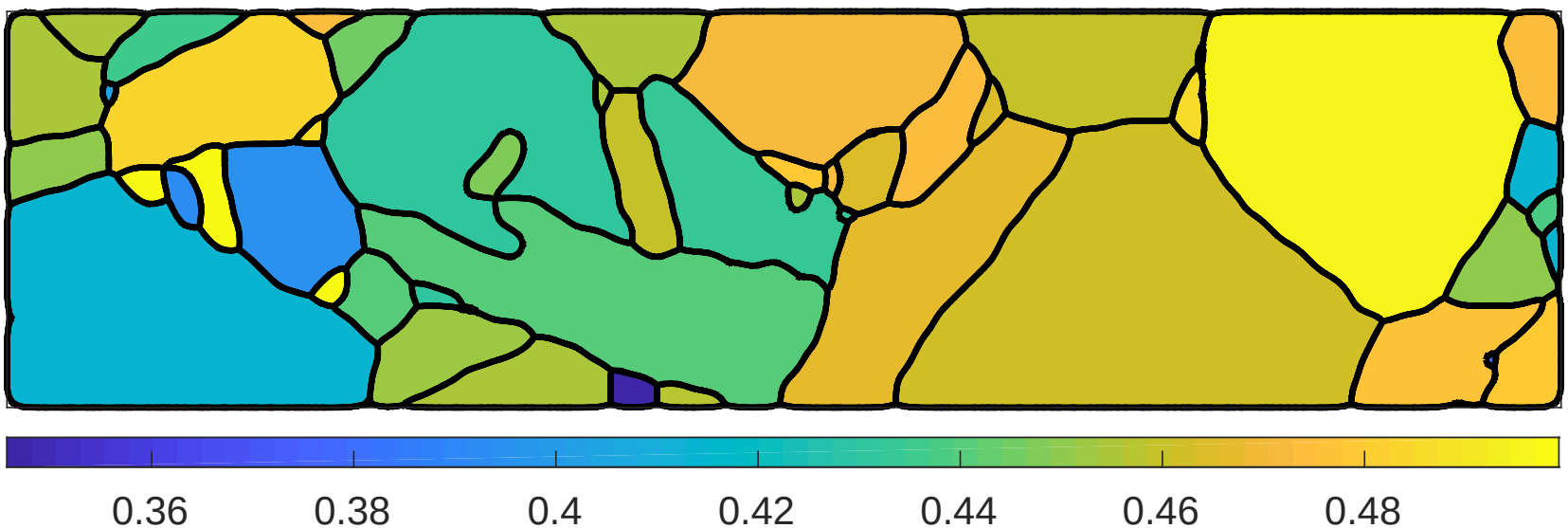
Strain Based Analysis

```
eps = strainTensor(diag([1,0,-1]))
epsCrystal = inv(grains.meanOrientation) * eps
[TF, b, mori] = calcTaylor(epsCrystal, sS);
```



Strain Based Analysis

```
eps = strainTensor(diag([1,0,-1]))
epsCrystal = inv(grains.meanOrientation) * eps
[TF, b, mori] = calcTaylor(epsCrystal, sS);
plot(grains,TF)
```



Strain Based Analysis

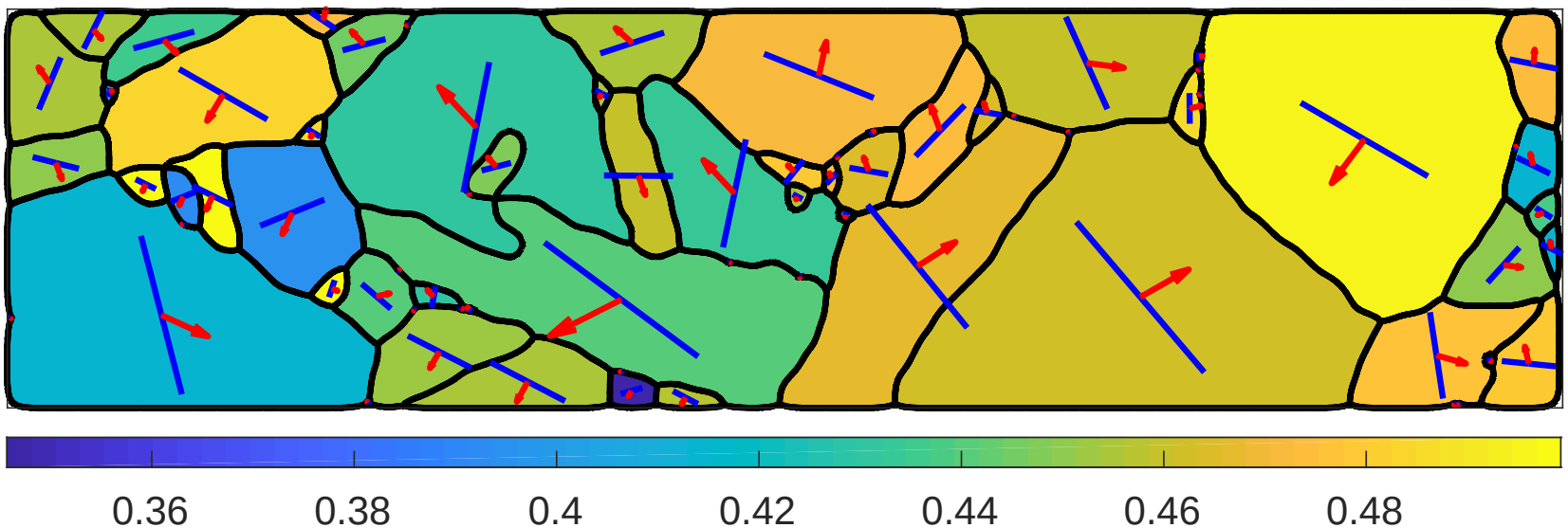
```
eps = strainTensor(diag([1,0,-1]))
```

```
epsCrystal = inv(grains.meanOrientation) * eps
```

```
[TF, b, mori] = calcTaylor(epsCrystal, sS);
```

```
plot(grains,TF)
```

```
[bMax ,bMaxId] = max(b,[],2);
sSactive = grains.meanOrientation .* sS(bMaxId);
quiver(grains,sSactive.b,'color','red')
quiver(grains,sSactive.trace,'color','blue')
```



Rolling texture evolution

```
ori = orientation.rand(10000,CS)
h = Miller({0,0,1},{1,1,1}),CS)
plotPDF(ori,'contourf')
```

the slip systems

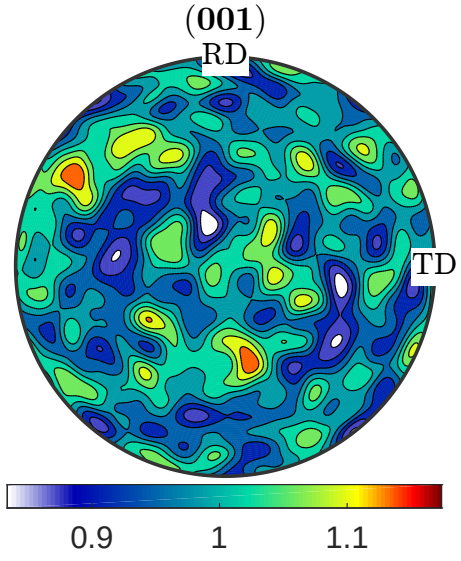
```
sS=symmetrise(slipSystem.fcc(CS))
```

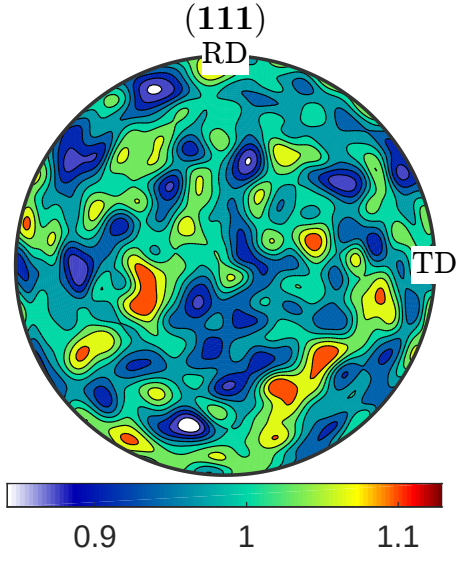
30 percent strain

```
eps = strainTensor(diag([0.3 0 -0.3]))
```

iterate deformation:

```
for i = 1:10
  [TF,b,mori] = calcTaylor(...
    inv(ori) * eps ./ 10, sS);
  ori = ori .* mori;
  plotPDF(ori,h,'contourf')
end
```





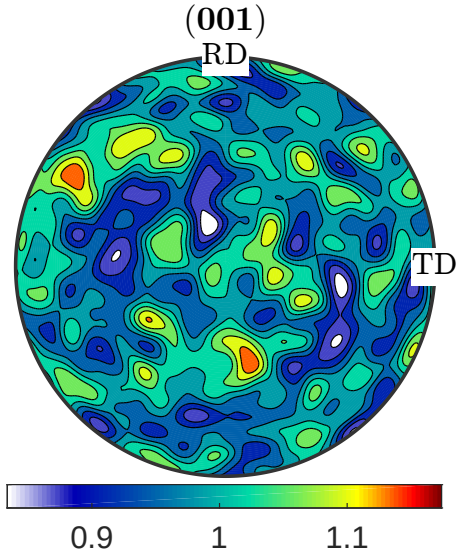
Rolling texture evolution

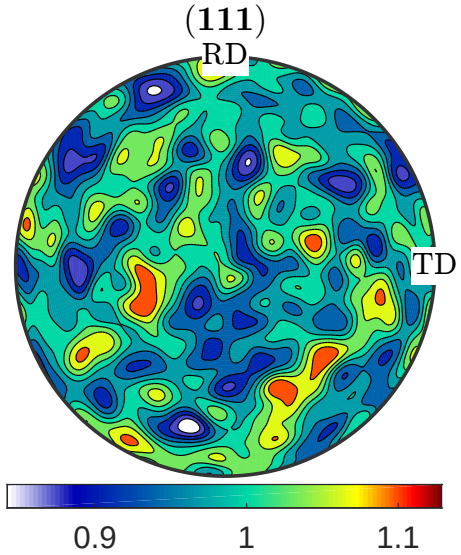
```
ori = orientation.rand(10000,CS)
h = Miller({0,0,1},{1,1,1}),CS)
plotPDF(ori,'contourf')
```

the slip systems

```
sS=symmetrise(slipSystem.fcc(CS))
```

```
sS = slipSystem (show methods, plot)
 size: 12 x 1
 mineral: Austenite (fcc) (432)
   u    v    w | h    k    l
   0    1   -1    1    1    1
  -1    0    1    1    1    1
   1   -1    0    1    1    1
   1   -1    0    1    1   -1
   1    0    1    1    1   -1
   0    1    1    1    1   -1
   0    1   -1   -1    1    1
   1    0    1   -1    1    1
   1    1    0   -1    1    1
  -1    0    1    1   -1    1
   1    1    0    1   -1    1
```





Rolling texture evolution

```
ori = orientation.rand(10000,CS)
h = Miller({0,0,1},{1,1,1}),CS)
plotPDF(ori,'contourf')
```

the slip systems

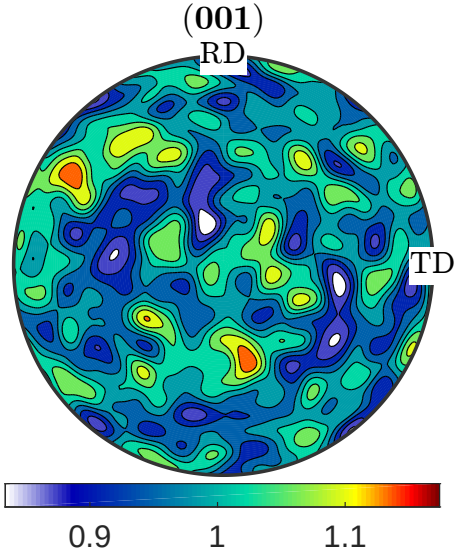
```
sS=symmetrise(slipSystem.fcc(CS))
```

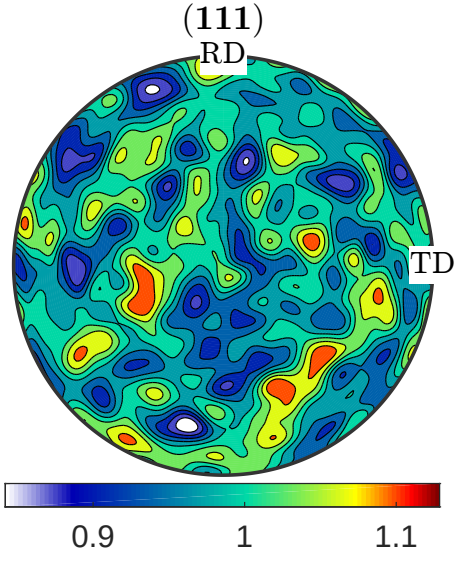
30 percent strain

```
eps = strainTensor(diag([0.3 0 -0.3]))
```

iterate deformation:

```
for i = 1:10
  [TF,b,mori] = calcTaylor(...
    inv(ori) * eps ./ 10, sS);
  ori = ori .* mori;
  plotPDF(ori,h,'contourf')
end
```





Rolling texture evolution

```
ori = orientation.rand(10000,CS)
h = Miller({0,0,1},{1,1,1}),CS)
plotPDF(ori,'contourf')
```

the slip systems

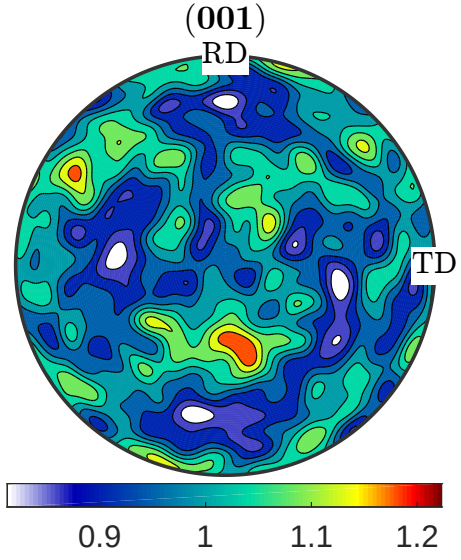
```
sS=symmetrise(slipSystem.fcc(CS))
```

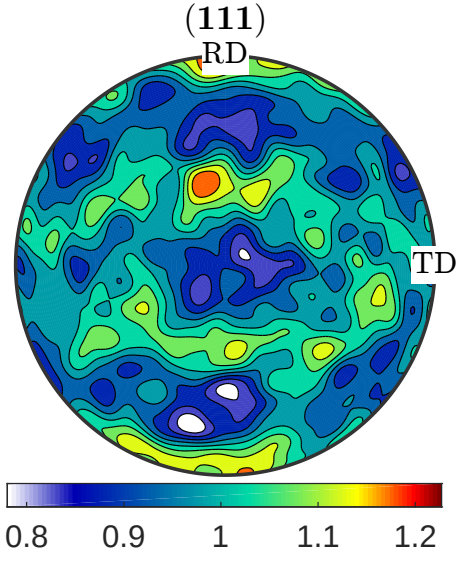
30 percent strain

```
eps = strainTensor(diag([0.3 0 -0.3]))
```

iterate deformation:

```
for i = 1:10
  [TF,b,mori] = calcTaylor(...
    inv(ori) * eps ./ 10, sS);
  ori = ori .* mori;
  plotPDF(ori,h,'contourf')
end
```





Rolling texture evolution

```
ori = orientation.rand(10000,CS)
h = Miller({0,0,1},{1,1,1}),CS)
plotPDF(ori,'contourf')
```

the slip systems

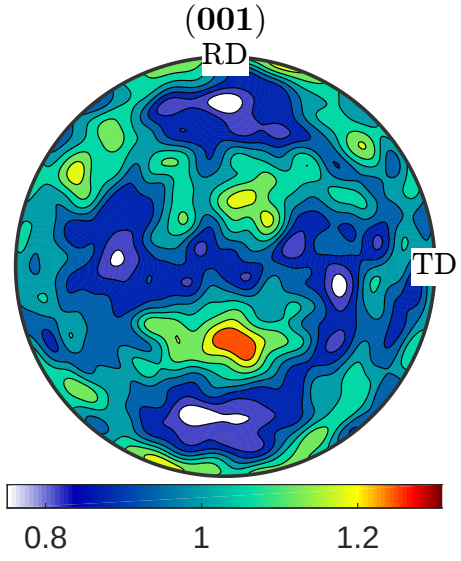
```
sS=symmetrise(slipSystem.fcc(CS))
```

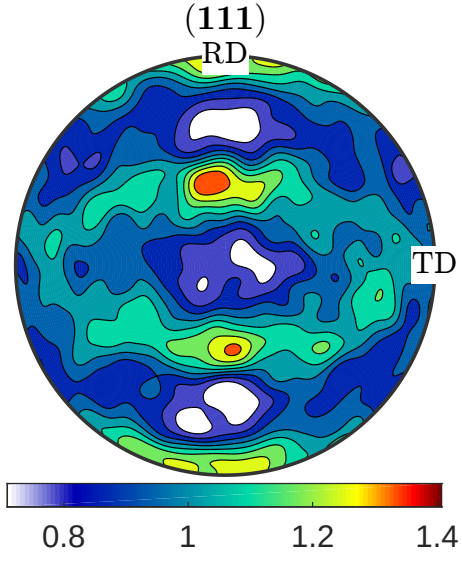
30 percent strain

```
eps = strainTensor(diag([0.3 0 -0.3]))
```

iterate deformation:

```
for i = 1:10
  [TF,b,mori] = calcTaylor(...
    inv(ori) * eps ./ 10, sS);
  ori = ori .* mori;
  plotPDF(ori,h,'contourf')
end
```





Rolling texture evolution

```
ori = orientation.rand(10000,CS)
h = Miller({0,0,1},{1,1,1}),CS)
plotPDF(ori,'contourf')
```

the slip systems

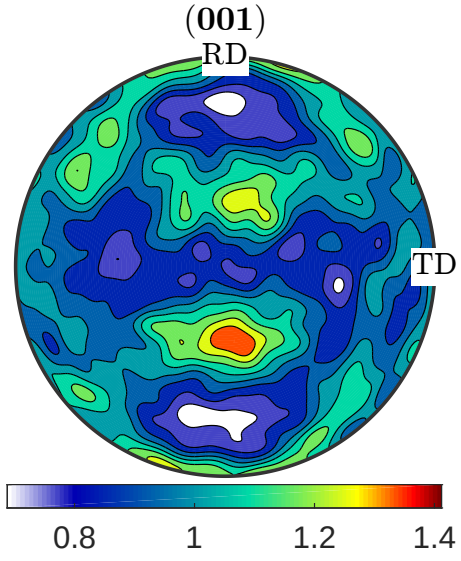
```
sS=symmetrise(slipSystem.fcc(CS))
```

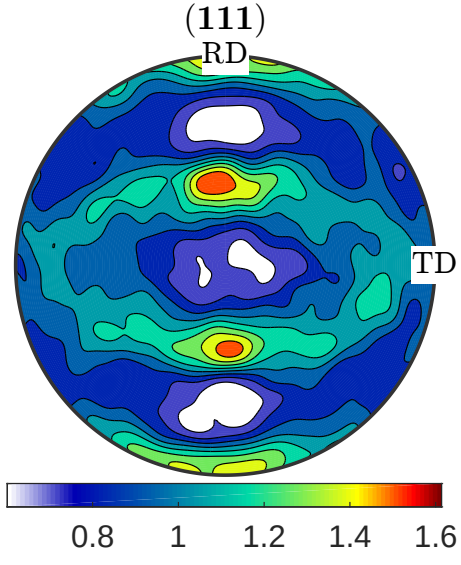
30 percent strain

```
eps = strainTensor(diag([0.3 0 -0.3]))
```

iterate deformation:

```
for i = 1:10
  [TF,b,mori] = calcTaylor(...
    inv(ori) * eps ./ 10, sS);
  ori = ori .* mori;
  plotPDF(ori,h,'contourf')
end
```





Rolling texture evolution

```
ori = orientation.rand(10000,CS)
h = Miller({0,0,1},{1,1,1}),CS)
plotPDF(ori,'contourf')
```

the slip systems

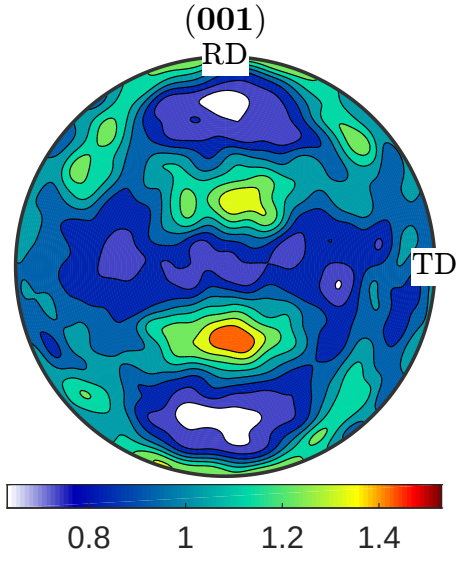
```
sS=symmetrise(slipSystem.fcc(CS))
```

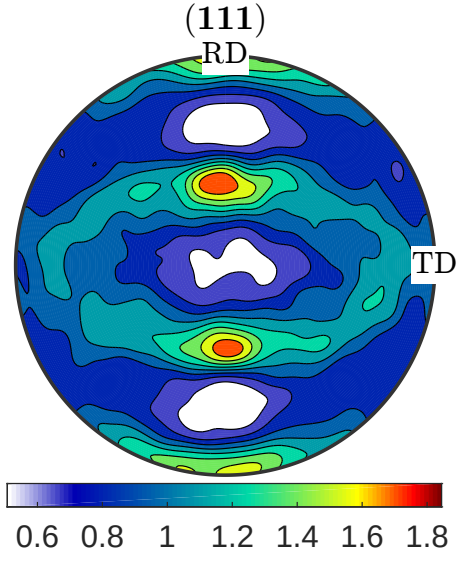
30 percent strain

```
eps = strainTensor(diag([0.3 0 -0.3]))
```

iterate deformation:

```
for i = 1:10
  [TF,b,mori] = calcTaylor(...
    inv(ori) * eps ./ 10, sS);
  ori = ori .* mori;
  plotPDF(ori,h,'contourf')
end
```





Rolling texture evolution

```
ori = orientation.rand(10000,CS)
h = Miller({0,0,1},{1,1,1}),CS)
plotPDF(ori,'contourf')
```

the slip systems

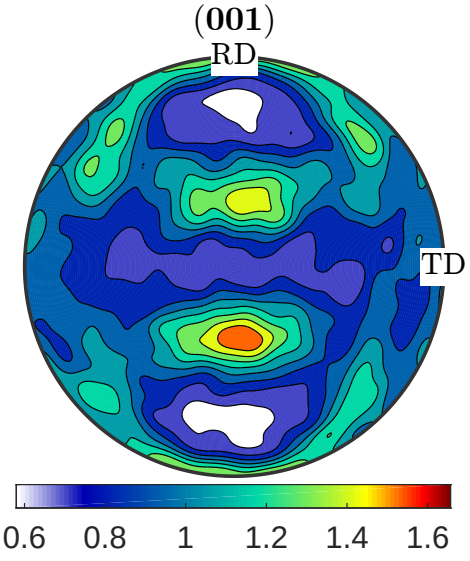
```
sS=symmetrise(slipSystem.fcc(CS))
```

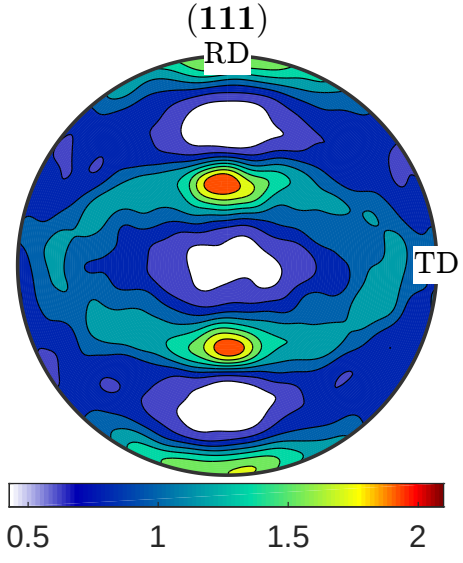
30 percent strain

```
eps = strainTensor(diag([0.3 0 -0.3]))
```

iterate deformation:

```
for i = 1:10
  [TF,b,mori] = calcTaylor(...
    inv(ori) * eps ./ 10, sS);
  ori = ori .* mori;
  plotPDF(ori,h,'contourf')
end
```





Rolling texture evolution

```
ori = orientation.rand(10000,CS)
h = Miller({0,0,1},{1,1,1}),CS)
plotPDF(ori,'contourf')
```

the slip systems

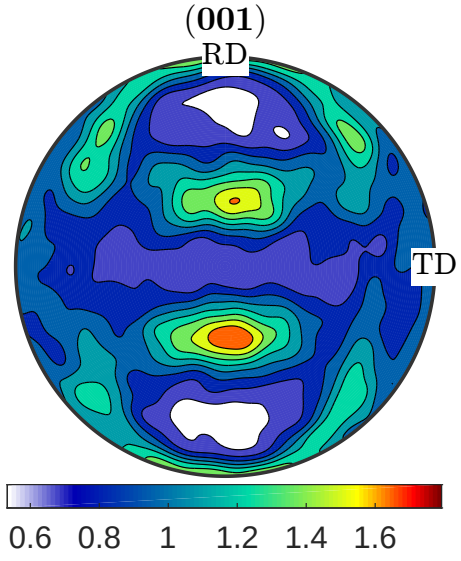
```
sS=symmetrise(slipSystem.fcc(CS))
```

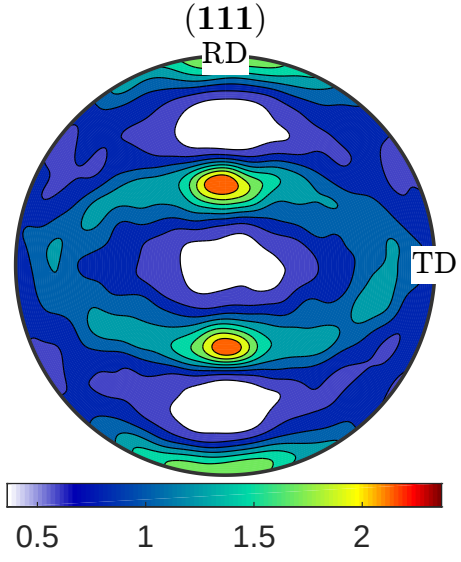
30 percent strain

```
eps = strainTensor(diag([0.3 0 -0.3]))
```

iterate deformation:

```
for i = 1:10
  [TF,b,mori] = calcTaylor(...
    inv(ori) * eps ./ 10, sS);
  ori = ori .* mori;
  plotPDF(ori,h,'contourf')
end
```





Rolling texture evolution

```
ori = orientation.rand(10000,CS)
h = Miller({0,0,1},{1,1,1}),CS)
plotPDF(ori,'contourf')
```

the slip systems

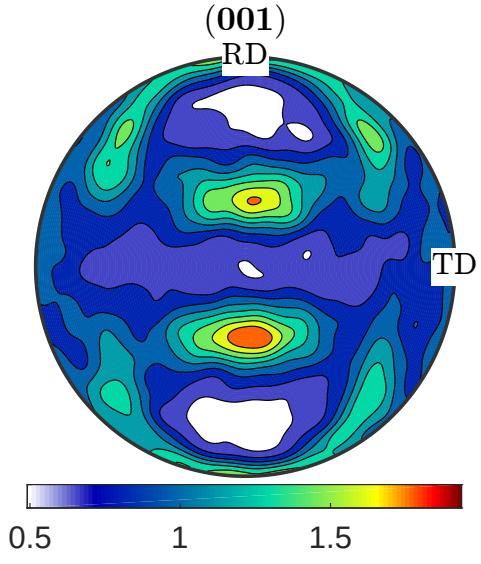
```
sS=symmetrise(slipSystem.fcc(CS))
```

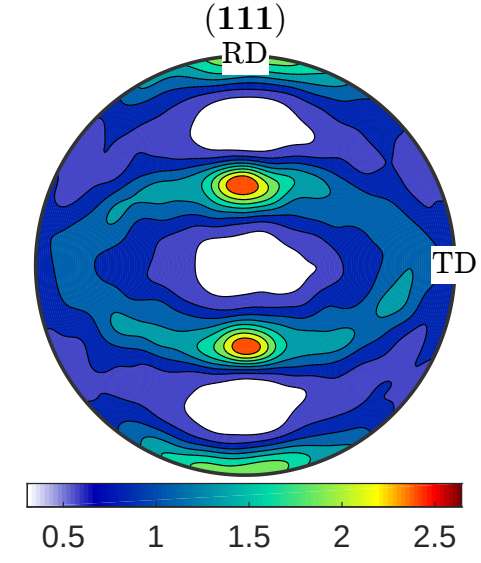
30 percent strain

```
eps = strainTensor(diag([0.3 0 -0.3]))
```

iterate deformation:

```
for i = 1:10
  [TF,b,mori] = calcTaylor(...
    inv(ori) * eps ./ 10, sS);
  ori = ori .* mori;
  plotPDF(ori,h,'contourf')
end
```





Rolling texture evolution

```
ori = orientation.rand(10000,CS)
h = Miller({0,0,1},{1,1,1}),CS)
plotPDF(ori,'contourf')
```

the slip systems

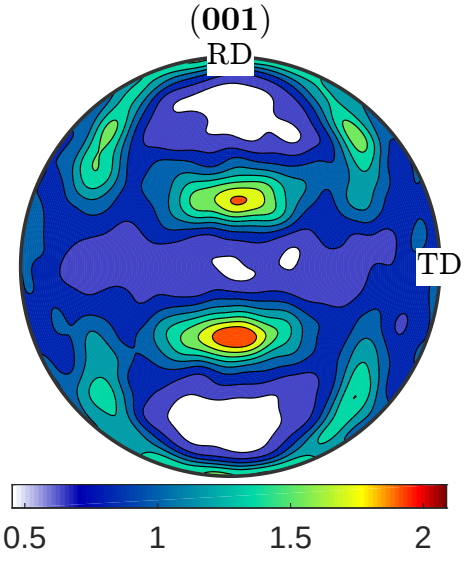
```
sS=symmetrise(slipSystem.fcc(CS))
```

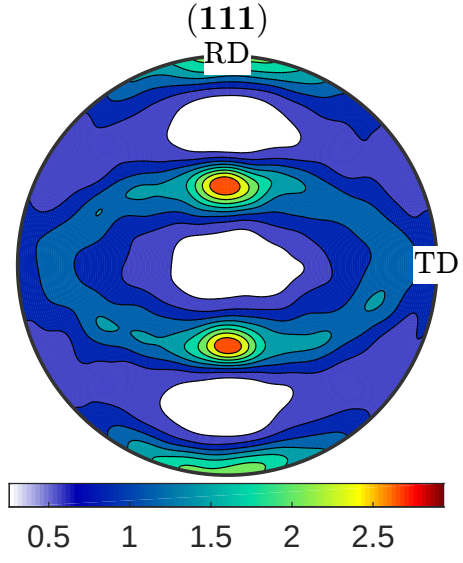
30 percent strain

```
eps = strainTensor(diag([0.3 0 -0.3]))
```

iterate deformation:

```
for i = 1:10
  [TF,b,mori] = calcTaylor(...
    inv(ori) * eps ./ 10, sS);
  ori = ori .* mori;
  plotPDF(ori,h,'contourf')
end
```





Rolling texture evolution

```
ori = orientation.rand(10000,CS)
h = Miller({0,0,1},{1,1,1}),CS)
plotPDF(ori,'contourf')
```

the slip systems

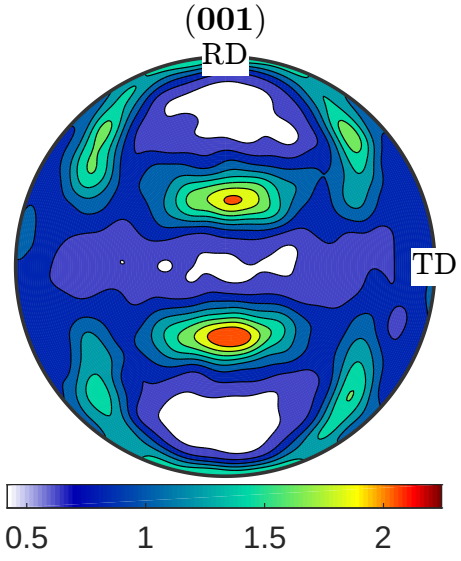
```
sS=symmetrise(slipSystem.fcc(CS))
```

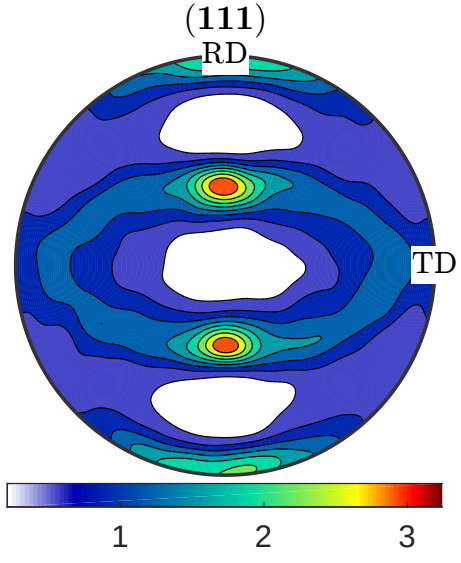
30 percent strain

```
eps = strainTensor(diag([0.3 0 -0.3]))
```

iterate deformation:

```
for i = 1:10
  [TF,b,mori] = calcTaylor(...
    inv(ori) * eps ./ 10, sS);
  ori = ori .* mori;
  plotPDF(ori,h,'contourf')
end
```





Rolling texture evolution

```
ori = orientation.rand(10000,CS)
h = Miller({0,0,1},{1,1,1}),CS)
plotPDF(ori,'contourf')
```

the slip systems

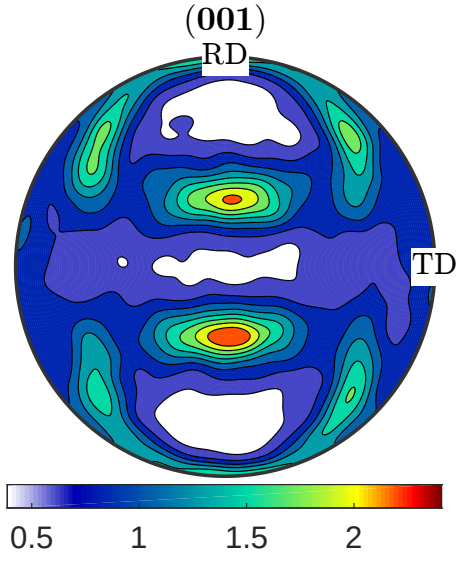
```
sS=symmetrise(slipSystem.fcc(CS))
```

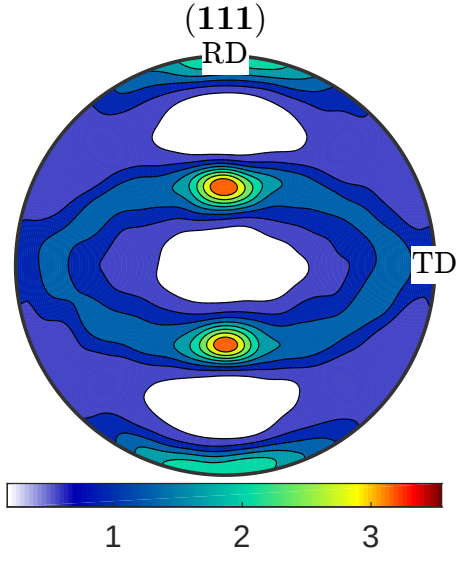
30 percent strain

```
eps = strainTensor(diag([0.3 0 -0.3]))
```

iterate deformation:

```
for i = 1:10
  [TF,b,mori] = calcTaylor(...
    inv(ori) * eps ./ 10, sS);
  ori = ori .* mori;
  plotPDF(ori,h,'contourf')
end
```





Misc

```
epsGrain
```

```
epsGrain = strainTensor (show methods, plot)
  size    : 37 x 1
  rank    : 2 (3 x 3)
  mineral: fcc (432)
```

```
epsGrain(1)
```

```
epsGrain{1,1}
```

Misc

```
epsGrain
```

```
epsGrain(1)
```

```
ans = strainTensor (show methods, plot)
  rank   : 2 (3 x 3)
  mineral: fcc (432)

 *10^-2
  57.35   -24.9 -70.07
  -24.9 -20.11   7.41
 -70.07   7.41 -37.24
```

```
epsGrain{1,1}
```

Misc

```
epsGrain
```

```
epsGrain(1)
```

```
epsGrain{1,1}
```

```
ans =
    0.5735
    0.7898
   -0.2335
    0.5767
    0.1689
    0.1850
    0.7818
   -0.1903
   -0.2332
    0.6518
    0.1988
    0.2005
    0.1999
   -0.2329
    0.6348
```