

---

# hexrd Documentation

*Release 0.2.3.dev*

**Joel Bernier, et. al.**

March 10, 2015



<b>1</b>	<b>Welcome to HEXRD’s documentation!</b>	<b>1</b>
1.1	The HEXRD API . . . . .	1
	<b>Python Module Index</b>	<b>67</b>



---

## Welcome to HEXRD's documentation!

---

Contents:

### 1.1 The HEXRD API

---

<code>hexrd.arrayutil</code>	
<code>hexrd.fileutil</code>	
<code>hexrd.gridutil</code>	
<code>hexrd.matrixutil</code>	
<code>hexrd.orientations</code>	
<code>hexrd.pfigutil</code>	
<code>hexrd.plotwrap</code>	***
<code>hexrd.quadrature.q1db</code>	
<code>hexrd.quadrature.q2db</code>	
<code>hexrd.quadrature.q3db</code>	
<code>hexrd.tens</code>	
<code>hexrd.valunits</code>	Module for associating units with scalar quantities
<code>hexrd.xrd.crystallography</code>	
<code>hexrd.xrd.detector</code>	
<code>hexrd.xrd.distortion</code>	
<code>hexrd.xrd.experiment</code>	Module for wrapping the main functionality of the xrd package.
<code>hexrd.xrd.fitting</code>	
<code>hexrd.xrd.grain</code>	
<code>hexrd.xrd.hydra</code>	Hydra detector tools
<code>hexrd.xrd.indexer</code>	
<code>hexrd.xrd.material</code>	Module for XRD material class
<code>hexrd.xrd.rotations</code>	
<code>hexrd.xrd.spacegroup</code>	Interface with sglite for hkl generation and Laue group determination
<code>hexrd.xrd.spotfinder</code>	
<code>hexrd.xrd.symmetry</code>	
<code>hexrd.xrd.transforms</code>	
<code>hexrd.xrd.transforms_CAPI</code>	
<code>hexrd.xrd.xrdbase</code>	
<code>hexrd.xrd.xrutil</code>	

---

### 1.1.1 Module: `arrayutil`

#### 6 Functions

`hexrd.arrayutil.getMem(shape, asOnes=False, asZeros=False, typeInt=False)`

`hexrd.arrayutil.toArray(a)`

`hexrd.arrayutil.writeArray(fid, *args, **dargs)`

Print to file, pasting arrays together; eventually replace with `numpy.savetxt?`

`hexrd.arrayutil.arrayToString(a)`

`hexrd.arrayutil.structuredSort(order, things)`

sort things by order, return sorted things

`hexrd.arrayutil.histoFit(data, nBins, plot=False)`

Fit data using histogramming; useful for stuff pulled in using `DataThief`

### 1.1.2 Module: `fileutil`

#### 5 Classes

`class hexrd.fileutil.Log(logFileName=None, toScreen=True)`

for logging

`__init__(logFileName=None, toScreen=True)`

`class hexrd.fileutil.FileDescr(*args)`

base class

`__init__(*args)`

`class hexrd.fileutil.FileLink(fileName)`

Bases: `hexrd.fileutil.FileDescr`

`__init__(fileName)`

`class hexrd.fileutil.FileLinkWild(fileNameWild)`

Bases: `hexrd.fileutil.FileDescr`

must be given an absolute path

`__init__(fileNameWild)`

`class hexrd.fileutil.FileForm(fileName, formFile=None, form=None, dictForDefs=None)`

Bases: `hexrd.fileutil.FileDescr`

`__init__(fileName, formFile=None, form=None, dictForDefs=None)`

#### 22 Functions

`hexrd.fileutil.resolveWild(fname)`

might be better to go to `glob.glob()` call here

`hexrd.fileutil.catList(lines, sep='')`

`hexrd.fileutil.fileToForm(fname)`

```
hexrd.fileutil.readFloatDataA (fname)
```

This is definitely faster than `readFloatData` when it is appropriate, which is pretty much any time the data can be interpreted as an array and numpy is available

```
hexrd.fileutil.readFloatData (fname=None)
```

```
hexrd.fileutil.readDataFlat (fname)
```

```
hexrd.fileutil.archiveDir (dirName)
```

```
hexrd.fileutil.archiveFile (fileName)
```

```
hexrd.fileutil.getFromPipe (command, werr=False)
```

```
hexrd.fileutil.getSysType ()
```

```
hexrd.fileutil.getNCorePerNode ()
```

```
hexrd.fileutil.getHostName ()
```

```
hexrd.fileutil.getBankNames ()
```

```
hexrd.fileutil.rmSafe (fileName)
```

```
hexrd.fileutil.rmDirF (dirName)
```

```
hexrd.fileutil.rmWorkDir (workdir)
```

```
hexrd.fileutil.listFiles (filename)  
most useful if filename contains a wildcard
```

```
hexrd.fileutil.rmWild (filename)
```

```
hexrd.fileutil.getScratchBaseDir ()
```

```
hexrd.fileutil.argListToStr (argv)
```

```
hexrd.fileutil.dictToDefs (d)
```

```
hexrd.fileutil.readFLT (fname, structured=False)  
Read a Fable-style .flt file
```

JVB 2011/03/24

### 1.1.3 Module: `gridutil`

#### 7 Functions

```
hexrd.gridutil.cellIndices (edges, points_1d)  
get indices in a 1-d regular grid.
```

edges are just that:

**point: x (2.5)**

**edges: |1 |2 |3 |4 |5**

**indices: |0 |1 |2 |3 |**

above the deltas are + and the index for the point is 1

**point: x (2.5)**

**edges:** |5 |4 |3 |2 |1|

**indices:** |0 |1 |2 |3 |

here the deltas are - and the index for the point is 2

- can handle grids with +/- deltas
- be careful when using with a cyclical angular array! edges and points must be mapped to the same branch cut, and  $\text{abs}(\text{edges}[0] - \text{edges}[-1]) = 2\pi$

`hexrd.gridutil.cellConnectivity(m, n, p=1, origin='ul')`  
p x m x n (layers x rows x cols)

origin can be upper left – ‘ul’ <default> or lower left – ‘ll’

choice will affect handedness (cw or ccw)

`hexrd.gridutil.cellCentroids(crd, con)`  
`con.shape = (nele, 4)` `crd.shape = (ncrd, 2)`  
`con.shape = (nele, 8)` `crd.shape = (ncrd, 3)`

`hexrd.gridutil.computeArea(polygon)`  
must be ORDERED and CONVEX!

`hexrd.gridutil.computeIntersection(line1, line2)`  
compute intersection of two-dimensional line intersection  
this is an implementation of two lines:

`line1 = [ [x0, y0], [x1, y1] ]` `line2 = [ [x3, y3], [x4, y4] ]`  
<[http://en.wikipedia.org/wiki/Line-line\\_intersection](http://en.wikipedia.org/wiki/Line-line_intersection)>

`hexrd.gridutil.isinside(point, boundary, ccw=True)`  
Assumes CCW boundary ordering

`hexrd.gridutil.sutherlandHodgman(subjectPolygon, clipPolygon)`

## 1.1.4 Module: `matrixutil`

### 22 Functions

`hexrd.matrixutil.columnNorm(a)`  
normalize array of column vectors (hstacked, axis = 0)

`hexrd.matrixutil.rowNorm(a)`  
normalize array of row vectors (vstacked, axis = 1)

`hexrd.matrixutil.unitVector(a)`  
normalize array of column vectors (hstacked, axis = 0)

`hexrd.matrixutil.nullSpace(A, tol=1e-14)`  
computes the null space of the real matrix A

`hexrd.matrixutil.blockSparseOfMatArray(matArray)`  
Constructs a block diagonal sparse matrix (csc format) from a (p, m, n) ndarray of p (m, n) arrays  
...maybe optional args to pick format type?

`hexrd.matrixutil.symmToVecMV(A)`  
convert from symmetric matrix to Mandel-Voigt vector representation (JVB)



`hexrd.matrixutil.vecMVToSymm(A)`

convert from Mandel-Voigt vector to symmetric matrix representation (JVB)

`hexrd.matrixutil.vecMVCOBMatrix(R)`

GenerateS array of 6 x 6 basis transformation matrices for the Mandel-Voigt tensor representation in 3-D given by:

$[A] = [[A_{11}, A_{12}, A_{13}], [A_{12}, A_{22}, A_{23}], [A_{13}, A_{23}, A_{33}]]$

$\{A\} = [A_{11}, A_{22}, A_{33}, \sqrt{2}A_{23}, \sqrt{2}A_{13}, \sqrt{2}A_{12}]$

where the operation  $R * A * R.T$  (in tensor notation) is obtained by the matrix-vector product  $[T]*\{A\}$ .

USAGE

$T = \text{vecMVCOBMatrix}(R)$

INPUTS

1.R is (3, 3) an ndarray representing a change of basis matrix

OUTPUTS

1.T is (6, 6), an ndarray of transformation matrices as described above

NOTES

1.Components of symmetric 4th-rank tensors transform in a manner analogous to symmetric 2nd-rank tensors in full matrix notation.

SEE ALSO

`symmToVecMV`, `vecMVToSymm`, `quatToMat`

`hexrd.matrixutil.nrmlProjOfVecMV(vec)`

Gives vstacked  $p \times 6$  array to perform  $n' * A * n$  as  $[N]*\{A\}$  for p hstacked input 3-vectors using the Mandel-Voigt convention.

$Nvec = \text{normalProjectionOfMV}(vec)$

\*) the input vector array need not be normalized; it is performed in place

`hexrd.matrixutil.rankOneMatrix(vec1, *args)`

Create rank one matrices (dyadics) from vectors.

$r1mat = \text{rankOneMatrix}(vec1)$   $r1mat = \text{rankOneMatrix}(vec1, vec2)$

$vec1$  is  $m1 \times n$ , an array of n hstacked  $m1$  vectors  $vec2$  is  $m2 \times n$ , (optional) another array of n hstacked  $m2$  vectors

**$r1mat$  is  $n \times m1 \times m2$ , an array of n rank one matrices** formed as  $c1*c2'$  from columns  $c1$  and  $c2$

With one argument, the second vector is taken to the same as the first.

Notes:

\*) **This routine loops on the dimension m, assuming this** is much smaller than the number of points, n.

`hexrd.matrixutil.skew(A)`

skew-symmetric decomposition of n square (m, m) ndarrays. Result is a (squeezed) (n, m, m) ndarray

`hexrd.matrixutil.symm(A)`

symmetric decomposition of n square (m, m) ndarrays. Result is a (squeezed) (n, m, m) ndarray.

`hexrd.matrixutil.skewMatrixOfVector(w)`

given a (3, n) ndarray, w, of n hstacked axial vectors, computes the associated skew matrices and stores them in an (n, 3, 3) ndarray. Result is (3, 3) for w.shape = (3, 1) or (3, ).

See also: `vectorOfSkewMatrix`

`hexrd.matrixutil.vectorOfSkewMatrix(W)`

given an (n, 3, 3) or (3, 3) ndarray, W, of n stacked 3x3 skew matrices, computes the associated axial vector(s) and stores them in an (3, n) ndarray. Result always has ndim = 2.

See also: `skewMatrixOfVector`

`hexrd.matrixutil.multMatArray(ma1, ma2)`

multiply two 3-d arrays of 2-d matrices

`hexrd.matrixutil.uniqueVectors(v, tol=1e-12)`

Sort vectors and discard duplicates.

USAGE:

`uvec = uniqueVectors(vec, tol=1.0e-12)`

v – tol – (optional) comparison tolerance

4. (a)Boyce 2010-03-18

`hexrd.matrixutil.findDuplicateVectors(vec, tol=1e-14, equivPM=False)`

Find vectors in an array that are equivalent to within a specified tolerance

USAGE:

`eqv = DuplicateVectors(vec, *tol)`

INPUT:

1. **vec is n x m, a double array of m horizontally concatenated** n-dimensional vectors.

**\*2) tol is 1 x 1, a scalar tolerance. If not specified, the default** tolerance is 1e-14.

**\*3)** set equivPM to True if vec and -vec are to be treated as equivalent

OUTPUT:

1. eqv is 1 x p, a list of p equivalence relationships.

NOTES:

Each equivalence relationship is a 1 x q vector of indices that represent the locations of duplicate columns/entries in the array vec. For example:

1 2 2 2 1 2 7 |

**vec =** | |

2 3 5 3 2 3 3 |

eqv = [[1x2 double] [1x3 double]], where

eqv[0] = [0 4] eqv[1] = [1 3 5]

`hexrd.matrixutil.normvec(v)`

`hexrd.matrixutil.normvec3(v)`

`hexrd.matrixutil.normalized(v)`

```
hexrd.matrixutil.cross (v1, v2)
hexrd.matrixutil.determinant3 (mat)
```

## 1.1.5 Module: orientations

### 8 Classes

```
class hexrd.orientations.RotationParameterization (args)
    template for rotation parameterization class

    __init__ (args)

    toMatrix ()
        use matrix as common representation all classes should have a constructor that works using this
        ** Construct [C] matrix (Kocks convention) ** {a} = [C] {a} ** sm cr

class hexrd.orientations.RotInv (*args)
    Bases: hexrd.orientations.RotationParameterization
    rotation invariants

    __init__ (*args)

class hexrd.orientations.CanovaEuler (*args)
    Bases: hexrd.orientations.RotationParameterization

    __init__ (*args)

class hexrd.orientations.KocksEuler (*args)
    Bases: hexrd.orientations.RotationParameterization

    Kocks Euler angles see equation 6 page 65 of koc-tom-wen-98a (Kocks, Tome, & Wenk; Texture and
    Anisotropy)

    __init__ (*args)

    toBunge ()
        trusting Table 1a of koc-tom-wen-98a (Kocks, Tome, & Wenk; Texture and Anisotropy)

class hexrd.orientations.BungeEuler (*args)
    Bases: hexrd.orientations.RotationParameterization

    Bunge Euler angles see koc-tom-wen-98a (Kocks, Tome, & Wenk; Texture and Anisotropy)

    __init__ (*args)

    toKocks ()
        trusting Table 1a of koc-tom-wen-98a (Kocks, Tome, & Wenk; Texture and Anisotropy)

class hexrd.orientations.Quat (*args)
    Bases: hexrd.orientations.RotationParameterization

    quaternions, normalized; parameterization of SO(3); do not bother making unique (is 2-to-1)

    __init__ (*args)

    T ()
        return transposed quaternion

    static getRandQuat (n=1)
        sample uniform orientation distribution to get quaternion parameters
```

**misorAng** (*other*)  
compute the misorientation angle, in radians, in  $[0, 2\pi]$

**normalize** ()  
normalize the quaternion

**static normalizeQuat** (*q*)  
normalize in place

**normalized** ()  
return normalize quaternion

**transpose** ()  
transpose the quaternion in place; retains metadata

**transposed** ()  
return transposed quaternion

**class** hexrd.orientations.**SymmGroup** (*\*args*)  
symmetry group

**\_\_init\_\_** (*\*args*)

**findDistinct** (*qList*)  
given a list of quaternions, find those which are distinct under the symmetry group

**class** hexrd.orientations.**Fiber** (*latVec*, *recipVec*)  
Like John Edmiston's MakeFiber class, but with the implementation more tightly coupled to the rest of the code base

**\_\_init\_\_** (*latVec*, *recipVec*)

**distBetweenFibers** (*other*)  
Compute the distance between two fibers using the polar decomposition of the projection operator taking one geodesic plane to the other. input: instance of MakeFiber class output: (max\_eigenvalue, Rotation at max\_eigenvalue), intersecting fibers would have max\_eigenvalue ~ 1. Rotation at max\_eigenvalue would be the 'closest' Rotation which would relate the two fibers.

## 25 Functions

hexrd.orientations.**arccosSafe** (*temp*)

hexrd.orientations.**orthogonalize** (*rMatIn*)

hexrd.orientations.**traceToAng** (*tr*)  
given trace of a rotation matrix find the angle or rotation

hexrd.orientations.**matToCanova** (*r*)

hexrd.orientations.**invToRodr** (*inv*)  
do not check for divide-by-zero

hexrd.orientations.**rodrToInv** (*rodr*)  
do not check for divide-by-zero

hexrd.orientations.**rodrToQuat** (*rodr*)

hexrd.orientations.**invToM** (*theta*, *n0*, *n1*, *n2*)

hexrd.orientations.**invToQuat** (*inv*)

hexrd.orientations.**bungeToMat** (*euler*)

`hexrd.orientations.matToQuat(r)`

based on Spurrier's algorithm for quaternion extraction, as described in cite{sim-vuq-85a}

returns a 4-vector, not a Quat instance

BibTeX: @TechReport{sim-vuq-85a, author = {J. C. Simo and L. {Vu Quoc}}, title = {Three dimensional finite strain rod model part

{II}: computational aspects, Memorandum No. {UCB/ERL M85/31}},

**institution** = {Electronics Research Laboratory, College of Engineering, University of California, Berkeley},

year = {1985} }

`hexrd.orientations.matToThetaN(r)`

2nd order tensor => angle/axis

to see that angle is right take R in a basis so that  $R_{11}=1$ ; get  $\text{tr}(R) = 1 + 2 \cos(\theta)$ , solve for  $\theta$  and use argument about invariance of  $\text{tr}(R)$

references: 1) box 4 in Simo and VuQuoc, 1985, ERL Berkeley memorandum no. UCB/ERL M85/31 2) Marin and Dawson 98 part 1, equation for update  $d_{rstar}$  (exponential mapping)

$n$  is in vector notation of a skew tensor according to  $w_i = 1/2 \epsilon_{ijk} W_{jk}$

`hexrd.orientations.quatToInv(q)`

`hexrd.orientations.quatToMat(quat)`

Take an array of  $n$  quats (numpy ndarray,  $4 \times n$ ) and generate an array of rotation matrices ( $n \times 3 \times 3$ )

Uses the truncated series expansion for the exponential map; divide-by-zero is checked using the global 'tiny-RotAng'

`hexrd.orientations.quatToProdMat(quat, mult='right')`

Form  $4 \times 4$  arrays to perform the quaternion product

**USAGE** `qmats = quatToProdMat(quats, mult='right')`

## INPUTS

1. quats is  $(4, n)$ , a numpy ndarray array of  $n$  quaternions horizontally concatenated
2. mult is a keyword arg, either 'left' or 'right', denoting the sense of the multiplication:

`/ quatToProdMat(h, 'right') * q`

`q * h -> quatToProdMat(q, 'left') * h`

## OUTPUTS

1. qmats is  $(n, 4, 4)$ , the left or right quaternion product operator

## NOTES

**\*) This function is intended to replace a cross-product based** routine for products of quaternions with large arrays of quaternions (e.g. applying symmetries to a large set of orientations).

`hexrd.orientations.sampleToLatticeT2(A_sm, C)`

T

$[A_{sm}] = [C][A_{lat}][C]$

`hexrd.orientations.latticeToSampleT2(A_lat, C)`

T

$[A_{sm}] = [C][A_{lat}][C]$

hexrd.orientations.**latticeToSampleV**(*V\_lat*, *C*)

hexrd.orientations.**rotMatrixFromCrystalVectors**(*cvs1=None*, *cvs2=None*, *cvs3=None*)

Make a rotation matrix in the RotationParameterization convention from components of crystal vectors that are along given sample directions

hexrd.orientations.**transposeQuats**(*qList*)

hexrd.orientations.**stripQuatList**(*qList*)

hexrd.orientations.**writeQuats**(*qList*, *f*)

hexrd.orientations.**makeQuatsBall**(*qRef*, *thetaScale*, *n*)

make *n* quaternions in a ball around *qRef*, with ball size scaled by *thetaScale*

hexrd.orientations.**makeQuatsComponents**(*nGrain*, *scale=None*)

hexrd.orientations.**millerBravais2Normal**(*invec*, *\*args*)

**Generate the normal(s) for a plane(s) given in** the Miller-Bravais convention for the hexagonal basis {*a1*, *a2*, *a3*, *c*}. The basis for the output {*o1*, *o2*, *o3*} is chosen such that:

$o1 \parallel a1 \quad o3 \parallel c \quad o2 = o3 \wedge o1$

returns a (3, *n*) array of horizontally concatenated unit vectors

## 1.1.6 Module: pfigutil

### 6 Functions

hexrd.pfigutil.**sph2n**(*coords\_sph*)

convert from chi/eta spherical coordinates to normal vectors; can use with coords from femODF.FemHemisphere

hexrd.pfigutil.**n2sph**(*nVectors*)

hexrd.pfigutil.**n2eap**(*nVectors*, *flip=True*)

unit vectors to equal-area projection

hexrd.pfigutil.**renderEAProj**(*nVecs*, *vals*, *n*, *patch=False*, *sum=False*, *nzByContrib=True*, *northernOnly=False*)

render an equal-area projects of general pole values, on an *n*-by-*n* grid; if *sum=True*, then sum contributions, otherwise average them; returns a masked array

hexrd.pfigutil.**fromSouthern**(*nVecs*, *invert*)

hexrd.pfigutil.**drawLines**(*pw*, *pointLists*=[], *netStyle=None*, *netNDiv=12*, *netAlpha=0.5*, *rMat=None*, *southern=False*, *invertFromSouthern=True*, *origin*=(0.0, 0.0), *r=1.0*)

## 1.1.7 Module: plotwrap

\*\*\* For now, plotwrap is hardwired for TkAgg this is not great, but also not a high priority to fix for now; PlotWinP might be the start of a decent fix

but plotwrap was built with the idea that you could have a plot without a window, and pyplot relies on the new\_figure\_manager functions in the backends, which always make a window; what we need is something to make the figure and the canvas without the figure manager ... but FigureCanvasMac

... how does one get a drawable figure that is not necessarily drawn?!

## 4 Classes

```
class hexrd.plotwrap.PlotWin (numRows=1, numCols=-1, title='PlotWin window', figure=None,
                             relfigsize=(3, 3), axesList=None, noAutoAxesList=False, dpi=100)

    __init__ (numRows=1, numCols=-1, title='PlotWin window', figure=None, relfigsize=(3, 3), axes-
              List=None, noAutoAxesList=False, dpi=100)
        If pass negative numCols, then numRows is the number of plots and the layout is done automatically

    getAxes (plotNum, rect=None, withPW=False, **axprops)
        careful: plotNum is from 0, not 1 as is the case for subplot axprops is useful for this like setting sharex and
        sharey

class hexrd.plotwrap.PlotWinLite (canvas, figure, axes)
    Lightweight PlotWin substitute for when windows are being controlled by code outside of plotwrap

    __init__ (canvas, figure, axes)

    haveXLabels ()
        may want to turn off any functionality in this method

    haveYLabels ()
        may want to turn off any functionality in this method

class hexrd.plotwrap.PlotWinP (axes=None, as3D=False, title=None, **kwargs)
    Just wrap pyplot

    __init__ (axes=None, as3D=False, title=None, **kwargs)

    haveXLabels ()
        may want to turn off any functionality in this method

    haveYLabels ()
        may want to turn off any functionality in this method

    pwList = None
        for now, punt on setting a title

class hexrd.plotwrap.PlotWrap (**keyArgs)
    Bases: object

    __init__ (**keyArgs)

    colorbar (rect=(0.8, 0.1, 0.05, 0.8), adjustPos=True, thing=None, **kwargs)
        if set rect to None, then colorbar steals self.a

    destroy ()
        does not clean up self.a, just kills the window if this plot owns the window

    ownCanvas = None
        checking self.showByDefault here causes trouble because it is hard to attach a figure to a window later for
        a general backend ***

    save (**keyArgs)
        make hardcopy of the current figure; specify filename or prefix keyword argument
```

## 6 Functions

```
hexrd.plotwrap.autoTicks (x, n, format='%0.2e')
```

```
hexrd.plotwrap.argToPW(arg)
hexrd.plotwrap.hist2D(xVals, yVals, bins, hRange=None, weights=None, **kwargs)
    Plot 2D histogram of data yVals versus xVals, with number of bins given by bins (int or 2-tuple)
hexrd.plotwrap.plotHist2D(xedges, yedges, H, hRange=None, logScale=False, minCount=1,
                           win=None, xlabel=None, ylabel=None, xformat=None, yformat=None,
                           nXTicks=0, nYTicks=0, winArgs={})
    winArgs can include things like title, relfigsize, dpi
hexrd.plotwrap.makeHist2D(xVals, yVals, bins, hRange=None, weights=None)
hexrd.plotwrap.main()
```

## 1.1.8 Module: `quadrature.q1db`

### 7 Functions

```
hexrd.quadrature.q1db.qloc1()
hexrd.quadrature.q1db.qloc2()
hexrd.quadrature.q1db.qloc3()
hexrd.quadrature.q1db.qloc4()
hexrd.quadrature.q1db.qloc5()
hexrd.quadrature.q1db.qloc8()
hexrd.quadrature.q1db.qLoc(quadr, promote=False)
```

## 1.1.9 Module: `quadrature.q2db`

### 5 Functions

```
hexrd.quadrature.q2db.qloc1()
hexrd.quadrature.q2db.qloc4()
hexrd.quadrature.q2db.qloc9()
hexrd.quadrature.q2db.qLocFrom1D(quadr1d)
    product of 1d quadrature rules; given accuracy may be available with fewer quadrature points using a native 3D rule
hexrd.quadrature.q2db.qLoc(quadr)
```

## 1.1.10 Module: `quadrature.q3db`

### 5 Functions

```
hexrd.quadrature.q3db.qloc1()
hexrd.quadrature.q3db.qloc8()
hexrd.quadrature.q3db.qloc27()
    3x3x3 quadrature, product of qloc1d03 in three directions
```



`hexrd.quadrature.q3db.qLocFrom1D(quadr1d)`  
 product of 1d quadrature rules; given accuracy may be available with fewer quadrature points using a native 3D rule

`hexrd.quadrature.q3db.qLoc(quadr)`

### 1.1.11 Module: `tens`

#### 4 Classes

**class** `hexrd.tens.T2Symm(args)`  
 template for symmetric second order tensor components

`__init__(args)`

**class** `hexrd.tens.T2Vecds(vecds)`  
 Bases: `hexrd.tens.T2Symm`

`__init__(vecds)`

**class** `hexrd.tens.T2Svec(val)`  
 Bases: `hexrd.tens.T2Symm`

`__init__(val)`

**class** `hexrd.tens.T2SvecP(svecp)`  
 Bases: `hexrd.tens.T2Symm`

`__init__(svecp)`

#### 23 Functions

`hexrd.tens.vecdvToVecds(vecdv)`  
 convert from `[t1,...,t5,v]` to `vecds[:]` representation, where `v` is the relative volume

`hexrd.tens.vecdsToSymm(vecds)`  
 convert from `vecds` representation to symmetry matrix

`hexrd.tens.traceToVecdsS(Akk)`

`hexrd.tens.vecdsSToTrace(vecdsS)`

`hexrd.tens.trace3(A)`

`hexrd.tens.symmToVecds(A)`  
 convert from symmetry matrix to `vecds` representation

`hexrd.tens.matxToSkew(A)`

`hexrd.tens.skewOfMatx(A)`

`hexrd.tens.matxToSymm(A)`

`hexrd.tens.symmOfMatx(A)`

`hexrd.tens.symmToMVvec(A)`  
 convert from symmetric matrix to Mandel-Voigt vector representation (JVB)

`hexrd.tens.MVvecToSymm(A)`  
 convert from Mandel-Voigt vector to symmetric matrix representation (JVB)

`hexrd.tens.MVCOBMatrix` (*R*)

GenerateS array of 6 x 6 basis transformation matrices for the Mandel-Voigt tensor representation in 3-D given by:

$$[A] = [[A_{11}, A_{12}, A_{13}], \\ [A_{12}, A_{22}, A_{23}], [A_{13}, A_{23}, A_{33}]]$$

*V*

$$\{A\} = [A_{11}, A_{22}, A_{33}, \sqrt{2}A_{23}, \sqrt{2}A_{13}, \sqrt{2}A_{12}]$$

where the operation  $R * A * R.T$  (in tensor notation) is obtained by the matrix-vector product  $[T]*\{A\}$ .

USAGE

$$T = \text{MVCOBMatrix}(R)$$

INPUTS

1. *R* is (3, 3) an ndarray representing a change of basis matrix

OUTPUTS

1. *T* is (6, 6), an ndarray of transformation matrices as described above

NOTES

1. Components of symmetric 4th-rank tensors transform in a manner analogous to symmetric 2nd-rank tensors in full matrix notation.

SEE ALSO

`symmToMVvec`, `quatToMat`

`hexrd.tens.NormalProjectionOfMV` (*vec*)

`hexrd.tens.svecToVecds` (*svec*)

convert from svec to vecds representation

`hexrd.tens.symmPlusI` (*Ain*)

add the identity to a symmetric matrix

`hexrd.tens.svecpToSvec` (*svecp*)

`hexrd.tens.symmToSvec` (*symm*)

`hexrd.tens.matxToSvec` (*matx*)

`hexrd.tens.svecToMatx` (*svec*)

`hexrd.tens.svecToSymm` (*svec*)

`hexrd.tens.dAiAoH_svecList` (*aInv*)

derivative of inverse of symmetric matrix wrt svec components of that matrix; *aInv* is the inverse of the matrix

`hexrd.tens.svecToSvecP` (*svec*)

### 1.1.12 Module: `valunits`

Module for associating units with scalar quantities

This module has been modified from its original form by removing the call to the “units” executable and restricting the units to only those used by the heXRD package.

## 2 Classes

**class** `hexrd.valunits.UNames`

Bases: `object`

Units used in this module

**class** `hexrd.valunits.valWUnit` (*name, unitType, value, unit*)

Value with units

**\_\_init\_\_** (*name, unitType, value, unit*)

Initialization

**INPUTS** *name*

(str) name of the item

**unitType** (str) class of units, e.g. ‘length’, ‘angle’

**value** (float) numerical value

**unit** (str) name of unit

**getVal** (*toUnit*)

Return value in requested units

**INPUTS**

**toUnit** (str) requested unit for output

**isAngle** ()

Return true if quantity is an angle

**isEnergy** ()

Return true if quantity is an energy

**isLength** ()

Return true if quantity is a length

## 2 Functions

`hexrd.valunits.toFloat` (*val, unitName*)

Return the raw value of the object

**INPUTS**

**val** (float|valWUnit) object with value

**unitName** (str) name of unit

This function returns the raw value of the object, ignoring the unit, if it is numeric or converts it to the requested units and returns the magnitude if it is a `valWUnit` instance.

For example:

```
>>> print toFloat(1.1, 'radians')
1.1
>>> v = valWUnit('vee', 'angle', 1.1, 'radians')
>>> print toFloat(v, 'degrees')
63.0253574644
```

hexrd.valunits.**valWithDflt** (*val, dflt, toUnit=None*)  
Return value or default value

## 1.1.13 Module: xrd.crystallography

### 1 Class

**class** hexrd.xrd.crystallography.**PlaneData** (*hkls, \*args, \*\*kwargs*)

Bases: `object`

Careful with ordering: Outputs are ordered by the 2-theta for the hkl unless you get `self.__hkls` directly, and this order can change with changes in lattice parameters (`lparms`); setting and getting exclusions works on the current hkl ordering, not the original ordering (in `self.__hkls`), but exclusions are stored in the original ordering in case the hkl ordering does change with lattice parameters

if not None, `tThWidth` takes priority over `strainMag` in setting two-theta ranges; changing `strainMag` automatically turns off `tThWidth`

**\_\_init\_\_** (*hkls, \*args, \*\*kwargs*)

**getDD\_tThs\_lparms** ()

derivatives of `tThs` with respect to lattice parameters; have not yet done coding for analytic derivatives, just wimp out and finite difference

**getHKLID** (*hkl*)

can call on a single hkl or list of hkls

**getHKLS** (*asStr=False, thisTTh=None, allHKLS=False*)

if pass `thisTTh`, then only return hkls overlapping the specified 2-theta; if set `allHKLS` to true, the ignore exclusions, `tThMax`, etc

**getLatticeOperators** ()

gets lattice vector operators as a new (deepcopy)

**getLatticeType** ()

This is the lattice type

**getLaueGroup** ()

This is the Schoenflies tag

**getNhklRef** ()

does not use exclusions or the like

**getPhaseID** ()

may return None if not set

**getPlaneNormals** ()

gets both `+(hkl)` and `-(hkl)` normals

**getPlaneSpacings** ()

gets plane spacings

**getSymHKLS** (*asStr=False, indices=None*)

new function that returns all symmetric hkls

**getTThRanges** (*strainMag=None, lparms=None*)

Return 2-theta ranges for included hkls

return array is n x 2

**get\_hkls** ()

do not do return self.\_\_hkls, as everywhere else hkls are returned in 2-theta order; transpose is to comply with lparm convention

**hkls**

do not do return self.\_\_hkls, as everywhere else hkls are returned in 2-theta order; transpose is to comply with lparm convention

**latVecOps**

gets lattice vector operators as a new (deepcopy)

**static makePlaneData** (*hkls, lparms, qsym, symmGroup, strainMag, wavelength*)

hkls : need to work with crystallography.latticePlanes lparms : need to work with crystallography.latticePlanes laueGroup : see symmetry module wavelength : wavelength strainMag : swag of strian magnitudes

**static makeScatteringVectors** (*hkls, rMat\_c, bMat, wavelength, chiTilt=None*)

modeled after QFromU.m

## 13 Functions

hexrd.xrd.crystallography.**hklToStr** (*x*)

hexrd.xrd.crystallography.**tempSetOutputDegrees** (*val*)

hexrd.xrd.crystallography.**revertOutputDegrees** ()

hexrd.xrd.crystallography.**cosineXform** (*a, b, c*)

Spherical trig transform to take alpha, beta, gamma to expressions for cos(alpha\*). See ref below.

[1] R. J. Neustadt, F. W. Cagle, Jr., and J. Waser, “Vector algebra and the relations between direct and reciprocal lattice quantities”. Acta Cryst. (1968), A24, 247–248

hexrd.xrd.crystallography.**processWavelength** (*arg*)

Convert an energy value to a wavelength. If argument has units of length or energy, will convert to globally specified unit type for wavelength (dUnit). If argument is a scalar, assumed input units are keV.

hexrd.xrd.crystallography.**latticeParameters** (*lvec*)

Generates direct and reciprocal lattice vector components in a crystal-relative RHON basis, X. The convention for fixing X to the lattice is such that  $a \parallel x1$  and  $c^* \parallel x3$ , where  $a$  and  $c^*$  are direct and reciprocal lattice vectors, respectively.

hexrd.xrd.crystallography.**latticePlanes** (*hkls, lparms, ltype='cubic', wavelength=1.54059292, strainMag=None*)

Generates lattice plane data in the direct lattice for a given set of Miller indices. Vector components are written in the crystal-relative RHON basis, X. The convention for fixing X to the lattice is such that  $a \parallel x1$  and  $c^* \parallel x3$ , where  $a$  and  $c^*$  are direct and reciprocal lattice vectors, respectively.

USAGE:

planeInfo = latticePlanes(hkls, lparms, \*\*kwargs)

INPUTS:

1.hkls (3 x n float ndarray) is the array of Miller indices for the planes of interest. The vectors are assumed to be concatenated along the 1-axis (horizontal).

2. `lparms` (1 x m float list) is the array of lattice parameters, where m depends on the symmetry group (see below).

3. The following optional keyword arguments are recognized:

**\*) `ltype=(string)` is a string representing the symmetry type of** the implied Laue group. The 11 available choices are shown below. The default value is 'cubic'. Note that each group expects a lattice parameter array of the indicated length and order.

latticeType lparms ———— 'cubic' a 'hexagonal' a, c 'trigonal' a, c 'rhombohedral' a, alpha (in degrees) 'tetragonal' a, c 'orthorhombic' a, b, c 'monoclinic' a, b, c, beta (in degrees) 'triclinic' a, b, c, alpha, beta, gamma (in degrees)

**\*) `wavelength=<float>` is a value represented the wavelength in** Angstroms to calculate bragg angles for. The default value is for Cu K-alpha radiation (1.54059292 Angstrom)

**\*) `strainMag=None`**

#### OUTPUTS:

1. `planeInfo` is a dictionary containing the following keys/items:

**`normals (3, n)` double array array of the components to the** unit normals for each {hkl} in X (horizontally concatenated)

**`dspacings (n, )` double array array of the d-spacings for** each {hkl}

**`2thetas (n, )` double array array of the Bragg angles for** each {hkl} relative to the specified wavelength

#### NOTES:

**\*) This function is effectively a wrapper to 'latticeVectors'.** See 'help(latticeVectors)' for additional info.

**\*) Lattice plane d-spacings are calculated from the reciprocal** lattice vectors specified by {hkl} as shown in Appendix 1 of [1].

#### REFERENCES:

[1] B. D. Cullity, "Elements of X-Ray Diffraction, 2 ed.". Addison-Wesley Publishing Company, Inc., 1978. ISBN 0-201-01174-3

```
hexrd.xrd.crystallography.latticeVectors(lparms, tag='cubic', radians=False, debug=False)
```

Generates direct and reciprocal lattice vector components in a crystal-relative RHON basis, X. The convention for fixing X to the lattice is such that  $a \parallel x_1$  and  $c^* \parallel x_3$ , where a and  $c^*$  are direct and reciprocal lattice vectors, respectively.

#### USAGE:

```
lattice = LatticeVectors(lparms, <symmTag>)
```

#### INPUTS:

1. `lparms` (1 x n float list) is the array of lattice parameters, where n depends on the symmetry group (see below).

2. `symTag` (string) is a case-insensitive string representing the symmetry type of the implied Laue group. The 11 available choices are shown below. The default value is 'cubic'. Note that each group expects a lattice parameter array of the indicated length and order.

latticeType lparms ———— 'cubic' a 'hexagonal' a, c 'trigonal' a, c 'rhombohedral' a, alpha (in degrees) 'tetragonal' a, c 'orthorhombic' a, b, c 'monoclinic' a, b, c, beta (in degrees) 'triclinic' a, b, c, alpha, beta, gamma (in degrees)

#### OUTPUTS:

1.lattice is a dictionary containing the following keys/items:

**F (3, 3) double array transformation matrix taking** components in the direct lattice (i.e. {uvw}) to the reference, X

**B (3, 3) double array transformation matrix taking** components in the reciprocal lattice (i.e. {hkl}) to X

**BR (3, 3) double array transformation matrix taking** components in the reciprocal lattice to the Fable reference frame (see notes)

**U0 (3, 3) double array transformation matrix** (orthogonal) taking components in the Fable reference frame to X

vol double the unit cell volume

**dparms (6, ) double list the direct lattice parameters:** [a b c alpha beta gamma]

**rparms (6, ) double list the reciprocal lattice** parameters: [a\* b\* c\* alpha\* beta\* gamma\*]

NOTES:

**\*) The conventions used for assigning a RHON basis, X -> {x1, x2, x3}, to each point group are consistent with those published in Appendix B of [1]. Namely:  $a \parallel x1$  and  $c^* \parallel x3$ . This differs from the convention chosen by the Fable group, where  $a^* \parallel x1$  and  $c \parallel x3$  [2].**

**\*) The unit cell angles are defined as follows:**  $\alpha = \arccos(b^*c/|b||c|)$ ,  $\beta = \arccos(c^*a/|c||a|)$ , and  $\gamma = \arccos(a^*b/|a||b|)$ .

**\*) The reciprocal lattice vectors are calculated using the** crystallographic convention, where the prefactor of  $2\pi$  is omitted. In this convention, the reciprocal lattice volume is  $1/V$ .

**\*) Several relations from [3] were employed in the component** calculations.

REFERENCES:

[1] J. F. Nye, “Physical Properties of Crystals: Their Representation by Tensors and Matrices”. Oxford University Press, 1985. ISBN 0198511655

[2] E. M. Lauridsen, S. Schmidt, R. M. Suter, and H. F. Poulsen, “Tracking: a method for structural characterization of grains in powders or polycrystals”. J. Appl. Cryst. (2001). 34, 744–750

[3] R. J. Neustadt, F. W. Cagle, Jr., and J. Waser, “Vector algebra and the relations between direct and reciprocal lattice quantities”. Acta Cryst. (1968), A24, 247–248

hexrd.xrd.crystallography.**hexagonalIndicesFromRhombohedral** (hkl)  
converts rhombohedral hkl to hexagonal indices

hexrd.xrd.crystallography.**rhombohedralIndicesFromHexagonal** (HKL)  
converts hexagonal hkl to rhombohedral indices

hexrd.xrd.crystallography.**rhombohedralParametersFromHexagonal** (a\_h, c\_h)  
converts hexagonal lattice parameters (a, c) to rhombohedral lattice parameters (a, alpha)

hexrd.xrd.crystallography.**getFriedelPair** (tth0, eta0, \*ome0, \*\*kwargs)  
Get the diffractometer angular coordinates in degrees for the Friedel pair of a given reflection (min angular distance).

AUTHORS:

10. (a)Bernier – 10 Nov 2009

USAGE:

**ome1, eta1 = getFriedelPair(tth0, eta0, \*ome0, display=False, units='degrees', convention='hexrd')**

## INPUTS:

- 1.tth0 is a list (or ndarray) of 1 or n the bragg angles (2theta) for the n reflections (tiled to match eta0 if only 1 is given).
- 2.eta0 is a list (or ndarray) of 1 or n azimuthal coordinates for the n reflections (tiled to match tth0 if only 1 is given).
- 3.ome0 is a list (or ndarray) of 1 or n reference oscillation angles for the n reflections (denoted omega in [1]). This argument is optional.
- 4.Keyword arguments may be one of the following:

Keyword Values{default} Action \_\_\_\_\_ 'display' True|{False} toggles display info to cmd line 'units' 'radians'|{'degrees'} sets units for input angles 'convention' 'fable'|{'hexrd'} sets conventions defining

the angles (see below)

**'chiTilt' None the inclination (about Xlab) of** the oscillation axis

## OUTPUTS:

- 1.ome1 contains the osciallition angle coordinates of the Friedel pairs associated with the n input reflections, relative to ome0 (i.e. ome1 = <result> + ome0). Output is in DEGREES!
- 2.eta1 contains the azimuthal coordinates of the Friedel pairs associated with the n input reflections. Output units are controlled via the module variable 'outputDegrees'

## NOTES:

**JVB) The ouputs ome1, eta1 are written using the selected convention, but the** units are alway degrees. May change this to work with Nathan's global...

**JVB) In the 'fable' convention [1], {XYZ} form a RHON basis where X is** downstream, Z is vertical, and eta is CCW with +Z defining eta = 0.

**JVB) In the 'hexrd' convention [2], {XYZ} form a RHON basis where Z is upstream,** Y is vertical, and eta is CCW with +X defining eta = 0.

## REFERENCES:

- [1] E. M. Lauridsen, S. Schmidt, R. M. Suter, and H. F. Poulsen, "Tracking: a method for structural characterization of grains in powders or polycrystals". J. Appl. Cryst. (2001). 34, 744–750
- [2] J. V. Bernier, M. P. Miller, J. -S. Park, and U. Lienert, "Quantitative Stress Analysis of Recrystallized OFHC Cu Subject to Deformed In Situ", J. Eng. Mater. Technol. (2008). 130. DOI:10.1115/1.2870234

hexrd.xrd.crystallography.getDparms (lp, lpTag, radians=True)

Utility routine for getting dparms, that is the lattice parameters without symmetry – 'triclinic'

## 1.1.14 Module: xrd.detector

### 24 Classes

**class** hexrd.xrd.detector.FmtCoordIdeal (planeData, workDist)

\_\_init\_\_ (planeData, workDist)

**class** hexrd.xrd.detector.ThreadReadFrame (img, readArgs, castArgs)  
Bases: threading.Thread



```

__init__(img, readArgs, castArgs)
class hexrd.xrd.detector.Framer2DRC(ncols, nrows, dtypeDefault='int16', dtypeRead='uint16',
                                   dtypeFloat='float64')
    Bases: object
    Base class for readers.

    You can make an instance of this class and use it for most of the things a reader would do, other than actually
    reading frames

    __init__(ncols, nrows, dtypeDefault='int16', dtypeRead='uint16', dtypeFloat='float64')

    getDark()
        needed in findSpotsOmegaStack

    getDeltaOmega()
        needed in findSpotsOmegaStack

    getEmptyMask()
        convenience method for getting an empty mask or bin frame

    getFrameOmega(iFrame=None)
        needed in findSpotsOmegaStack

    getNFrames()
        number of total frames with real data, not number remaining needed in findSpotsOmegaStack

    classmethod maxVal(dtypeRead)
        maximum value that can be stored in the image pixel data type; redefine as desired

    read(nskip=0, nframes=1, sumImg=False)
        needed in findSpotsOmegaStack

class hexrd.xrd.detector.FrameWriter(*args, **kwargs)
    Bases: hexrd.xrd.detector.Framer2DRC

    __init__(*args, **kwargs)

class hexrd.xrd.detector.ReadGeneric(filename, ncols, nrows, *args, **kwargs)
    Bases: hexrd.xrd.detector.Framer2DRC

    may eventually want ReadGE to inherit from this, or pull common things off to a base class

    __init__(filename, ncols, nrows, *args, **kwargs)

    getDark()
        no dark yet supported

    getFrameOmega(iFrame=None)
        if iFrame is none, use internal counter

    makeNew()
        return a clean instance for the same data files useful if want to start reading from the beginning

    read(nskip=0, nframes=1, sumImg=False)
        sumImg can be set to True or to something like numpy.maximum

    subtractDark = None
        keep things for makeNew convenience

class hexrd.xrd.detector.ReadGE(fileInfo, *args, **kwargs)
    Bases: hexrd.xrd.detector.Framer2DRC

    Read in raw GE files; this is the class version of the foregoing functions

```

## NOTES

- \*) The flip axis ('v'ertical) was verified on 06 March 2009 by JVB and UL.** This should be rechecked if the configuration of the GE changes or you are unsure.
- \*) BE CAREFUL! nframes should be < 10 or so, or you will run out of** memory in the namespace on a typical machine.
- \*)** The header is currently ignored
- \*) If a dark is specified, this overrides the use of empty frames as** background; dark can be a file name or frame
- \*) In multiframe images where background subtraction is requested but no** dark is specified, attempts to use the empty frame(s). An error is returned if there are not any specified. If there are multiple empty frames, the average is used.

**\_\_init\_\_** (*fileInfo, \*args, \*\*kwargs*)

meant for reading a series of frames from an omega sweep, with fixed delta-omega for each frame

omegaStart and omegaDelta can follow fileInfo or be specified in whatever order by keyword

fileInfo: string, (string, nempty), or list of (string, nempty) for multiple files

for multiple files and no dark, dark is formed only from empty frames in the first file

**classmethod display** (*thisframe, roi=None, pw=None, \*\*kwargs*)

this is a bit ugly in that it sidesteps the dtypeRead property

**getFrameOmega** (*iFrame=None*)

if iFrame is none, use internal counter

**getFrameUseMask** ()

this is an optional toggle to turn the mask on/off

**getNFrames** ()

number of total frames with real data, not number remaining

**indicesToMask** (*indices*)

Indices can be a list of indices, as from makeIndicesTThRanges

**makeNew** ()

return a clean instance for the same data files useful if want to start reading from the beginning

**classmethod maxVal** (*dummy*)

maximum value that can be stored in the image pixel data type

**rawRead** (*\*args, \*\*kwargs*)

wrapper around readRaw that does the same flipping as the reader instance from which it is called

**read** (*nskip=0, nframes=1, sumImg=False, mask=None*)

sumImg can be set to True or to something like numpy.maximum

**readBBox** (*bbox, raw=True, doFlip=None*)

with raw=True, read more or less raw data, with bbox = [(iLo,iHi),(jLo,jHi),(fLo,fHi)]

careful: if raw is True, must set doFlip if want frames potentially flipped; can set it to a reader instance to pull the doFlip value from that instance

**classmethod readDark** (*darkFile, nframes=1*)

dark subtraction is done before flipping, so do not flip when reading either

**classmethod readRaw** (*fname, mode='raw', headerlen=0*)

read a raw binary file; if specified, headerlen is in bytes; does not do any flipping

```

useThreading
    turn threading on or off

class hexrd.xrd.detector.ReadMar165 (mode)
    Bases: hexrd.xrd.detector.Framer2DRC

    placeholder; not yet really implemented

    __init__ (mode)

class hexrd.xrd.detector.ReadMar165NB1 (*args, **kwargs)
    Bases: hexrd.xrd.detector.ReadMar165

    __init__ (*args, **kwargs)

class hexrd.xrd.detector.ReadMar165NB2 (*args, **kwargs)
    Bases: hexrd.xrd.detector.ReadMar165

    __init__ (*args, **kwargs)

class hexrd.xrd.detector.ReadMar165NB3 (*args, **kwargs)
    Bases: hexrd.xrd.detector.ReadMar165

    __init__ (*args, **kwargs)

class hexrd.xrd.detector.ReadMar165NB4 (*args, **kwargs)
    Bases: hexrd.xrd.detector.ReadMar165

    __init__ (*args, **kwargs)

class hexrd.xrd.detector.LineStyles (lt=None)
    do not want to just cycle through default plot line colors, as end up with black lines

    __init__ (lt=None)

class hexrd.xrd.detector.Peak1DAtLoc (centers, xVecDflt=None)
    base class for 1D peak shapes at fixed location; fixed that is unless newCenter is passed to the __call__ method

    __init__ (centers, xVecDflt=None)
        If __init__ is called with a list, then put one peak at each location

    d_dCenters (xVec, p)
        derivative of call with respect to centers

    d_dp (xVec, p)
        derivative of call with respect to p assuming each eval depends only on its own point!

    d_dx (xVec, p)
        derivative of call with respect to xVec

    eval (xVec, p)
        xVec is parameters, p is positions

    fitFloatingCenter (tThVals, intensityVals, xVecGuess=None, centersGuess=None, weights=4, tTh-
                        Width=None, fitGoodnessTol=0.5)
        Note that centers are kept as they are – if you want to actually change the centers of the function you need
        to call setCenters(cFit) after calling this function

class hexrd.xrd.detector.PeakPV1DAtLoc (*args, **kwargs)
    Bases: hexrd.xrd.detector.Ppeak1DAtLoc

    the pseudo-Voigt:  $f = A * (n * fl + (1 - n) * fg)$ 

    __init__ (*args, **kwargs)

```

```
d_dx (xVec, p)
    allow p to be general shape
```

```
class hexrd.xrd.detector.PeakLorentzian1DAtLoc (*args, **kwargs)
    Bases: hexrd.xrd.detector.Peak1DAtLoc
    __init__ (*args, **kwargs)
    d_dx (xVec, p)
        allow p to be general shape
    getNParams ()
        2 parameters for background, 2 for intensity and width of each peak
```

```
class hexrd.xrd.detector.PeakGauss1DAtLoc (*args, **kwargs)
    Bases: hexrd.xrd.detector.Peak1DAtLoc
    __init__ (*args, **kwargs)
    d_dCenters (xVec, p)
        allow p to be general shape
    d_dx (xVec, p)
        allow p to be general shape
    getNParams ()
        2 parameters for background, 2 for intensity and width of each peak
```

```
class hexrd.xrd.detector.MultiRingBinned (detectorGeom, planeData, dataFrame, func-
                                         Type='pv', refineParamsDG=True, re-
                                         fineParamsL=False, targetNRho=None, po-
                                         larRebinKwargs={}, quadr=4, npdivMax=8,
                                         samplingFactor=1, singleRebin=True, distortion-
                                         FreeRefDG=False, log=None)
    like MultiRingEval, but works with polar rebinned (or 'caked') images
    no funcXVecList to init because expectation is that always pulled from dataFrame
    should work fine whether or not corrected is True in polarRebinKwargs; but note that default is changed to
    True
    if etaRange is not specified in polarRebinKwargs (or is None), then the etaRange is calculated based on numEta
    so that the first eta bin is centered around an angle of zero
    note that this works by matching intensities for binned 1D functions in a least squares problem; one could proba-
    bly instead form a residual on the two-theta values of the image frame positions for the peak centers found with
    independently floating centers (on the output of the getTThErrors method)
```

KEYWORD ARGS

```
funcType = funcTypeDflt, refineParamsDG = True, refineParamsL = False, targetNRho = 30, polar-
RebinKwargs = {}, quadr = 4, npdivMax = 4, samplingFactor = 0.25, singleRebin = True, distortionFreeRefDG
= False, log=None: if not None, then a file-like object with a "write" method;
__init__ (detectorGeom, planeData, dataFrame, funcType='pv', refineParamsDG=True, re-
         fineParamsL=False, targetNRho=None, polarRebinKwargs={}, quadr=4, npdivMax=8,
         samplingFactor=1, singleRebin=True, distortionFreeRefDG=False, log=None)
doFit (xVec0=None, **kwargs)
    kwargs can have things like ftol and xtol for leastsq
eval (xVec)
    careful: this updates the settings in detectorGeom and planeData
```

**floatingCentersIJ = None**

make a reference detector geom

**getTThErrors** (*plot=False, units='strain', outputFile=None*)

convenient way of looking at the errors, though not how the errors are actually measured in the fitting procedure; get the tTh values at the image frame locations deemed to be the centers with the floating-center fits

units can be: 'mm' <radius>, 'd-spacing', 'strain', 'radians' <tTh>, 'degrees' <tTh>

**logfile**

file for log messages

**plotByRingEta** (*iRingSet, iEta, win=None, sqrtIntensity=True, alpha=0.25*)

may have redundant work here, but assume this is not a big deal if doing plots

**polImg = None**

tth values for figuring out where the rings fall

**prbkw = None**

things from the user

**rhoPxRange = None**

and now compute number of rho bins across all ring sets

**ticMethod = None**

defaults which want different from those for polarRebin's defaults

**wQP = None**

leastsq to do fit, with floating center

**class hexrd.xrddetector.MultiRingEval** (*detectorGeom, planeData, indicesList=None, iHKLLLists=None, dataFrame=None, funcType='pv', refineParamsDG=True, refineParamsL=False, funcXVecList=None, copyFrame=False, quadr=3*)

For working with data as rings, particularly for fitting detector geometry or lattice parameters.

**\_\_init\_\_** (*detectorGeom, planeData, indicesList=None, iHKLLLists=None, dataFrame=None, funcType='pv', refineParamsDG=True, refineParamsL=False, funcXVecList=None, copyFrame=False, quadr=3*)

Mostly meant for use with DetectorGeomGE.fit

If funcXVecList is passed, then entries in this list are used for peak function forms, and these peak function forms do not appear in the degrees of freedom

Note that ranges for 2-thetas from planeData need to be such that rings are adequately covered

Can optionally pass indicesList and iHKLLLists if they are already handy

if copyFrame is True, then data in dataFrame is copied

**deval** (*xVec*)

useful to pass, for example, as Dfun to leastsq; bit of a misnomer in that deval is the derivative of \_\_call\_\_, not eval

**doFit** (*xVec0=None, \*\*lsKwargs*)

lsKwargs can have things like ftol and xtol for leastsq

**eval** (*xVec, thisframe=None*)

if thisframe is passed, the put values on the frame

**jQP = None**

do not worry about dvQP kinds of contributions for now

**radialFitXVec** (*dataFrame=None, plot=False, plotTitlePrefix='', quadr1d=None*)

if *dataFrame* is not provided, use *self.dataFrame*

if *quadr1d* is not specified, use *quadr* specified in *init*

**radialPlotData** (*dataFrame=None, plotTitlePrefix=''*)

for simple radial plotting, useful if other things are mysteriously breaking

**setFuncXVecList** (*funcXVecList*)

only okay if *funcXVecList* set on *init*

**class** `hexrd.xrd.detector.DetectorBase` (*reader*)

Bases: `object`

base class for a detector

**\_\_init\_\_** (*reader*)

**getPVecScaling** ()

scaling, suitable for scaling perturbations for finite differencing

**getParamScalings** ()

scalings, suitable for scaling perturbations for finite differencing

**class** `hexrd.xrd.detector.Detector2DRC` (*ncols, nrows, pixelPitch, vFactorUnc, vDark, reader, \*args, \*\*kwargs*)

Bases: `hexrd.xrd.detector.DetectorBase`

base class for 2D row-column detectors

**\_\_init\_\_** (*ncols, nrows, pixelPitch, vFactorUnc, vDark, reader, \*args, \*\*kwargs*)

**angOnDetector** (*tTh, eta, \*args*)

note: returns a scalar if *tTh* and *eta* have single entries

**angToXYO** (*x0, y0, \*args, \*\*kwargs*)

convert Cartesian to polar

uses blocking to call vectorized version

**angToXYOBBBox** (*\*args, \*\*kwargs*)

given either *angBBox* or *angCOM* (angular center) and *angPM* (+/-values), compute the bounding box on the image frame

if *forSlice=True*, then returned *bbox* is appropriate for use in array slicing

if *reader* or *omegas* is passed, then convert from *omegas* to frames; and if *doWrap=True*, then frames may be a list for an *omega* range that spans the branch cut

**angToXYO\_V** (*tth, eta\_l, \*args, \*\*kwargs*)

opposite of *xyoToAng*

**cartesianCoordsOfPixelIndices** (*row, col, ROI=None*)

converts *[i, j]* pixel array indices to cartesian spatial coords where the lower left corner of the image is (0, 0)

Output units are in the pixel pitch units (see *self.pixelPitch*)

Will optionally take the upper left-hand corner (min row, min col) of a ROI when dealing with subregions on the detector as in when zooming in on diffraction spots...

**\***) explicitly enforce this to be self-consistent with radial distortion correction, etc...

**display** (*thisframe, planeData=None, \*\*kwargs*)

wrap reader display method; display coordinates as 2-theta and eta given that *self* knows how to do this

if pass planeData, then it is used to list HKLs overlapping the given 2-theta location

...\*\*\* option for drawing lab-frame glyph

**displayIdeal** (*thisframe*, *planeData=None*, *workDist=None*, *nlump=None*, *\*\*kwargs*)  
render and display frame on ideal detector plane; if workDist is not specified, then use self.workDist

**drawRings** (*drawOn*, *planeData*, *withRanges=False*, *legendLoc=(0.05, 0.5)*, *legendMaxLen=10*,  
*ideal=None*, *range=None*, *lineType=None*, *lineWidth=1.0*)

If drawOn is a PlotWrap instance, draw on the existing instance, otherwise pass drawOn to display and return the resulting PlotWrap instance

planeData.exclusions can be used to work with a subset of rings;

set legendLoc to None or False to skip making the legend

removes any existing lines in the axes

if pass ideal, then display rings on an ideal detector with the working distance taken from the value of the ideal argument

**drawRingsGUI** (*thisframe*, *planeData*, *displayKWArgs={}*, *sliderRangeFactor=1.0*, *funcType='pv'*)  
a simple GUI

**getAngPixelSize** (*xyo*, *delta\_omega*)  
get pixel size in angular coordinates at a given cartesian coordinate position

**getPRBOverlay** (*polarRebinKWArgs*)  
Return plottable coordinates of rebinning sector.  
Takes in dictionary of keyword args for polarRebin  
for etas, right now assumes stopEta > startEta, CCW

**getParamScalings** ()  
scalings, suitable for scaling perturbations for finite differencing

**getRings** (*planeData*, *ranges=False*)  
Return a list of rings for the given hkl  
Already filters on the exclusions.

**getVScale** (*vThese*)  
get scale factors for use in uncertainty quantification

**makeIndicesTThRanges** (*planeData*, *cullDupl=False*)  
return a list of indices for sets of overlapping two-theta ranges; to plot, can do something like:

mask = self.reader.getEmptyMask()

mask[indices] = True

With cullDupl set true, eliminate HKLs with duplicate 2-thetas

**makeMaskTThRanges** (*planeData*)  
Mask in the sense that reader with the mask will exclude all else

**pixelIndicesOfCartesianCoords** (*x*, *y*, *ROI=None*)  
converts [i, j] pixel array indices to cartesian spatial coords where the lower left corner of the image is (0, 0)

Output units are in the pixel pitch units (see self.pixelPitch)

Will optionally take the upper left-hand corner (min row, min col) of a ROI when dealing with subregions on the detector as in when zooming in on diffraction spots...

*\*)* explicitly enforce this to be self-consistent with radial distortion correction, etc...

**polarRebin** (*thisFrame*, *npdiv*=2, *rhoRange*=[100, 1000], *numRho*=1200, *etaRange*=array([-0.08726646, 6.19591884]), *numEta*=36, *ROI*=None, *corrected*=False, *verbose*=True, *log*=None)

Caking algorithm

#### INPUTS

*thisFrame* *npdiv*=2, pixel subdivision (n x n) to determine bin membership *rhoRange*=[100, 1000] - radial range in pixels *numRho*=1200 - number of radial bins *etaRange*=num.pi\*num.r\_[-5, 355]/180. - range of eta *numEta*=36 - number of eta subdivisions *ROI*=None - region of interest (four vector) *corrected*=False - uses 2-theta instead of rho *verbose*=True,

**renderIdeal** (*thisframe*, *nlump*=None, *workDist*=None)

render the frame on an ideal detector plane; returns interpolated frame data *zi* on a regular grid *xi*, *yi*; suitable for use with *pcolormesh*(*xim*, *yim*, *zi*), with *xim*, *yim* = *num.meshgrid*(*xi*, *yi*); note that *pcolormesh* is used instead of *pcolor* because *zi* may be a masked array

**xyoToAng** (*x0*, *y0*, *\*args*, *\*\*kwargs*)

convert Cartesian to polar

uses blocking to call vectorized version

**xyoToAngAll** ()

get angular positions of all pixels

**xyoToAngCorners** ()

get angular positions of corner pixels

**xyoToAngMap** (*x0*, *y0*, *\*args*, *\*\*kwargs*)

eta by default is in [-pi,pi] if all data are in the left quadrants, remap eta into [0,2\*pi]

**xyoToAng\_V** (*x0*, *y0*, *\*args*, *\*\*kwargs*)

Convert radial spectra obtained from polar rebinned powder diffraction images to angular spectra.

USAGE: *mappedData* = *XFormRadialSpectra*(*t*, *D*, *tilt*, *xydata*, *azim*, *tthRange*, *radDistFuncHandle*, *radDistArgs*)

#### INPUTS:

1) *t* is 2 x 1 (double), the origin translation. The convention is from 'true' to 'estimated' centers. 2) *D* is 1 x 1 (double), the sample-to-detector distance in mm. 3) *gammaYprime* is 1 x 1 (double), the angle between the 'ideal' and 'experimental' X-axes (horizontal). 4) *gammaXhatPrime* is 1 x 1 (double), the angle between the 'ideal' and 'experimental' Y-axes (vertical). 5) *xydata* is 1 x n (cell), the cell array of data 6) *azim* 7) *tthRange* 8) *radDistFuncHandle*

#### OUTPUT:

1) *mappedData* is 1 x n (cell), the cell array of mapped radial data corresponding to the input 'xydata'.

**class** *hexrd.xrd.detector.DetectorGeomMar165* (*\*args*, *\*\*kwargs*)

Bases: *hexrd.xrd.detector.Detector2DRC*

**\_\_init\_\_** (*\*args*, *\*\*kwargs*)

**radialDistortion** (*xin*, *yin*, *invert*=False)

no distortion correction

**class** *hexrd.xrd.detector.DetectorGeomGE* (*\*args*, *\*\*kwargs*)

Bases: *hexrd.xrd.detector.Detector2DRC*

handle geometry of GE detector, such as geometric and radial distortion corrections; *x* and *y* are in pixels, as is *rho*; pixels are numbered from (0,0);

**\_\_init\_\_** (*\*args*, *\*\*kwargs*)



**radialDistortion** (*xin, yin, invert=False*)

Apply radial distortion to polar coordinates on GE detector

xin, yin are 1D arrays or scalars, assumed to be relative to self.xc, self.yc Units are [mm, radians]. This is the power-law based function of Bernier.

Available Keyword Arguments :

invert = True or >False< :: apply inverse warping

**class** hexrd.xrd.detector.**DetectorGeomFrelon** (*\*args, \*\*kwargs*)

Bases: [hexrd.xrd.detector.Detector2DRC](#)

handle geometry of GE detector, such as geometric and radial distortion corrections; x and y are in pixels, as is rho; pixels are numbered from (0,0);

**\_\_init\_\_** (*\*args, \*\*kwargs*)

**radialDistortion** (*xin, yin, invert=False*)

Apply radial distortion to polar coordinates on GE detector

xin, yin are 1D arrays or scalars, assumed to be relative to self.xc, self.yc Units are [mm, radians]. This is the power-law based function of Bernier.

$$(p[0]*(ri/rx)**p[3] * \text{num.cos}(2.0 * ni) + p[1]*(ri/rx)**p[4] * \text{num.cos}(4.0 * ni) + p[2]*(ri/rx)**p[5] + 1)*ri$$

$$1 + p[0]*(ri/rx)**p[2] * \text{num.cos}(p[4] * ni) + p[1]*(ri/rx)**p[3]$$

Available Keyword Arguments :

invert = True or >False< :: apply inverse warping

**class** hexrd.xrd.detector.**DetectorGeomQuadGE** (*\*args, \*\*kwargs*)

Bases: [hexrd.xrd.detector.DetectorBase](#)

No global parameters – all detector parameters hang off of the sub-detectors; although some data are stored off of this higher level class for convenience

**\_\_init\_\_** (*\*args, \*\*kwargs*)

pass ReadGE instance as the reader for now; perhaps make a ReadQuadGE class later if it turns out to be needed

**dgDummy = None**

cleanup after auto-parsing of keyword args

**displayIdeal** (*framesQuad, planeData=None, workDist=None, nlump=None, doFmtCoord=True, \*\*kwargs*)

display all sub-detectors on an idealized detector plane

If matplotlib gets around to enabling the transform argument to imshow, that might be a much faster approach than what is currently done here, although what is done here is nice in that it takes account of all of the distortions, not just the in-plane rotation. The idea would be that the in-plane rotation would be, by far, the biggest effect. `import matplotlib.transforms as mtransforms tr = mtransforms.Affine2D() tr.rotate(self.zTilt) imshow( , transform=tr)`

**displaySub** (*iQuad, thisframe, planeData=None, \*\*kwargs*)

convenience for displaying a sub-detector ...\*\*\* need to code labAxesGlyph support in display

**drawRings** (*drawOn, planeData, workDist=None, \*\*kwargs*)

assumes ideal geometry, as from displayIdeal

**fitProcedureA** (*planeData*, *framesQuad*, *iRefQuad*=0, *funcType*='pv', *funcXVecList*=None, *quadr*=1, *doGUI*=0, *doMRingPlot*=False)

A procedure for fitting the set of detectors; do not need to click 'fit' button in GUI – done inside the procedure.

Watch out – MultiRingEval instances a memory hogs, especially while creating Jacobian matrices!

If want to just refine detector geometry and not the functional forms for the rings, pass *funcXVecList* as True or as something like a list of arrays from MultiRingEval.getFuncXVecList()

**classmethod getRefineFlagsDflt** ()

no parameters to refine for this detector; call fitProcedureA for a procedure to refine individual sub-detectors

**getTThMax** ()

min over sub-detectors, where for each sub-detector max two-theta is evaluated as the max over points checked in getTThMax for the sub-detector

**setCentersFromRef** (*iRefQuad*=0)

this assumes all of the tilts are the same

**setQuadOffsets** (*iRefQuad*=0)

this assumes all of the tilts are the same

## 13 Functions

hexrd.xrd.detector.**angToXYIdeal** (*tTh*, *eta*, *workDist*)

hexrd.xrd.detector.**mapAngs** (*eta*, *doMap*=None)

hexrd.xrd.detector.**getCentered** (*vmin*, *vmax*)

hexrd.xrd.detector.**getCMap** (*spec*)

hexrd.xrd.detector.**omeRangeToFrameRange** (*omeA*, *omeB*, *omegaStart*, *omegaDelta*, *nFrames*, *checkWrap*=True, *slicePad*=1)

assumes omegas are evenly spaced omegaDelta may be negative

hexrd.xrd.detector.**frameInRange** (*iFrame*, *frameRange*)

for use with output from omeRangeToFrameRange; trust that slicePad=1 was used in omeRangeToFrameRange

hexrd.xrd.detector.**getNFramesFromBytes** (*fileBytes*, *nbytesHeader*, *nbytesFrame*)

hexrd.xrd.detector.**mar165IDim** (*mode*)

hexrd.xrd.detector.**getOmegaMMReaderList** (*readerList*, *overall*=False)

get omega min/max information from a list of readers

hexrd.xrd.detector.**detectorList** ()

hexrd.xrd.detector.**newDetector** (*detectorType*, *\*args*, *\*\*kwargs*)

Return a detector of the requested type

### INPUTS

detectorType - a string in the detector type list [see detectorList()]

hexrd.xrd.detector.**newGenericReader** (*ncols*, *nrows*, *\*args*, *\*\*kwargs*)

currently just returns a Framer2DRC

hexrd.xrd.detector.**newGenericDetector** (*ncols*, *nrows*, *pixelPitch*, *\*args*, *\*\*kwargs*)

If reader is passed as None, then a generic reader is created

**Keyword Arguments:** vFactorUnc vDark reader readerKwargs getDParamDflt setDParamZero getDParamScalings getDParamRefineDflt radialDistortion

If *\*args* is an existing detector geometry, then additional keyword arguments may include:

pVec

If *\*args* is (xc, yc, workDist, xTilt, yTilt, zTilt) detector parameters, then additional keyword arguments may include:

distortionParams

### 1.1.15 Module: `xrd.distortion`

#### 2 Functions

`hexrd.xrd.distortion.dummy(xy_in, params, invert=False)`

`hexrd.xrd.distortion.GE_41RT(xy_in, params, invert=False)`

Apply radial distortion to polar coordinates on GE detector

xin, yin are 1D arrays or scalars, assumed to be relative to self.xc, self.yc Units are [mm, radians]. This is the power-law based function of Bernier.

Available Keyword Arguments :

invert = True or >False< :: apply inverse warping

### 1.1.16 Module: `xrd.experiment`

Module for wrapping the main functionality of the xrd package.

The Experiment class is the primary interface. Other classes are helpers.

#### 9 Classes

**class** `hexrd.xrd.experiment.FitModes`

Bases: `object`

Indicators for single-frame or multiframe data files

**class** `hexrd.xrd.experiment.ImageModes`

Bases: `object`

Indicators for single-frame or multiframe data files

**class** `hexrd.xrd.experiment.Experiment` (`cfgFile='/home/docs/checkouts/readthedocs.org/user_builds/hexrd/envs/0.2.x/local/lib/python2.7/site-packages/hexrd/data/materials.cfg'`,  
`matFile='/home/docs/checkouts/readthedocs.org/user_builds/hexrd/envs/0.2.x/local/lib/python2.7/site-packages/hexrd/data/all_materials.cfg'`)

Bases: `object`

Wrapper for xrd functionality

**\_\_init\_\_** (`cfgFile='/home/docs/checkouts/readthedocs.org/user_builds/hexrd/envs/0.2.x/local/lib/python2.7/site-packages/hexrd/data/materials.cfg'`, `matFile='/home/docs/checkouts/readthedocs.org/user_builds/hexrd/envs/0.2.x/local/lib/python2.7/site-packages/hexrd/data/all_materials.cfg'`)

Constructor for Experiment

INPUTS

**cfgFile** – name of the config file to use for initialization; an empty string indicates that default values for options are used

**matFile** – name of the materials data file; a real file name is required here

**activeImage**

Active image

**activeMaterial**

Active Material

Can be set by number (index in material list) or by name.

On output, it is always a material instance.

**activeReader**

Get method for activeReader

Reader is set by using index in reader list or by name.

**active\_img**

Current image

**add\_to\_img\_list** (*name*)

Append the active image to the image list

**calInput**

(read only) Calibration input data

**calibrate** (*log=None*)

Calibrate the detector

Currently, uses polar rebin only.

**clear\_reader** ()

Close current reader

**clear\_spots** ()

Reset the list of spots

**curFrameNumber**

Current frame number

**detector**

(read only) detector

**dump\_grainList** (*f*)

dump grainList to cPickle

**export\_grainList** (*f, dspTol=None, etaTol=None, omeTol=None, doFit=False, sort=True*)

export method for grainList

**find\_raw\_spots** ()

find spots using current reader and options

**fitRMats**

(get-only) Rotation matrices from indexing

**getSavedReader** (*which*)

Get a specified reader

**get\_spots\_ind** ()

Select spots for indexing

**hydra**  
(read only) hydra image class

**img\_list**  
(get only) List of saved images (get only)

**img\_names**  
(get-only) List of names for saved images

**index\_opts**  
(get-only) Options for indexing

**loadDetector** (*fname*)  
Load the detector information from a file  
INPUTS *fname* – the name of the file to load from

**loadMaterialList** (*fname*)  
Load the pickled material list from a file  
INPUTS *fname* – the name of the file to load from

**loadRawSpots** (*fname*)  
Load the detector information from a file  
INPUTS *fname* – the name of the file to load from

**loadReaderList** (*fname*)  
Load the reader list from a file  
INPUTS *fname* – the name of the file to load from

**matDict**  
(read only) Dictionary mapping material names to material

**matList**  
List of materials

**matNames**  
(read only) List of material names

**newDetector** (*gp*, *dp*)  
Create a new detector with given geometry and distortion parameters  
*gp* - initial geometric parameters *dp* - initial distortion parameters

**newMaterial** ()  
Create a new material and add it to the list

**newReader** ()  
Add new reader to the list and make it active  
Changes name if necessary.

**numFramesTotal**  
Number of frames available for reading

**raw\_spots**  
(get-only) spots from image before culling and association with rings

**readImage** (*frameNum=1*)  
Read and return an image  
DESCRIPTION  
This reads an image according to the active reader specification, saving it in the `activeImage` attribute.

**readerListAddCurrent** ()  
Add current list to list of saved readers

**readerNames**  
Return list of saved readers

**refineFlags**  
(read only) refinement flags for calibration

**refine\_grains** (*minCompl*, *nSubIter*=3, *doFit*=False, *etaTol*=valWUnit("etaTol", "ANGLE", 1.0, "degrees"), *omeTol*=valWUnit("etaTol", "ANGLE", 1.0, "degrees"), *fineDspTol*=0.005, *fineEtaTol*=valWUnit("etaTol", "ANGLE", 0.5, "degrees"), *fineOmeTol*=valWUnit("etaTol", "ANGLE", 0.5, "degrees"))  
refine a grain list

**run\_indexer** ()  
Run indexer

**saveDetector** (*fname*)  
Save the detector information to a file  
  
INPUTS *fname* – the name of the file to save in

**saveRMats** (*f*)  
save rMats to npy file

**saveRawSpots** (*fname*)  
Save the detector information to a file  
  
INPUTS *fname* – the name of the file to save in

**saveReaderList** (*fname*)  
Save the reader list to a file  
  
INPUTS *fname* – the name of the file to save in

**savedReaders**  
Return list of saved readers

**Experiment.simulateGrain** (*rMat*=array([[ 1., 0., 0.],  
[ 0., 1., 0.],  
[ 0., 0., 1.]]), *vMat*=array([ 1., 1., 1., 0., 0., 0.]), *planeData*=None, *detector*)  
Simulate a grain with choice of active material

**spotOpts**  
(get-only) spot finding options

**spot\_readers**  
(get-only) list of readers used to generate spots

**spots\_for\_indexing**  
(get-only) spots associated with rings

**class** hexrd.xrd.experiment.**ReaderInput** (*name*='reader', *desc*='no description')

Bases: `object`

ReaderInput

This class is for holding input required to instantiate a reader object. Currently, only GE reader is supported.

**\_\_init\_\_** (*name*='reader', *desc*='no description')

Constructor for ReaderInput

INPUT *name* – [optional] (str) *name* *desc* – [optional] (str) *description*

NOTES \* currently only GE reader is supported

**RC**

alias of ReadGE

**aggMode**

Mode identifier for frame aggregation

**aggModeOp**

(read only) option to pass to GE reader instances for aggregation mode

**darkFile**

Full pathname of dark file

**getNumberOfFrames ()**

Return number of frames available in data files

**hasImages**

(get only) true if list of images has been set

**imageNames**

Get method for imageNames

**makeReader ()**

Return a reader instance based on self

**setOmegaInfo (imgName, omin, omax, odel)**

Set omega info for the specified image

**class** hexrd.xrd.experiment.**CalibrationInput** (*mat, xtol=1e-06*)

Bases: `object`

CalibrationInput

**\_\_init\_\_** (*mat, xtol=1e-06*)

Constructor for CalibrationInput

**cakeArgs**

(get only) Keyword arguments for polar rebinning

**calData**

(get only) Lattice parameter data for calibrant

This provides a deepcopy with wavelength, strain magnitude and two-theta width set.

**calMat**

Calibration material (calibrant)

**fitType**

fit type: direct or caked

**class** hexrd.xrd.experiment.**DetectorInfo** (*gParams=[], dParams=[]*)

Bases: `object`

Class for detector and associated data

**\_\_init\_\_** (*gParams=[], dParams=[]*)

Constructor for detectorInfo

**calibrate** (*calInp, rdrInp, mat, log=None*)

Calibrate this detector using specified reader and options

**class** hexrd.xrd.experiment.**PolarRebinOpts**

Bases: `object`

Options for polar rebinning

`__init__()`

Constructor for PolarRebinOpts

This routine sets default values for caking options.

**The following attributes (with initial values) can be modified directly.** etaMin = 0 etaMax = 360  
rhoMin = 100 rhoMax = 1000 numEta = 36 numRho = 500 correct = True

**kwargs**

(get only) Return keyword args to pass to polarRebin

**class** `hexrd.xrd.experiment.SpotOptions`

Bases: `object`

Manage options available for spot finding and analysis

Mainly, this manages the keyword options to the `findSpotsOmegaStack()` static method in the `Spots` class.

`__init__()`

SpotOptions Constructor

**class** `hexrd.xrd.experiment.IndexOptions`

Bases: `object`

indexOptions

`__init__()`

Constructor for indexOptions

## 5 Functions

`hexrd.xrd.experiment.newName(name, nlist)`

return a name not in the list, but based on name input

`hexrd.xrd.experiment.saveExp(e, f)`

save experiment to file

`hexrd.xrd.experiment.loadExp(inpFile, matFile='/home/docs/checkouts/readthedocs.org/user_builds/hexrd/envs/0.2.x/local_packages/hexrd/data/materials.cfg')`

Load an experiment from a config file or from a saved exp file

**inpFile** – the name of either the config file or the saved exp file; empty string means start new experiment

**matFile** – name of the materials file

`hexrd.xrd.experiment.refineDetector(grainList, scl=None, gtol=1e-06)`

`hexrd.xrd.experiment.objFunc(x, grainList, scl)`

### 1.1.17 Module: `xrd.fitting`

#### 7 Functions

`matchOmegas(xyo_det, hkls_idx, chi, rMat_c, bMat, wavelength, vInv=array([ 1., 1., 1., 0.  
[-0.],  
[-1.]]), etaVec=array([[ 1.],  
[ 0.]`



```
[ 0.]), omePeriod=None)
```

For a given list of (x, y, ome) points, outputs the index into the results from `oscillAnglesOfHKLs`, including the calculated omega values.

```
hexrd.xrd.fitting.geomParamsToInput (tiltAngles,  chi,  expMap_c,  tVec_d,  tVec_s,  tVec_c,
                                         dParams)
```

`hexrd.xrd.fitting.inputToGeomParams` ( $p$ )

```
calibrateDetectorFromSX(xyo_det, hkls_idx, bMat, wavelength, tiltAngles, chi, expMap_c, tVec=[-0.,  
[-1.]], etaVec=array([[ 1.],  
[ 0.],  
[ 0.])), distortion=(<function GE_4lRT at 0x7fad8697cb18>, [0.0, 0.0, 0.0, 2.0, 2.0, 2], asBool=False,  
False, False, False, False, False, False, False), dtype=bool), pScl=array([1, 1, 1, 1, 1, 1,
```

```
hexrd.xrd.fitting.objFuncSX(pFit, pFull, pFlag, dFunc, dFlag, xyo_det, hkl_idx, bMat, vInv,
                             wavelength, bVec, eVec, omePeriod, simOnly=False, returnScalarValue=False)
```

```
fitGrain(xyo_det, hkls_idx, bMat, wavelength, detectorParams, expMap_c, tVec_c, vInv, beamV
[-0.],
[-1.]], etaVec=array([[ 1.],
[ 0.],
[ 0.]]), distortion=(<function GE_4lRT at 0x7fad8697cb18>, [0.0, 0.0, 0.0, 2.0, 2.0, 2]), c
True, True, True], dtype=bool), gScl=array([ True,  True,  True,  True,  True,  True,  T
True,  True,  True], dtype=bool), omePeriod=None, factor=0.1, xtol=0.0001, ftol=0.0001)
```

```
hexrd.xrd.fitting.objFuncFitGrain(gFit, gFull, gFlag, detectorParams, xyo_det, hkl_idx, bMat,
wavelength, bVec, eVec, dFunc, dParams, omePeriod, simOnly=False, returnScalarValue=False)
```

```
gFull[0] = expMap_c[0] gFull[1] = expMap_c[1] gFull[2] = expMap_c[2] gFull[3] = tVec_c[0] gFull[4] =
tVec_c[1] gFull[5] = tVec_c[2] gFull[6] = vInv_MV[0] gFull[7] = vInv_MV[1] gFull[8] = vInv_MV[2] gFull[9]
= vInv_MV[3] gFull[10] = vInv_MV[4] gFull[11] = vInv_MV[5]
```

```
detectorParams[0] = tiltAngles[0] detectorParams[1] = tiltAngles[1] detectorParams[2] = tiltAngles[2] detector-
Params[3] = tVec_d[0] detectorParams[4] = tVec_d[1] detectorParams[5] = tVec_d[2] detectorParams[6] = chi
detectorParams[7] = tVec_s[0] detectorParams[8] = tVec_s[1] detectorParams[9] = tVec_s[2]
```

### 1.1.18 Module: xrd.grain

## 1 Class

```
class hexrd.xrd.grain.Grain (spots, refineFlags=None, pVec=None, grainData=None, **kwargs)
```

Bases: object

A (maybe) indexed grain

method to fit: centroid, orientation, strain, strain+orientation; small and large-strain versions indices into spots  
a reference to spots? reference lattice parameters – not planeData in case it gets changed with pressure

fitting methods for orientation, stretch, and centroid?

what happens if fit a spot and the fit is bad? what if decide to refine the spot into two spots for clear cases of modest overlap? does that happen often enough that we need to worry about it? should Spots class handle a change in spot numbers: NO can Spot fit methods easily be generalized? Spot should probably barf if asked for fit center if multiple peaks found unless an index is given for which peak set claimedBy for spots that are found to be bad? – yes, and then if another grain wants to claim the spot, it can ask the claiming grain to hand over the spot or tell it whether there are multiple peaks or whatever

`__init__` (*spots*, *refineFlags=None*, *pVec=None*, *grainData=None*, *\*\*kwargs*)

**bMat**

Returns the reciprocal lattice vector components consistent with the stretch tensor. components are written in the CRYSTAL FRAME.

**checkClaims** ()

useful if have not done claims yet and want to check and see if spots are still available; updates completeness too

**claimSpots** (*asMaster=None*)

claim spots; particularly useful if claimingSpots was False on init; assume conflicts are handled elsewhere or ignored if want to claim spots using this method;

**fMat**

Returns the lattice vector components consistent with the stretch tensor. Components are written in the CRYSTAL FRAME.

**fit** (*xtol=1e-12*, *ftol=1e-12*, *fitPVec=True*, *display=True*, *fout=None*)

Fits the cell distortion and orientation with respect to the reference in terms of the deformation gradient  $F = R * U$  where R is proper orthogonal and U is symmetric positive-definite; i.e. the right polar decomposition factors.

**fitPrecession** (*weighting=False*, *display=True*, *xtol=1e-12*, *ftol=1e-12*, *fout=None*)

Fit the Center-Of-Mass coordinates of the grain in the sample frame

**getAlignmentRotation** ()

$\text{num.dot}(q, \text{num.eye}(3) - 2 * \text{num.diag}(\text{num.diag}(\text{num.dot}(r.T, fMat)) < 0))$

**getCellVolume** ()

Returns the volume of the direct lattice consistent with the stretch tensor.

**getFitResid** (*fitPVec=True*, *norm=None*)

returns as shape (n,3), so that len of return value is number of vectors

**getLatticeParams** ()

Returns the lattice parameters consistent with stretch tensor

**getLatticeVectors** ()

Returns the lattice vector components consistent with the stretch tensor. Components are written in the CRYSTAL FRAME.

**getReciprocalAlignmentRotation** ()

**getReciprocalLatticeVectors** ()

Returns the reciprocal lattice vector components consistent with the stretch tensor. components are written in the CRYSTAL FRAME.

**getReferenceLatticeParams** ()

Return the reference lattice parameters stored on self

**getRightStretchTensor** ()

Returns the components of the right stretch tensor, which is symmetric positive-definite. Components are written in the CRYSTAL FRAME. The output is calculated as:  $U = R^T * V * R$  This is for convenience and cannot be set independently to preserve self-consistency.

**getStretchTensor** ()

Returns the components of the left stretch tensor, which is symmetric positive-definite. Components are written in the SAMPLE FRAME. This is the primary representation of the stretch tensor in the code base

**latticeParameters**

Returns the lattice parameters consistent with stretch tensor

**minimizeFiberDistance** (*xtol=1e-12, ftol=1e-12*)

find orientation by minimizing distance to all fibers

**newGrain** (*newSpots, claimingSpots=False, lineage=None, phaseID=None, rMatTransf=None, vMat=None, omeTol=None, etaTol=None, \*\*kwargs*)

return a new grain instance without changing self; the new instance will use newSpots;

NOTE: claimingSpots is False by default, so if a grain is to be kept, may want to call claimSpots() method

phaseID and rMatTransf are useful for twins or phase transformations

**referenceLatticeParameters**

Return the reference lattice parameters stored on self

**setStretchTensor** (*vVec*)

Sets stretch tensor properly from a 6-vector in the Mandel-Voigt notation.

SEE ALSO: matrixutil.vecMVTtoSymm()

**set\_pVec** (*pVec*)

sets pVec properly

**strip** ()

meant for multiprocessing, to strip out things that do not really need to be pickled and sent

**uMat**

Returns the components of the right stretch tensor, which is symmetric positive-definite. Components are written in the CRYSTAL FRAME. The output is calculated as:  $U = R^T * V * R$  This is for convenience and cannot be set independently to preserve self-consistency.

**updateGVecs** (*rMat=None, bMat=None, chiTilt=None*)

special routine for updating the predicted G-vector angles for subsequent fitting \*) need to do this after updating chiTilt, or fixed bMat, etc... \*) assumption is that the changes are SMALL so that the existing list of

valid reflection is still valid...

**vMat**

Returns the components of the left stretch tensor, which is symmetric positive-definite. Components are written in the SAMPLE FRAME. This is the primary representation of the stretch tensor in the code base

**vol**

Returns the volume of the direct lattice consistent with the stretch tensor.

## 1.1.19 Module: xrd.hydra

Hydra detector tools

This is just a first pass at laying out a class for the hydra reader. Needs much more development.

### 1 Class

**class** hexrd.xrd.hydra.Hydra

Bases: `object`

Hydra image processing

**\_\_init\_\_** ()

Constructor for Hydra.

**loadImages()**

Load the four hydra images

## 1.1.20 Module: `xrd.indexer`

### 1 Class

**class** `hexrd.xrd.indexer.GrainSpotter`

Interface to grain spotter, which must be in the user's path

`__init__()`

### 9 Functions

`hexrd.xrd.indexer.convertUToRotMat(Urows, U0, symTag='Oh', display=False)`

Takes GrainSpotter gff output in rows

U11 U12 U13 U21 U22 U23 U13 U23 U33

and takes it into the hexrd/APS frame of reference

Urows comes from grainspotter's gff output U0 comes from `xrd.crystallography.latticeVectors.U0`

**convertRotMatToFableU(rMats, U0=array([[ 1., 0., 0.],  
[ 0., 1., 0.],  
[ 0., 0., 1.]]), symTag='Oh', display=False)**

Makes GrainSpotter gff output

U11 U12 U13 U21 U22 U23 U13 U23 U33

and takes it into the hexrd/APS frame of reference

Urows comes from grainspotter's gff output U0 comes from `xrd.crystallography.latticeVectors.U0`

`hexrd.xrd.indexer.testThisQ(thisQ)`

NOTES: (\*) doFit is not done here – in multiprocessing, that would end

up happening on a remote process and then different processes would have different data, unless spotsArray were made to be fancier

(\*) kludge stuff so that this function is outside of fiberSearch

**hexrd.xrd.indexer.fiberSearch(spotsArray, hklList, iPhase=0, nsteps=120, minCompleteness=0.6, minPctClaimed=0.95, preserveClaims=False, frie-  
lOnly=True, dspTol=None, etaTol=0.025, omeTol=0.025, eta-  
TolF=0.00225, omeTolF=0.00875, nStdDev=2, quitAfter=None,  
doRefinement=True, debug=True, doMultiProc=True, nC-  
PUs=None, outputGrainList=False)**

This indexer finds grains by performing 1-d searches along the fibers under the valid spots associated with each reflection order specified in `hklList`. The set of spots used to generate the candidate orientations may be restricted to Friedel pairs only.

`hklList` *must* have length > 0; Each `hkl` entry in `hklList` *must* be a tuple, not a list

the output is a concatenated list of orientation matrices ((n, 3, 3) `numpy.ndarray`).

`hexrd.xrd.indexer.pgRefine(x, etaOmeMaps, omegaRange, threshold)`

```
hexrd.xrd.indexer.paintGrid(quats,      etaOmeMaps,      threshold=None,      bMat=None,
                             omegaRange=None,      etaRange=None,      ome-
                             Tol=0.017453292519943295,      etaTol=0.017453292519943295,
                             omePeriod=(-3.141592653589793, 3.141592653589793), doMulti-
                             Proc=False, nCPUs=None, debug=False)
```

do a direct search of omega-eta maps to paint each orientation in quats with a completeness

bMat is in CRYSTAL frame

etaOmeMaps is instance of xrd.xrutil.CollapseOmeEta

omegaRange= $([-\text{num.pi}/3., \text{num.pi}/3.])$  for example

\*) lifted mainly from grain.py

\*) **self.etaGrid, self.omeGrid = num.meshgrid(self.etaEdges, self.omeEdges)** this means that ETA VARIES FASTEST!

...make a new function that gets called by grain to do the g-vec angle computation?

```
hexrd.xrd.indexer.paintGridThis(quat)
```

```
hexrd.xrd.indexer.writeGVE(spotsArray, fileroot, **kwargs)
```

write Fable gve file from Spots object

fileroot is the root string used to write the gve and ini files

Outputs:

No return value, but writes the following files:

<fileroot>.gve <fileroot>\_grainSpotter.ini (points to -> <fileroot>\_grainSpotter.log)

Keyword arguments:

Mainly for GrainSpotter .ini file, but some are needed for gve files

'sgNum': <225> 'phaseID': <None> 'cellString': <F> 'omeRange': <-60, 60, 120, 240> the oscillation range(s)

\*\*\*currently pulls from spots

'deltaOme': <0.25, 0.25> the oscillation delta(s) \*\*\*currently pulls from spots

'minMeas': <24> 'minCompl': <0.7> 'minUniqn': <0.5> 'uncertainty': <[0.10, 0.25, .50]> the min [tTh, eta, ome] uncertainties

in degrees

'eulStep': <2> 'nSigmas': <2> 'minFracG': <0.90> 'nTrials': <100000> 'positionfit': <True>

Notes:

\*) **The omeRange is currently pulled from the spotsArray input; the kwarg** has no effect as of now. Will change this to 'override' the spots info if the user, say, wants to pare down the range.

\*) **There is no etaRange argument yet, but presumably GrainSpotter knows** how to deal with this. Pending feature...

## 1.1.21 Module: xrd.material

Module for XRD material class

Use the Material class directly for new materials. Known materials are defined by name in materialDict.

## 1 Class

**class** `hexrd.xrd.material.Material` (*name='material', cfgP=None*)

Bases: `object`

Simple class for holding lattice parameters, accessible by name.

The class references materials by name and contains lattice and space group data.

**\_\_init\_\_** (*name='material', cfgP=None*)

Constructor for Material

*name* – (str) name of material  
*cfgP* – (instance) configuration file parser with  
– the material name as a section

**beamEnergy**

Beam energy in keV

**hklMax**

Max sum of squares for HKLs

**latticeParameters**

Lattice parameters

On output, all six parameters are returned.

On input, either all six or a minimal set is accepted.

The values have units attached, i.e. they are `valWunit` instances.

**name**

Name of material

**planeData**

(read only) Return the `planeData` attribute (lattice parameters)

**sgnum**

Space group number

**spaceGroup**

(read only) Space group

## 1 Function

`hexrd.xrd.material.loadMaterialList` (*cfgFile*)

Load a list of materials from a file

The file uses the config file format. See `ConfigParser` module.

### 1.1.22 Module: `xrd.rotations`

#### 20 Functions

`hexrd.xrd.rotations.arccosSafe` (*temp*)

Protect against numbers slightly larger than 1 in magnitude due to round-off

`hexrd.xrd.rotations.fixQuat` (*q*)

flip to positive *q*<sub>0</sub> and normalize

`hexrd.xrdat.rotations.invertQuat(q)`  
 silly little routine for inverting a quaternion

`hexrd.xrdat.rotations.misorientation(q1, q2, *args)`  
 sym is a tuple (crystal\_symmetry, \*sample\_symmetry) generally coded, may split up special cases for no symmetry or crystal/sample only...

`hexrd.xrdat.rotations.quatProduct(q1, q2)`  
 Product of two unit quaternions.

`qp = quatProduct(q2, q1)`

**q2, q1 are 4 x n, arrays whose columns are** quaternion parameters

**qp is 4 x n, an array whose columns are the** quaternion parameters of the product; the first component of qp is nonnegative

If  $R(q)$  is the rotation corresponding to the quaternion parameters  $q$ , then

$$R(qp) = R(q2) R(q1)$$

`hexrd.xrdat.rotations.quatProductMatrix(quats, mult='right')`  
 Form 4 x 4 arrays to perform the quaternion product

**USAGE** `qmats = quatProductMatrix(quats, mult='right')`

#### INPUTS

1. `quats` is (4, n), a numpy ndarray array of n quaternions horizontally concatenated
2. `mult` is a keyword arg, either 'left' or 'right', denoting the sense of the multiplication:

$$/ \text{quatProductMatrix}(h, \text{mult}='right') * q$$

$$q * h \rightarrow < \text{quatProductMatrix}(q, \text{mult}='left') * h$$

#### OUTPUTS

1. `qmats` is (n, 4, 4), the left or right quaternion product operator

#### NOTES

**\*) This function is intended to replace a cross-product based** routine for products of quaternions with large arrays of quaternions (e.g. applying symmetries to a large set of orientations).

`hexrd.xrdat.rotations.quatOfAngleAxis(angle, rotaxis)`  
 make an hstacked array of quaternions from arrays of angle/axis pairs

`hexrd.xrdat.rotations.quatOfExpMap(expMap)`

`hexrd.xrdat.rotations.quatOfRotMat(R)`

`hexrd.xrdat.rotations.rotMatOfExpMap_opt(expMap)`  
 Optimized version of `rotMatOfExpMap`

`hexrd.xrdat.rotations.rotMatOfExpMap_orig(expMap)`  
 Original `rotMatOfExpMap`, used for comparison to optimized version

`hexrd.xrdat.rotations.rotMatOfQuat(quat)`  
 Take an array of n quats (numpy ndarray, 4 x n) and generate an array of rotation matrices (n x 3 x 3)

Uses the truncated series expansion for the exponential map; divide-by-zero is checked using the global 'tiny-RotAng'

`hexrd.xrdat.rotations.angleAxisOfRotMat(R)`

```
hexrd.xrd.rotations.distanceToFiber(c, s, q, qsym, **kwargs)

discreteFiber(c, s, B=array([[ 1.,  0.,  0.],
[ 0.,  1.,  0.],
[ 0.,  0.,  1.]]), ndiv=120, invert=False, csym=None, ssym=None)

hexrd.xrd.rotations.mapAngle(ang, *args, **kwargs)
    Utility routine to map an angle into a specified period

hexrd.xrd.rotations.angularDifference_orig(angList0, angList1, units='radians')
    Do the proper (acute) angular difference in the context of a branch cut.

    *) Default angular range in the code is [-pi, pi] *) ... maybe more efficient not to vectorize?

hexrd.xrd.rotations.angularDifference_opt(angList0, angList1, units='radians')
    Do the proper (acute) angular difference in the context of a branch cut.

    *) Default angular range in the code is [-pi, pi]

hexrd.xrd.rotations.printTestName(num, name)

hexrd.xrd.rotations.testRotMatOfExpMap(numpts)
    Test rotation matrix from axial vector
```

### 1.1.23 Module: xrd.spacegroup

Interface with sglite for hkl generation and Laue group determination

This module contains mappings from space group number to either Hall or Hermann-Mauguin notation, as well as the inverse notations.

Space groups can be mapped to crystal class (one of 32 point groups) and then to crystal system .

NOTES:

- Laue group is the crystal class if you add a center of symmetry. There are 11 Laue groups, determined directly from the point group.
- This module avoids the use of numpy and uses math module instead. That means the hkl lists are not numpy arrays, but simple lists of tuples.
- Rhombohedral lattices:

#### REFERENCES

1. Mappings among space group number, Hall symbols and Hermann-Mauguin symbols.  
[http://cci.lbl.gov/sginfo/hall\\_symbols.html](http://cci.lbl.gov/sginfo/hall_symbols.html)
2. For mapping space group number to point group (crystal class in Schonflies notation)  
[http://en.wikipedia.org/wiki/Space\\_group](http://en.wikipedia.org/wiki/Space_group)
3. Crystallography and crystal defects By Anthony Kelly, G. W. Groves, P. Kidd
4. Contains point group from sgnum. [http://en.wikipedia.org/wiki/Space\\_group#Classification\\_systems\\_for\\_space\\_groups](http://en.wikipedia.org/wiki/Space_group#Classification_systems_for_space_groups)
5. Point group to laue group <http://www.ruppweb.org/Xray/tutorial/32point.htm>
6. For discussion of rhombohedral lattice and “obverse” and “reverse” settings for lattice parameters. Crystal structure determination (book) By Werner Massa

#### TESTING

Run this module as main to generate all space groups and test the HKL evaluation.



## 1 Class

**class** `hexrd.xrd.spacegroup.SpaceGroup` (*sgnum*)

Bases: `object`

Wrapper on `sglite`

**\_\_init\_\_** (*sgnum*)

Constructor for `SpaceGroup`

INPUTS *sgnum* – (int) space group number (between 1 and 230)

**HallSymbol**

(read only) Hall symbol

**SgOps**

(read only) An `sglite.SgOps` instance

**getHKLS** (*ssmax*)

Return a list of HKLs with a cutoff sum of square

INPUTS *ssmax* – cutoff sum of squares

OUTPUTS *hkls* – a list of all HKLs with sum of squares less than  
or equal to the cutoff, excluding systematic absences and symmetrically equivalent *hkls*

DESCRIPTION

**hermannMauguin**

(read only) Hermann-Mauguin symbol

**latticeType**

Lattice type

Possible values are ‘cubic’, ‘hexagonal’, ‘trigonal’, ‘tetragonal’, ‘orthorhombic’, ‘monoclinic’ and ‘triclinic’

Rhombohedral lattices are treated as trigonal using the “obverse” setting.

**laueGroup**

Schonflies symbol for Laue group (read only)

**pointGroup**

Schonflies symbol for point group (read only)

**reqParams**

(read only) Zero-based indices of required lattice parameters

**sgnum**

Space group number

**sixLatticeParams** (*lparams*)

Return the complete set of six lattice parameters from the abbreviated set

INPUTS *lparams* – (tuple) the abbreviated set of lattice parameters

OUTPUTS *sparams* – (tuple) the complete set of lattice parameters;

(a, b, c, alpha, beta, gamma)

DESCRIPTION \* Output angles are in degrees

## 1 Function

`hexrd.xrdspacegroup.testHKLs()`

### 1.1.24 Module: `xrd.spotfinder`

## 12 Classes

**class** `hexrd.xrdspacegroup.spotfinder.IntensityFunc3D(*args, **kwargs)`

Bases: `object`

This is just a template for intensity distribution functions in 3D

`__init__(*args, **kwargs)`

**eval** (*xVec*, *x*, *y*, *z*, *w=None*, *vSub=None*, *vScale=None*, *diff=False*, *noBkg=False*)

if *w* is `None`: *x*, *y*, and *z* are 1D; if *w* has 1D weights: *x*, *y*, and *z* are 2D

if *vSub* is present, it is subtracted from results – useful for forming least-squares residuals; and *vScale* is used to scale the results if it is present

**class** `hexrd.xrdspacegroup.spotfinder.IntensityFuncGauss3D`

Bases: `hexrd.xrdspacegroup.spotfinder.IntensityFunc3D`

**8 parameters:** centers (3) FWHMs (3) scaling (1) background (1)

`__init__()`

**class** `hexrd.xrdspacegroup.spotfinder.IntensityFuncGauss3DGenEll`

Bases: `hexrd.xrdspacegroup.spotfinder.IntensityFunc3D`

generalization of `IntensityFuncGauss3D` to have principal axes generally aligned 11 parameters:

centers (3) diagonal “fwhm” (3) scaling (1) off-diagonal (3) background (1)

`__init__()`

**classmethod** `get2DFunc()`

if drop to 2D, also drop back to aligned ellipsoid

**guessXVec** (*x*, *y*, *z*, *w=None*, *v=None*, *noBkg=False*)

guess is for aligned ellipsoid

**class** `hexrd.xrdspacegroup.spotfinder.IntensityFuncMulti3D(subFuncClass, nSubFuncs, min-Width=None)`

Bases: `hexrd.xrdspacegroup.spotfinder.IntensityFunc3D`

combination of multiple overlapped functions

`__init__(subFuncClass, nSubFuncs, minWidth=None)`

**guessXVecPureNDImage** (*x*, *y*, *z*, *pxlCenterList*, *w=None*, *v=None*, *pxl=None*, *gaussFilter-Sigma=None*, *noBkg=False*)

for now, rely on specified centers; assumes that *v* can be made into integers without too much loss of precision

**class** `hexrd.xrdspacegroup.spotfinder.IntensityFunc2D(*args, **kwargs)`

Bases: `object`

This is just a template for intensity distribution functions in 2D; It could be unified with the 3D version, but that would potentially make the interface harder to understand

`__init__(*args, **kwargs)`

**eval** (*xVec*, *x*, *y*, *w=None*, *vSub=None*, *vScale=None*, *diff=False*, *noBkg=False*)

if *w* is None: *x* and *y* are 1D; if *w* has 1D weights: *x* and *y* are 2D

if *vSub* is present, it is subtracted from results – useful for forming least-squares residuals; and *vScale* is used to scale the results if it is present

**class** `hexrd.xrd.spotfinder.IntensityFuncGauss2D`

Bases: `hexrd.xrd.spotfinder.IntensityFunc2D`

**6 parameters:** centers (2) FWHMs (2) scaling (1) background (1)

`__init__` ()

**class** `hexrd.xrd.spotfinder.IntensityFuncMulti2D` (*subFuncClass*, *nSubFuncs*)

Bases: `hexrd.xrd.spotfinder.IntensityFunc2D`

combination of multiple overlapped functions

`__init__` (*subFuncClass*, *nSubFuncs*)

**guessXVec** (*x*, *y*, *pxlCenterList*, *w=None*, *v=None*, *pxl=None*, *gaussFilterSigma=None*, *noBkg=False*, *minWidth=None*, *wtrshd=None*)

for now, rely on specified centers; assumes that *v* can be made into integers without too much loss of precision

**class** `hexrd.xrd.spotfinder.UnfitableError` (*err*, *msg*)

Bases: `exceptions.Exception`

`__init__` (*err*, *msg*)

**class** `hexrd.xrd.spotfinder.FitFailedError` (*err*, *msg*)

Bases: `exceptions.Exception`

`__init__` (*err*, *msg*)

**class** `hexrd.xrd.spotfinder.Spot` (*key*, *delta\_omega*, *data=None*, *omega=None*, *iFrame=None*, *detectorGeom=None*)

Bases: `object`

`__init__` (*key*, *delta\_omega*, *data=None*, *omega=None*, *iFrame=None*, *detectorGeom=None*)

key should be unique among all spots in the collection; can call with initial data or not

if have tuple from `getDataMinimal` call, should be able to init directly with that tuple

**angCOM** (*useFit=True*, *detectorGeom=None*, *getUncertainties=False*, *iSubSpot=None*)

Get center-of-mass in twotheta, eta, omega coordinates. If *useFit*, then return value from function fit instead of simple estimate

Could look at *detectorGeom* to see if it has a *pVec* and do something more elaborate if *detectorGeom* is already set without a *pVec*, but that is probably asking for trouble

**append** (*dataDict*, *omega*, *iFrame*)

*dataDict* should be like one of the entries in *spotDataList* coming back from `spotFinderSingle`

**cleanFit** ()

clean up data associated with having done fit()

**static cullSpots** (*spots*, *tests*)

Apply a list of tests to spots and return a list with spots that fail culled

*spots* is a list of `Spot` instances; *tests* is a list of tests to be applied;

For convenience, some tests have been defined as static methods off of the `Spot` class. If a test is tests has a length, then the non-first entries are used as arguments.

**display** (*cmap=None, relfigsize=(1, 1), vAll=None, markersize=2, xyoPointsList=[([], {})], \*\*kwargs*)  
vAll, if present, is used instead of self.vAll; useful for results from fitting

**displayFlat** (*vAll=None, cmap=None, markersize=2, \*\*kwargs*)  
Flatten in Omega for display, no loss of x-y resolution

**displayFrames** (*reader, nFramesPad=0, sumImg=<ufunc 'maximum'>, \*\*kwargs*)  
display the frame(s) from which a spot came

**doMap**  
centralize this decision so that it does not change with the use of quadrature and the like

**exceptionOnFit** ()  
return the exception if stored on fit; note that not all exceptions get stored

**finalize** (*flatten=False, modifyDeltaOmega=False, cullDupl=True*)  
Could potentially get rid of self.data once finalize is called, but leave it around just in case it is useful for subsequent operations. It might, for example, be needed if we go back and read more intensity data in the vicinity of the spot.

**fit** (*funcType=None, quadr='auto', full\_output=False, uncertainties=False, confidence\_level=0.95, debug=None, detectorGeom=None, fout=<open file '<stdout>', mode 'w' at 0x7fad91ab0150>*)  
fit a distribution to a spot;

may throw a UnfitableError Exception if it is not possible to fit the spot with the given function

if want to re-fit a spot, call the cleanFit method first

if just want to make sure an attempt was made to fit the spot, call fitWrap

**fitWrap** (*\*args, \*\*kwargs*)  
useful if just want to make sure an attempt was made to fit the spot; only lets exceptions through if the spot is suspect

**flattenOmega** (*modifyDeltaOmega=False*)  
flatten spot in Omega;

unless modifyDeltaOmega is set, do not change delta\_omega, under the assumption that the spot is being flattened if it does not significantly spill across multiple omega frames

**getDoMap** ()  
centralize this decision so that it does not change with the use of quadrature and the like

**getFrames** (*reader=None*)  
if have self.data, do not need reader

**isMarked** (*marks*)  
return true of spot has been marked as indicated; marks can be an integer or list; if marks is a list, returns true if any of marks are true

**static loadSpots** (*f, minimal=True, closeLocal=True*)  
load spots stored with storeSpots

note that if a detectorGeom was stored by storeSpots, then all spots loaded here will have it attached to them

**merge** (*other*)  
merge in other's data

**setupMulti** (*xyoList, wtrshd=None*)  
for now, rely on candidate centers having been provided; probably want to do a fit

**setupGuardFromFWHM** (*fwhm, fout=<open file '<stdout>', mode 'w' at 0x7fad91ab0150>*)  
so far, written only for withOmega being True

**static spotFromDataList** (*key, delta\_omega, dataList*)  
 useful for merging data from spots that have already been finalized

**static storeSpots** (*f, spotList, closeLocal=True*)  
 store spots to f, which can be a filename or something which behaves like a shelve instance  
 stores a somewhat minimal set of data  
 all spots must have the same detectorGeom  
 consider changing to something like: `import cPickle as pickle; s = open(f, 'w'); pickle.dump(stuff, f); f.close()`

**xyoCOM** (*useFit=True, iSubSpot=None*)  
 Get center-of-mass in x, y, omega coordinates on the image stack. If useFit, then return value from function fit instead of simple estimate

**class** hexrd.xrd.spotfinder.**Spots** (*planeData, data, detectorGeom, omegaMM, \*\*kwargs*)  
 Bases: `object`

Meant for holding spot data in a form useful for indexing

**\_\_init\_\_** (*planeData, data, detectorGeom, omegaMM, \*\*kwargs*)  
 planeData : an instance or list (for multiple phases) of the PlaneData class  
 data : can any of:  
     spots : list of Spot instances, all of which must have been finalized  
     spotAngCoords : spot positions in angular coordinates  
     None  
 detectorGeom : an instance of DetectorGeomGE or the like  
 omegaMM : (min, max) of omega range, or list of such min/max tuples

**checkClaim** (*index*)  
 careful: unlike claimSpots, does not check grain association

**checkClaims** (*indices=None*)  
 careful: unlike claimSpots, does not check grain association; careful: indices should not be a boolean array

**claimSpots** (*indices, newClaimedBy, checkOnly=False, asMaster=False*)  
 careful: indices should not be a boolean array;  
 returns bool of same length as indices, with True entries for indices that are in conflict; if checkOnly, then do not actually claim the spots

**cleanMulti** ()  
 clean up spots for which split fitting was done, as they might need to change to single spots

**static ensslaveSpotData** (*d, dMaster*)  
 ends up, after potentially doing recursive calls, slaving master of d to master of dMaster

**static findSpotsOmegaStack** (*reader, nFrames, threshold, minPx, discardAtBounds=True, keepWithinBBox=True, overlapPixelDistance=None, nframesLump=1, padOmega=True, padSpot=True, debug=False, pw=None, fout=None, sumImg=None*)

... if merge at bounds, would need to do that before culling based on size (\*) spots from beginning would have already been finalized – can merge with them without trouble? (\*) add comments here about what happens when reader.wrapsAround

This method does not necessarily need to hang off of the Spots class, but Spots is a convenient place to put it.

If nFrames == 0, read all frames.

if pass overlapPixelDistance, then overlap is determined based on centroid distances in pixel units; with overlapPixelDistance being set to the tolerance for overlap; probably most useful when omega steps are large

**reader has been created by doing something like:** fileInfo = [('RUBY\_4537.raw', 2), ('RUBY\_4538.raw', 2)] reader = detector.ReadGE(fileInfo, subtractDark=True)

if go to parallel processing, perhaps return first and last lables for doing merges

**fitHasFailed** (*index, subSpotOnly=False*)  
if subSpotOnly the only return true if it was a subspot that failed to fit

**fitSpots** (*indices, \*args, \*\*kwargs*)  
fit spots with given indices and update spotAngCoords; for indices is None, all spots are considered for fitting; indices may be a single integer instead of a list of spot indices  
set claimsBased if want to base method on spots that have been claimed

**fitSpotsMulti** (*indices, threshold, \*args, \*\*kwargs*)  
like fitSpots, but consider splitting up spots with multiple peaks

**getAngCoords** (*indices=None*)  
... make returned thing unwritable even if it is a slice

**getHKLSpots** (*hkl, phaseID=None, unclaimedOnly=False, disallowMasterWhenSplit=True*)  
get boolean array that is True for spots associated with a given hkl (and optionally phase)

**getIntegratedIntensity** (*index, \*\*kwargs*)  
wrapper that takes care of subspot details

**getIterHKL** (*hkl, phaseID=None, unclaimedOnly=True, friedelOnly=False, iSpotLo=0, returnBothCoordTypes=False*)  
Returns an iterator over a given set of spots

**getIterPhase** (*phaseID=None, unclaimedOnly=True, friedelOnly=False, iSpotLo=0, returnBothCoordTypes=False*)  
Returns an iterator over a given set of spots

**getPixelIsInSpots** (*indices, xyo, pixelDist=0*)  
find spots that are within pixelDist of containing a given pixel; indices may be a single integer, a list of spot indices, or a boolean array;  
may want to use pixelDist of 1 or 2 to protect against dead pixels

**getXYOCords** (*indices*)  
... make returned thing unwritable even if it is a slice

**static mergeSpotsOmegaStack** (*spotsA, spotsB, readerA, readerB, keepWithinBBox=True, overlapPixelDistance=None, padOmega=True, logger=None*)  
see findSpotsOmegaStack (documentation in header and in the body) for comments about padOmega and overlapPixelDistance  
spotsB and readerB can be None if this is for wrap-around  
A and B can be the same – as would be the case for omega wrapping all the way around

**resetDetectorGeom** (*detectorGeom, doFits=False, fitArgs=[], fitKwargs={}*)  
update detector geometry; also fits all spots; probably only want to call this for a single-grain data set or the like; updates angular and cartesian coordinates, but not 2-theta associations or other meta-data

**class hexrd.xrd.spotfinder.SpotsIterator** (*spots, hkl, phaseID, unclaimedOnly, friedelOnly, iSpotLo, returnBothCoordTypes*)  
iterator over a given set of spots in a Spots instance, note that the iterator ignores relations among split spots

**\_\_init\_\_** (*spots, hkl, phaseID, unclaimedOnly, friedelOnly, iSpotLo, returnBothCoordTypes*)

## 18 Functions

```

hexrd.xrd.spotfinder.getBin (thisframe, threshold, padSpot)
hexrd.xrd.spotfinder.dilateObj (obj, shape, nDilate=1)
hexrd.xrd.spotfinder.emptyBox (obj, dtype)
hexrd.xrd.spotfinder.getDtype (this)
hexrd.xrd.spotfinder.copyBox (inpt, obj)
    deepcopy does not seem to do what we want with masked arrays
hexrd.xrd.spotfinder.getSpot (inpt, labels, objs, index, keepWithinBBox, padSpot, dark-
    frame=None)
    labels is from ndimage.label; objs is from ndimage.find_objects
hexrd.xrd.spotfinder.getSpotFromPixels (inpt, xThese, yThese, index, keepWithinBBox,
    padSpot, darkframe=None)
    this is a bit of an oddball function, mostly for data uniformity of data structures;
hexrd.xrd.spotfinder.cullSpotUsingBin (spot, bin)
    remove data for pixels where bin is true; vbox does not change, just change the locations that are marked as
    belonging with the spot
hexrd.xrd.spotfinder.getValuesOnly (inpt, labels, objs, index)
    labels is from ndimage.label; objs is from ndimage.find_objects
hexrd.xrd.spotfinder.getIndices (inpt, labels, obj, index, mode='default', minlabel=0)
hexrd.xrd.spotfinder.getImCOM (inpt, labels, objs, index, floor=None, getVSum=False)
    labels is from ndimage.label; objs is from ndimage.find_objects
    set floor for a minimum intensity to use in intensity weighted COM
    return sum of intensity as well if getVSum is True
hexrd.xrd.spotfinder.getObjSize (labels, objs, index)
    labels is from ndimage.label; objs is from ndimage.find_objects
hexrd.xrd.spotfinder.spotFinderSingle (thisframe, threshold, minPx, keepWithinBBox,
    padSpot, weightedCOM=True, pw=None, de-
    bug=False, darkframe=None)
    find spots in thisframe; threshold can be a scalar or defined over of same dimension as thisframe; minPx is
    the minimum number of pixels to be kept as a spot; weightedCOM is true to use thisframe data for weighting
    center-of-mass position; if pw is present, it is used for plotting (assumed to be plotwrap.PlotWrap instance)
hexrd.xrd.spotfinder.getWtrShd (inp, threshold, gaussFilterSigma=None, footPrint=None, fpi=5,
    numPeaks=None)
    fpi only used if footPrint is None
hexrd.xrd.spotfinder.doSpotThis (iSpot)
    meant for use with multiprocessing
hexrd.xrd.spotfinder.testSingle (fileInfo)
hexrd.xrd.spotfinder.testSpotFinder (fileInfo, delta_omega, omega_low, howMuch=0.1)
hexrd.xrd.spotfinder.main (argv=[])

```

### 1.1.25 Module: `xrd.symmetry`

#### 4 Functions

`hexrd.xrd.symmetry.toFundamentalRegion` (*q*, *crysSym*='Oh', *sampSym*=None)

`hexrd.xrd.symmetry.ltypeOfLaueGroup` (*tag*)  
See `quatOfLaueGroup`

`hexrd.xrd.symmetry.quatOfLaueGroup` (*tag*)  
Generate quaternion representation for the specified Laue group.

#### USAGE:

`qsym = quatOfLaueGroup(schoenfliesTag)`

#### INPUTS:

1) `schoenfliesTag` 1 x 1, a case-insensitive string representing the Schoenflies symbol for the desired Laue group. The 14 available choices are:

Class Symbol n

Triclinic Ci (S2) 1 Monoclinic C2h 2 Orthorhombic D2h (Vh) 4 Tetragonal C4h 4

D4h 8

**Trigonal C3i (S6) 3** D3d 6

**Hexagonal C6h 6** D6h 12

**Cubic Th 12** Oh 24

#### OUTPUTS:

1) `qsym` is (4, n) the quaterions associated with each element of the chosen symmetry group having n elements (dep. on group – see INPUTS list above).

#### NOTES:

**\*** The conventions used for assigning a RHON basis, {x1, x2, x3}, to each point group are consistent with those published in Appendix B of [1].

#### REFERENCES:

[1] Nye, J. F., “Physical Properties of Crystals: Their Representation by Tensors and Matrices”, Oxford University Press, 1985. ISBN 0198511655

`hexrd.xrd.symmetry.applySym` (*vec*, *qsym*, *csFlag*=False, *cullPM*=False, *tol*=1e-08)  
apply symmetry group to a single 3-vector (columnar) argument  
*csFlag* : centrosymmetry flag *cullPM* : cull +/- flag

### 1.1.26 Module: `xrd.transforms`

#### 22 Functions

`hexrd.xrd.transforms.makeGVector` (*hkl*, *bMat*)  
take a CRYSTAL RELATIVE B matrix onto a list of hkls to output unit reciprocal lattice vectors (a.k.a. lattice plane normals)

Required Arguments: `hkls` – (3, n) ndarray of n hstacked reciprocal lattice vector component



triplets

**bMat** – (3, 3) ndarray representing the matrix taking reciprocal lattice vectors to the crystal reference frame

Output: gVecs – (3, n) ndarray of n unit reciprocal lattice vectors

(a.k.a. lattice plane normals)

To Do: \* might benefit from some assert statements to catch improperly shaped input.

```
hexrd.xrtd.transforms.anglesToGVec(angs, bHat_l, eHat_l, rMat_s=None, rMat_c=None)
    from 'eta' frame out to lab (with handy kwargs to go to crystal or sample)
```

```
gvecToDetectorXY(gVec_c, rMat_d, rMat_s, rMat_c, tVec_d, tVec_s, tVec_c, beamVec=array([[ -0. ],
[ -1. ]]))
```

Takes a list of unit reciprocal lattice vectors in crystal frame to the specified detector-relative frame, subject to the conditions:

- 1.the reciprocal lattice vector must be able to satisfy a bragg condition
- 2.the associated diffracted beam must intersect the detector plane

Required Arguments: gVec\_c – (3, n) ndarray of n reciprocal lattice vectors in the CRYSTAL FRAME rMat\_d – (3, 3) ndarray, the COB taking DETECTOR FRAME components to LAB FRAME rMat\_s – (3, 3) ndarray, the COB taking SAMPLE FRAME components to LAB FRAME rMat\_c – (3, 3) ndarray, the COB taking CRYSTAL FRAME components to SAMPLE FRAME tVec\_d – (3, 1) ndarray, the translation vector connecting LAB to DETECTOR tVec\_s – (3, 1) ndarray, the translation vector connecting LAB to SAMPLE tVec\_c – (3, 1) ndarray, the translation vector connecting SAMPLE to CRYSTAL

Outputs: (3, m) ndarray containing the intersections of m <= n diffracted beams associated with gVecs

```
detectorXYToGvec(xy_det, rMat_d, rMat_s, tVec_d, tVec_s, tVec_c, distortion=(<function GE_4
[ -0. ],
[ -1. ]]), etaVec=array([[ 1. ],
[ 0. ],
[ 0. ]]))
```

Takes a list cartesian (x, y) pairs in the detector coordinates and calculates the associated reciprocal lattice (G) vectors and (bragg angle, azimuth) pairs with respect to the specified beam and azimuth (eta) reference directions

Required Arguments: xy\_det – (n, 2) ndarray or list-like input of n detector (x, y) points rMat\_d – (3, 3) ndarray, the COB taking DETECTOR FRAME components to LAB FRAME rMat\_s – (3, 3) ndarray, the COB taking SAMPLE FRAME components to LAB FRAME tVec\_d – (3, 1) ndarray, the translation vector connecting LAB to DETECTOR tVec\_s – (3, 1) ndarray, the translation vector connecting LAB to SAMPLE tVec\_c – (3, 1) ndarray, the translation vector connecting SAMPLE to CRYSTAL

Optional Keyword Arguments: beamVec – (3, 1) ndarray containing the incident beam direction components in the LAB FRAME etaVec – (3, 1) ndarray containing the reference azimuth direction components in the LAB FRAME

Outputs: (n, 2) ndarray containing the (tTh, eta) pairs associated with each (x, y) (3, n) ndarray containing the associated G vector directions in the LAB FRAME associated with gVecs

```
oscillAnglesOfHKLs(hkls, chi, rMat_c, bMat, wavelength, vInv=array([[ 1. ],
[ 1. ],
[ 1. ],
[ 0. ],
[ 0. ]],
```

```
[ 0.]], beamVec=array([[ -0.],  
[-0.],  
[-1.]]), etaVec=array([[ 1.],  
[ 0.],  
[ 0.]])
```

Takes a list of unit reciprocal lattice vectors in crystal frame to the specified detector-relative frame, subject to the conditions:

- 1.the reciprocal lattice vector must be able to satisfy a bragg condition
- 2.the associated diffracted beam must intersect the detector plane

Required Arguments: hkl – (3, n) ndarray of n reciprocal lattice vectors in the CRYSTAL FRAME chi – float representing the inclination angle of the oscillation axis (std coords) rMat\_c – (3, 3) ndarray, the COB taking CRYSTAL FRAME components to SAMPLE FRAME bMat – (3, 3) ndarray, the COB taking RECIPROCAL LATTICE components to CRYSTAL FRAME wavelength – float representing the x-ray wavelength in Angstroms

Optional Keyword Arguments: beamVec – (3, 1) ndarray containing the incident beam direction components in the LAB FRAME etaVec – (3, 1) ndarray containing the reference azimuth direction components in the LAB FRAME vInv – (6, 1) ndarray containing the indep. components of the inverse left stretch tensor

in the SAMPLE FRAME in the Mandel-Voigt notation

Outputs: ome0 – (3, n) ndarray containing the feasible (tTh, eta, ome) triplets for each input hkl (first solution) ome1 – (3, n) ndarray containing the feasible (tTh, eta, ome) triplets for each input hkl (second solution)

The reciprocal lattice vector, G, will satisfy the the Bragg condition when:

$$\mathbf{b} \cdot \mathbf{T} * \mathbf{G} / \|\mathbf{G}\| = -\sin(\theta)$$

where b is the incident beam direction ( $k_i$ ) and theta is the Bragg angle consistent with G and the specified wavelength. The components of G in the lab frame in this case are obtained using the crystal orientation, Rc, and the single-parameter oscillation matrix, Rs(ome):

$$\mathbf{R}(\text{ome}) * \mathbf{R}_c * \mathbf{G} / \|\mathbf{G}\|$$

The equation above can be rearranged to yeild an expression of the form:

$$a * \sin(\text{ome}) + b * \cos(\text{ome}) = c$$

which is solved using the relation:

$$a * \sin(x) + b * \cos(x) = \sqrt{a^2 + b^2} * \sin(x + \alpha)$$

$$\longrightarrow \sin(x + \alpha) = c / \sqrt{a^2 + b^2}$$

where:

$$\alpha = \arctan2(b, a)$$

The solutions are:

$$/ | \arcsin(c / \sqrt{a^2 + b^2}) - \alpha$$

$$\mathbf{x} = <$$

$$\pi - \arcsin(c / \sqrt{a^2 + b^2}) - \alpha$$

There is a double root in the case the reflection is tangent to the Debye-Scherrer cone ( $c^2 = a^2 + b^2$ ), and no solution if the Laue condition cannot be satisfied (filled with NaNs in the results array here)

```
polarRebin(thisFrame, npdiv=2, mmPerPixel=(0.2, 0.2), convertToTTh=False, rMat_d=array([[
```

```
[ 0., 1., 0.],
[ 0., 0., 1.]], tVec_d=array([ 0., 0., -1000.]), beamVec=array([[ -0.],
[ -0.],
[ -1.]]), etaVec=array([[ 1.],
[ 0.],
[ 0.]]), rhoRange=array([ 20, 200]), numRho=1000, etaRange=array([-0.08726646, 6.19591884])
```

Caking algorithm

#### INPUTS

thisFrame npdiv=2, pixel subdivision (n x n) to determine bin membership rhoRange=[100, 1000] - radial range in pixels numRho=1200 - number of radial bins etaRange=np.pi\*np.r\_[-5, 355]/180. - range of eta numEta=36 - number of eta subdivisions ROI=None - region of interest (four vector) corrected=False - uses 2-theta instead of rho verbose=True,

hexrd.xrd.transforms.**arccosSafe** (*temp*)

Protect against numbers slightly larger than 1 in magnitude due to round-off

hexrd.xrd.transforms.**angularDifference** (*angList0, angList1, units='radians'*)

Do the proper (acute) angular difference in the context of a branch cut.

*\*)* Default angular range is [-pi, pi]

hexrd.xrd.transforms.**mapAngle** (*ang, \*args, \*\*kwargs*)

Utility routine to map an angle into a specified period

hexrd.xrd.transforms.**reg\_grid\_indices** (*edges, points\_1d*)

get indices in a 1-d regular grid.

edges are just that:

**point: x (2.5)**

**edges: | 1 | 2 | 3 | 4 | 5**

**indices: | 0 | 1 | 2 | 3 |**

above the deltas are + and the index for the point is 1

**point: x (2.5)**

**edges: | 5 | 4 | 3 | 2 | 1**

**indices: | 0 | 1 | 2 | 3 |**

here the deltas are - and the index for the point is 2

- can handle grids with +/- deltas

- be careful when using with a cyclical angular array! edges and points must be mapped to the same branch cut, and  $\text{abs}(\text{edges}[0] - \text{edges}[-1]) = 2\pi$

hexrd.xrd.transforms.**columnNorm** (*a*)

normalize array of column vectors (hstacked, axis = 0)

hexrd.xrd.transforms.**rowNorm** (*a*)

normalize array of row vectors (vstacked, axis = 1)

hexrd.xrd.transforms.**unitVector** (*a*)

normalize array of column vectors (hstacked, axis = 0)

`hexrd.xrd.transforms.makeDetectorRotMat (tiltAngles)`

Form the (3, 3) tilt rotations from the tilt angle list:

`tiltAngles = [gamma_Xl, gamma_Yl, gamma_Zl]` in radians

`hexrd.xrd.transforms.makeOscillRotMat (oscillAngles)`

`oscillAngles = [chi, ome]`

`hexrd.xrd.transforms.makeRotMatOfExpMap (expMap)`

`hexrd.xrd.transforms.makeBinaryRotMat (axis)`

`hexrd.xrd.transforms.makeEtaFrameRotMat (bHat_l, eHat_l)`

make eta basis COB matrix with beam antiparallel with Z

takes components from ETA frame to LAB

`hexrd.xrd.transforms.validateAngleRanges (angList, startAngs, stopAngs, ccw=True)`

A better way to go. find out if an angle is in the range CCW or CW from start to stop

There is, of course, an ambiguity if the start and stop angle are the same; we treat them as implying  $2\pi$  having been mapped

`hexrd.xrd.transforms.rotate_vecs_about_axis (angle, axis, vecs)`

Rotate vectors about an axis

INPUTS *angle* - array of angles (len == 1 or n) *axis* - array of unit vectors (shape == (3, 1) or (3, n)) *vec* - array of vectors to be rotated (shape = (3, n))

Quaternion formula: if we split *v* into parallel and perpendicular components w.r.t. the axis of quaternion *q*,

$$v = a + n$$

then the action of rotating the vector  $\text{dot}(R(q), v)$  becomes

$$v_{\text{rot}} = (q_0^2 - |q|^2)(a + n) + 2 \cdot \text{dot}(q, a) \cdot q + 2 \cdot q_0 \cdot \text{cross}(q, n)$$

`hexrd.xrd.transforms.quat_product_matrix (q, mult='right')`

Form 4 x 4 array to perform the quaternion product

**USAGE** `qmat = quatProductMatrix(q, mult='right')`

#### INPUTS

1. `quats` is (4,), an iterable representing a unit quaternion horizontally concatenated
2. `mult` is a keyword arg, either 'left' or 'right', denoting the sense of the multiplication:

$$\text{quatProductMatrix}(h, \text{mult}='right') * q$$

$$q * h \rightarrow \text{quatProductMatrix}(q, \text{mult}='left') * h$$

#### OUTPUTS

1. `qmat` is (4, 4), the left or right quaternion product operator

#### NOTES

- \*) This function is intended to replace a cross-product based** routine for products of quaternions with large arrays of quaternions (e.g. applying symmetries to a large set of orientations).

`hexrd.xrd.transforms.quat_distance (q1, q2, qsym)`

## 1.1.27 Module: `xrd.transforms_CAPI`

### 19 Functions

`hexrd.xrd.transforms_CAPI.makeGVector(hkl, bMat)`

take a CRYSTAL RELATIVE B matrix onto a list of hkl's to output unit reciprocal lattice vectors (a.k.a. lattice plane normals)

Required Arguments: `hkl's` – (3, n) ndarray of n hstacked reciprocal lattice vector component triplets

`bMat` – (3, 3) ndarray representing the matrix taking reciprocal lattice vectors to the crystal reference frame

Output: `gVecs` – (3, n) ndarray of n unit reciprocal lattice vectors (a.k.a. lattice plane normals)

To Do: \* might benefit from some assert statements to catch improperly shaped input.

`gvecToDetectorXY(gVec_c, rMat_d, rMat_s, rMat_c, tVec_d, tVec_s, tVec_c, beamVec=array([[ -0. ], [ -1. ]]))`

Takes a list of unit reciprocal lattice vectors in crystal frame to the specified detector-relative frame, subject to the conditions:

- 1.the reciprocal lattice vector must be able to satisfy a bragg condition
- 2.the associated diffracted beam must intersect the detector plane

Required Arguments: `gVec_c` – (n, 3) ndarray of n reciprocal lattice vectors in the CRYSTAL FRAME `rMat_d` – (3, 3) ndarray, the COB taking DETECTOR FRAME components to LAB FRAME `rMat_s` – (3, 3) ndarray, the COB taking SAMPLE FRAME components to LAB FRAME `rMat_c` – (3, 3) ndarray, the COB taking CRYSTAL FRAME components to SAMPLE FRAME `tVec_d` – (3, 1) ndarray, the translation vector connecting LAB to DETECTOR `tVec_s` – (3, 1) ndarray, the translation vector connecting LAB to SAMPLE `tVec_c` – (3, 1) ndarray, the translation vector connecting SAMPLE to CRYSTAL

Outputs: (m, 2) ndarray containing the intersections of  $m \leq n$  diffracted beams associated with `gVecs`

`detectorXYToGvec(xy_det, rMat_d, rMat_s, tVec_d, tVec_s, tVec_c, beamVec=array([[ -0. ], [ -1. ]]), etaVec=array([[ 1. ], [ 0. ], [ 0. ]]))`

Takes a list cartesian (x, y) pairs in the detector coordinates and calculates the associated reciprocal lattice (G) vectors and (bragg angle, azimuth) pairs with respect to the specified beam and azimuth (eta) reference directions

Required Arguments: `xy_det` – (n, 2) ndarray or list-like input of n detector (x, y) points `rMat_d` – (3, 3) ndarray, the COB taking DETECTOR FRAME components to LAB FRAME `rMat_s` – (3, 3) ndarray, the COB taking SAMPLE FRAME components to LAB FRAME `tVec_d` – (3, 1) ndarray, the translation vector connecting LAB to DETECTOR `tVec_s` – (3, 1) ndarray, the translation vector connecting LAB to SAMPLE `tVec_c` – (3, 1) ndarray, the translation vector connecting SAMPLE to CRYSTAL

Optional Keyword Arguments: `beamVec` – (3, 1) ndarray containing the incident beam direction components in the LAB FRAME `etaVec` – (3, 1) ndarray containing the reference azimuth direction components in the LAB FRAME

Outputs: (n, 2) ndarray containing the (tTh, eta) pairs associated with each (x, y) (n, 3) ndarray containing the associated G vector directions in the LAB FRAME associated with gVecs

```
oscillAnglesOfHKLs(hkls, chi, rMat_c, bMat, wavelength, vInv=None, beamVec=array([[ -0. ],
[ -0. ],
[ -1. ]]), etaVec=array([[ 1. ],
[ 0. ],
[ 0. ]]))
```

Takes a list of unit reciprocal lattice vectors in crystal frame to the specified detector-relative frame, subject to the conditions:

- 1.the reciprocal lattice vector must be able to satisfy a bragg condition
- 2.the associated diffracted beam must intersect the detector plane

Required Arguments: hkls – (n, 3) ndarray of n reciprocal lattice vectors in the CRYSTAL FRAME chi – float representing the inclination angle of the oscillation axis (std coords) rMat\_c – (3, 3) ndarray, the COB taking CRYSTAL FRAME components to SAMPLE FRAME bMat – (3, 3) ndarray, the COB taking RECIPROCAL LATTICE components to CRYSTAL FRAME wavelength – float representing the x-ray wavelength in Angstroms

Optional Keyword Arguments: beamVec – (3, 1) ndarray containing the incident beam direction components in the LAB FRAME etaVec – (3, 1) ndarray containing the reference azimuth direction components in the LAB FRAME

Outputs: ome0 – (n, 3) ndarray containing the feasible (tTh, eta, ome) triplets for each input hkl (first solution) ome1 – (n, 3) ndarray containing the feasible (tTh, eta, ome) triplets for each input hkl (second solution)

The reciprocal lattice vector, G, will satisfy the the Bragg condition when:

$$\mathbf{b} \cdot \mathbf{T} * \mathbf{G} / \|\mathbf{G}\| = -\sin(\theta)$$

where b is the incident beam direction ( $k_i$ ) and theta is the Bragg angle consistent with G and the specified wavelength. The components of G in the lab frame in this case are obtained using the crystal orientation, Rc, and the single-parameter oscillation matrix, Rs(ome):

$$\mathbf{R}(\text{ome}) * \mathbf{R}_c * \mathbf{G} / \|\mathbf{G}\|$$

The equation above can be rearranged to yield an expression of the form:

$$a * \sin(\text{ome}) + b * \cos(\text{ome}) = c$$

which is solved using the relation:

$$a * \sin(x) + b * \cos(x) = \sqrt{a^2 + b^2} * \sin(x + \alpha)$$

$$\longrightarrow \sin(x + \alpha) = c / \sqrt{a^2 + b^2}$$

where:

$$\alpha = \text{atan2}(b, a)$$

The solutions are:

$$/ \arcsin(c / \sqrt{a^2 + b^2}) - \alpha$$

$$\mathbf{x} = <$$

$$\pi - \arcsin(c / \sqrt{a^2 + b^2}) - \alpha$$

There is a double root in the case the reflection is tangent to the Debye-Scherrer cone ( $c^2 = a^2 + b^2$ ), and no solution if the Laue condition cannot be satisfied (filled with NaNs in the results array here)

`hexrd.xrd.transforms_CAPI.arccosSafe` (*temp*)  
 Protect against numbers slightly larger than 1 in magnitude due to round-off

`hexrd.xrd.transforms_CAPI.angularDifference` (*angList0*, *angList1*, *units='radians'*)  
 Do the proper (acute) angular difference in the context of a branch cut.  
 \*) Default angular range is [-pi, pi]

`hexrd.xrd.transforms_CAPI.mapAngle` (*ang*, *\*args*, *\*\*kwargs*)  
 Utility routine to map an angle into a specified period

`hexrd.xrd.transforms_CAPI.columnNorm` (*a*)  
 normalize array of column vectors (hstacked, axis = 0)

`hexrd.xrd.transforms_CAPI.rowNorm` (*a*)  
 normalize array of row vectors (vstacked, axis = 1)

`hexrd.xrd.transforms_CAPI.unitRowVector` (*vecIn*)

`hexrd.xrd.transforms_CAPI.makeDetectorRotMat` (*tiltAngles*)  
 Form the (3, 3) tilt rotations from the tilt angle list:  
 tiltAngles = [gamma\_Xl, gamma\_Yl, gamma\_Zl] in radians

`hexrd.xrd.transforms_CAPI.makeOscillRotMat` (*oscillAngles*)  
 oscillAngles = [chi, ome]

`hexrd.xrd.transforms_CAPI.makeRotMatOfExpMap` (*expMap*)  
 make a rotation matrix from an exponential map

`hexrd.xrd.transforms_CAPI.makeRotMatOfQuat` (*quat*)  
 make rotation matrix from a unit quaternion  
 ...check to set if input is unit magnitude?

`hexrd.xrd.transforms_CAPI.makeBinaryRotMat` (*axis*)

`hexrd.xrd.transforms_CAPI.makeEtaFrameRotMat` (*bHat\_l*, *eHat\_l*)

`hexrd.xrd.transforms_CAPI.validateAngleRanges` (*angList*, *angMin*, *angMax*, *ccw=True*)

`hexrd.xrd.transforms_CAPI.rotate_vecs_about_axis` (*angle*, *axis*, *vecs*)

`hexrd.xrd.transforms_CAPI.quat_distance` (*q1*, *q2*, *qsym*)  
 qsy coming from `hexrd.xrd.crystallogray.PlaneData.getQSym()` is C-contiguous

## 1.1.28 Module: `xrd.xrdbase`

### 2 Functions

`hexrd.xrd.xrdbase.dataToFrame` (*data*, *sumImg=True*)  
 utility function to allow flexibility in input  
 data can be: (\*) an instance of ReadGE or the like, which is already set up, in which  
     case all frames are used and flattened  
 (\*) a frame

`hexrd.xrd.xrdbase.getGaussNDParams` (*xList*, *w=None*, *v=None*)

## 1.1.29 Module: `xrd.xrutil`

### 6 Classes

**class** `hexrd.xrd.xrutil.FormatEtaOme` (*etas, omes, A, T=False, debug=False*)

for plotting data as a matrix, with `ijAsXY=True`

`__init__` (*etas, omes, A, T=False, debug=False*)

**class** `hexrd.xrd.xrutil.OmeEtaPfig` (*omeEdges, etaEdges, cmap=None, vMM=None, doRot=False, invertFromSouthern=True, netStyle=None, netNDiv=12, netAlpha=0.5, pointLists=[], drawColorbar=True*)

Bases: `object`

works with data from with `CollapseOmeEta`

`__init__` (*omeEdges, etaEdges, cmap=None, vMM=None, doRot=False, invertFromSouthern=True, netStyle=None, netNDiv=12, netAlpha=0.5, pointLists=[], drawColorbar=True*)

**clearLines** ()

get rid of any existing lines and legends

**display** (*data, tTh, nP=True, opacity=[None], rangeVV\_w=None, winKwargs={}*)

if want to do interpolated results, `matplotlib.delaunay` has some triangulation and interpolation stuff, but have not sorted that out yet

**drawLines** (*nP, rMat=None, pointLists=None*)

if specify a `pointLists` argument then it replaces `self.pointLists`

**class** `hexrd.xrd.xrutil.CollapseOmeEta` (*readerList, planeData, hklList, detectorGeom, strainMag=None, nEtaBins=480, mask=None, threshold=None, applyLorenz=True, nframesLump=1, debug=False, computeMeanTwoTheta=False*)

Bases: `object`

Can pass a mask to use in addition to whatever the readers are already set up to do; with frames set zero where mask is True

If pass `pw`: after plot is made, can do calls on `self.pw` to plot other things on top of the image

while this takes a list of readers, it is not yet really coded for disjoint omega ranges

`__init__` (*readerList, planeData, hklList, detectorGeom, strainMag=None, nEtaBins=480, mask=None, threshold=None, applyLorenz=True, nframesLump=1, debug=False, computeMeanTwoTheta=False*)

**display** (*iData=0, rangeVV=None, cmap=None, pw=None, debug=False, pfig=False, comment='', tTh=False, rangeVV\_w=None, winKwargs={}, pfigKwargs={}*)  
`iData` is index into `hklList` passed on initialization, NOT into `planeData`

`pointList`, if specified is a list of tuples of the form `(3xn ndarray, style)` where style gets passed to `plotwrap`

can set `tTh` to 'withOpacity' if the 2-theta plot should be made with opacity (alpha) constructed from the intensity weights; so far this only works for `pfig=False`

**etas = None**

count frames

**iEtaBin = None**

protect against numerical silliness

**indices = None**

make `valsThese` into float so that do not overrun max int value



**tThRanges = None**  
 set up data

**class** hexrd.xrd.xrutil.**GrainPoles** (*omeEtaPfigs*)

**\_\_init\_\_** (*omeEtaPfigs*)

**class** hexrd.xrd.xrutil.**MultiSlopeFunc** (*m1, m2, xcrit, power=0.2*)

function that transitions smoothly from slope of m1 to slope of m2 in the vicinity of xcrit, with the smoothness of the transition dictated by a specified power; for large values of power, might eventually want to put in code to protect against numerical overflow

**\_\_init\_\_** (*m1, m2, xcrit, power=0.2*)

**class** hexrd.xrd.xrutil.**MultiSlopeFuncSmooth** (*m1, w1, p1, m2, xcrit, power=0.2*)

function that transitions smoothly from slope of m1 to slope of m2 in the vicinity of xcrit, with the smoothness of the transition dictated by a specified power;

**\_\_init\_\_** (*m1, w1, p1, m2, xcrit, power=0.2*)

## 48 Functions

hexrd.xrd.xrutil.**fitLParm** (*data, detectorGeom, planeData, funcType='pv', lParm0=None, funcXVecList=None, quadr1d=8, debug=False*)

fit lattice parameters to data; using dataToFrame to map the data to a frame

input planeData is not changed

hexrd.xrd.xrutil.**fitDG** (*data, detectorGeom, planeData, funcType='pv', funcXVecList=None, quadr1d=8, debug=False*)

fit detector geometry parameters to data; using dataToFrame to map the data to a frame

pass funcXVecList as True or as something like detectorGeom.fitRingsFunc.getFuncXVecList() if want to just refine detector geometry and not the functional forms for the rings

input detectorGeom is used to guess parameters and is not modified – a new detectorGeom is returned

hexrd.xrd.xrutil.**fitDGX** (*data, detectorGeom, planeData, funcType='pv', quadr1d=8, debug=False, nGIter=2, xFuncs=None, xDG=None*)

fit detector geometry parameters to data; using dataToFrame to map the data to a frame

uses a procedure that might end up being more robust than fitDG

input detectorGeom is used to guess parameters and is not modified – a new detectorGeom is returned

hexrd.xrd.xrutil.**textureToSpots** (*texture, planeData, detectorGeom, omeMM=None, etaMM=None, pVecs=None*)

take texture as returned from pyMps and make spots

hexrd.xrd.xrutil.**makeSynthSpots** (*rMats, pVecs, bMats, planeData, detectorGeom, omeMM=[-3.141592653589793, 3.141592653589793], etaMM=None, hklList=None, beamSize=None*)

make synthetic spots

hexrd.xrd.xrutil.**makePathVariantPoles** (*rMatRef, fromPhase, pathList, planeDataDict, hklID*)

```
hexrd.xrd.xrutil.displayPathVariants(data, rMatRef, fromPhase, pathList, planeData-Dict, detectorGeom, omeMin, omeMax, phaseForDfltPD=None, markerList=['D', 'o', 'p', 's', 'v', 'x', '+', '*', '<', '>', '1', '2', '3', '4', '^'], hklList=None, color=None, pointKwargs={}, hklIDs=None, pw=None)
```

```
hexrd.xrd.xrutil.makeRMatList(pathList, fromPhase)
```

```
hexrd.xrd.xrutil.findGrainsNewSpots(grainList0, spots, doFitting=True, minCompleteness=None, pathList=[], eosDict=None, refPDDict=None, tK=300.0, debug=True, indepFitPVec=False, findByPixelDist=1, maxIterRefit=3, pVecTol=None)
```

see if grains in grainList0 show up in spots; meant to be useful for taking existing grains from a load step and looking for them in a new load step; returns a new list of the same length, with None wherever a grain near the existing one was not found, and a list of grains for each path in pathList; grains in grainList0 are not modified; spots should be a Spots instance

```
hexrd.xrd.xrutil.stretchToLV(V, fMat)
```

from stretch V in crystal frame and fMat, compute new lattice parameters; fMat can be the 'F' from lparms.latticeVectors

$V = B + I$ , where B is the Biot strain

```
hexrd.xrd.xrutil.calculateBiotStrain(initLP, finalLP, display=False)
```

```
hexrd.xrd.xrutil.makeMNConn(m, n, tri=True)
```

m and n are number of edges, so 1 more than number of zones

```
hexrd.xrd.xrutil.makeNVecs(tth, etaIn, omeIn, asGrid=False)
```

```
hexrd.xrd.xrutil.omeEtaGridToNVecs(tTh, omegas, etas)
```

```
hexrd.xrd.xrutil.roty90(v)
```

```
hexrd.xrd.xrutil.makeMeasuredScatteringVectors(tTh, eta, ome, convention='hexrd', frame='sample')
```

Construct scattering vectors from angles (2theta, eta, ome) will do HEXRD/APS and Fable frames, sample or lab.

for fable frame geomtery, see:

[http://sourceforge.net/apps/trac/fable/attachment/wiki/WikiStart/Geometry\\_version\\_1.0.8.pdf](http://sourceforge.net/apps/trac/fable/attachment/wiki/WikiStart/Geometry_version_1.0.8.pdf)

```
hexrd.xrd.xrutil.doItMP(nSkip)
```

```
hexrd.xrd.xrutil.readFrameStack_multiproc(reader, func, nPerChunk=None, nCPUs=None, debug=False)
```

read the full frame stack using multiprocessing, applying fund to each frame to obtain the result;

use makeNew for each chunk; if reader.dark is a shared memory array then it remains so

```
hexrd.xrd.xrutil.thresholdStackDisplay(data, threshold, cmap=None, pw=None, detectorGeom=None, planeData=None, displayKWArgs={}, drawRingsKWArgs={})
```

passes sumImg=num.maximum to dataToFrame so that if data is a reader then frame ends up being the maximum over the image stack

```
hexrd.xrd.xrutil.grainPolesGUI(omeEtaPfigs)
```

GUI with sliders for rotating a grain's spots

execfile('examples/feSynthSpotsPfig.py') gui = grainPolesGUI([pwSB])

`hexrd.xrutil.darkFromStack` (*reader*, *nFrames*=0, *nChunk*=4, *medianSize*=None, *medianRange*=(-15, 15), *cutMinFactor*=None, *checkIntensityResolution*=False)

If *nFrames* == 0, read all frames.

If *medianSize* is specified then a median filter of the given size is used to find dead pixels, with pixels outside of *medianRange* marked as dead.

`hexrd.xrutil.tryFromShelf` (*shelfFileName*, *thingName*)  
try to pull the thing from the shelf and return None if it does not work

`hexrd.xrutil.putInShelf` (*shelfFileName*, *thingName*, *thing*)

`hexrd.xrutil.pullFromStack` (*reader*, *detectorGeom*, *tThMM*, *angWidth*, *angCen*, *threshold*=20, *distInAng*=False, *padSpot*=True, *mask3D*=None, *exitOnFail*=False)  
*angWidth* is distance from *angCen*, so actually half-width

do not yet deal with wrap-around (eg, reader that spans 2\*pi)

`hexrd.xrutil.grainSpotsFromStack` (*g*, *reader*, *detectorGeom*, *angWidth*, *threshold*, *\*\*kwargs*)  
wrapper around `spotFromStack`; takes a grain and returns a dictionary of spots for the grain, with keys (hklID, iThisHKL)

*angWidth* is for orientation spread; for 2-theta, *g.planeData* 2-theta ranges are used

can specify hklIDs to look for just a subset of spots

set *distInAng*=False if want the spots to have to contain the predicted angles, otherwise, the closest spots in the bounding boxes will be returned

`hexrd.xrutil.spotFromStack` (*reader*, *detectorGeom*, *tThMM*, *angWidth*, *angCen*, *threshold*, *fullBackground*=False, *asFrame*=False, *exitOnFail*=True, *distInAng*=False, *debug*=True)

if *asFrame*, then omegas come out as frame indices; note that *angCen* should still be specified as omega, not a frame index

`hexrd.xrutil.collapse` (*vAll*, *eta*, *ome*, *nOme*=None, *nEta*=None, *weightedList*=[], *averagedList*=[], *auxList*=[], *debug*=False)

Returns a sparse matrix, with zeros where no intensity falls;

pass *nEta* and *nOme* to control the number of eta and omega bins, otherwise they are determined automatically, with the omega binning assuming that omegas fall on frames at regular intervals with no gaps

for each entry in *weightedList*, also returns that data collapsed and weighted by *vAll*; similarly for *averagedList*

`hexrd.xrutil.displaySparse` (*a*, *vmin*=None, *vmax*=None, *cmap*=None, *fnt*=None, *marker-size*=3, *colorUnder*=None, *ijNZ*=None, *\*\*kwargs*)

`hexrd.xrutil.displayEtaOme` (*eta*, *ome*, *vAll*, *nEta*=None, *nOme*=None, *\*\*kwargs*)

`hexrd.xrutil.getLorenz` (*detectorGeom*, *\*args*)

`hexrd.xrutil.getEtaResolution` (*detectorGeom*, *tTh*)  
angular resolution along eta

`hexrd.xrutil.getTThResolution` (*detectorGeom*, *tTh*)

`hexrd.xrutil.drEtaOme` (*angCen*, *dEta*, *dOme*)  
compute true angular changes and dA corresponding to steps in eta and omega

`hexrd.xrutil.omeEtaGridToDA` (*tThNominal*, *etaEdges*, *omeEdges*)  
get grid patch areas, in the sense of solid angle (pole figure) coverage;

`hexrd.xrd.xrutil.bboxIntersect3D (b1, b2)`

0-tolerance bounding box intersection in 3d

`hexrd.xrd.xrutil.pfigFromSpots (spots, iHKL, phaseID=None, nOme=None, nEta=None, tThTol=None, etaExcl=0.1, plot=False, plotPrefix=None, debug=False)`

probably want to have collected spots without discarding those at boundaries (discardAtBounds=False)

depending on the context, may need to have done something like `iHKL = hklIDs.index(hklID)`

if `nEta` is negative, it is treated as the target lumping of pixels

`etaExcl` is in radians – etas within this range of  $\pm\pi/2$  degrees are left out; can set to `None` to turn this behavior off

can use `tThTol` to tighten down the two-theta tolerance

`hexrd.xrd.xrutil.mapAngCen (ang, angCen)`

map angle `ang` into equivalent value that is closest to `angCen`

`hexrd.xrd.xrutil.makeSynthFrames (spotParamsList, detectorGeom, omegas, intensityFunc=<hexrd.xrd.spotfinder.IntensityFuncGauss3D object at 0x7fad81e6f610>, asSparse=None, output=None, cutoffMult=4.0, debug=1)`

`intensityFunc` is an instance of a class that works as an intensity fuction.

`spotParamsList` should be a list with each entry being a list of arguments appropriate to the `intensityFunc.constructParams` function. For `intensityFunc=spotfinder.IntensityFuncGauss3D()`, each `spotParamsList` entry should be (center, fwhm, A), with center being the 3D spot center in angular coordinates (radians), fwhm being the (2-theta, eta, omega) widths in 3D, and A being an intensity scaling.

If output is specified as a string, then the frames with the given prefix are dumped to files instead of being accumulated in memory. If output is callable then frames are passed to `output()`.

If `asSparse` is true then sparse matrices are used to reduce memory footprint. The `asSparse` option is currently not coded for the case of output having been specied.

`cutoffMult` is the multiplier on the FWHM to determine the angular cutoff range for evaluating intensity for each spot.

`hexrd.xrd.xrutil.validateAngleRanges (angList, startAngs, stopAngs, ccw=True)`

A better way to go. find out if an angle is in the range CCW or CW from start to stop

There is, of course an ambigutiy if the start and stop angle are the same; we treat them as implying  $2\pi$

`hexrd.xrd.xrutil.tVec_d_from_old_parfile (old_par, detOrigin)`

`hexrd.xrd.xrutil.objFun_tVec_d (tvd_xy, rMat_d, beamXYD, detOrigin, bHat_l)`

`hexrd.xrd.xrutil.beamXYD_from_tVec_d (rMat_d, tVec_d, bVec_ref, detOrigin)`

`hexrd.xrd.xrutil.write_old_parfile (filename, results)`

`simulateOmeEtaMaps (omeEdges, etaEdges, planeData, expMaps, chi=0.0, etaTol=None, omeTol=None, [-0.], [-1.]), eVec=array([[ 1.], [ 0.], [ 0.]]), vInv=array([[ 1.], [ 1.], [ 1.], [ 0.], [ 0.]])`

```
[ 0.]]))
```

all angular info is entered in degrees

quats are (4, n)

...might want to creat module-level angluar unit flag ...might want to allow resvers delta omega

```
hexrd.xrd.xrutil.simulateGVecs(pd, detector_params, grain_params, ome_range=[(-
    3.141592653589793, 3.141592653589793)], ome_period=(-
    3.141592653589793, 3.141592653589793), eta_range=[(-
    3.141592653589793, 3.141592653589793)], panel_dims=[(-
    204.8, -204.8), (204.8, 204.8)], pixel_pitch=(0.2, 0.2),
    distortion=(<function dummy at 0x7fad8697caa0>, []))
```

simulate the monochromatic scattering for a specified

- space group
- wavelength
- orientation
- strain
- position
- detector parameters
- oscillation axis tilt (chi)

subject to

- omega (oscillation) ranges (list of (min, max) tuples)
- eta (azimuth) ranges

pd.....a hexrd.xrd.crystallography.PlaneData instance detector\_params...a (10,) ndarray containing the tilt angles (3), translation (3),

chi (1), and sample frame translation (3) parameters

**grain\_params.....a (12,) ndarray containing the exponential map (3), translation (3), and inverse stretch tensor compnents in Mandel-Voigt notation (6).**

- currently only one panel is supported, but this will likely change very soon

```
hexrd.xrd.xrutil.angularPixelSize(xy_det, xy_pixelPitch, rMat_d, rMat_s, tVec_d,
    tVec_s, tVec_c, distortion=(<function dummy at
    0x7fad8697caa0>[ ], ))
```

- choices to beam vector and eta vector specs have been supressed
- assumes xy\_det in UNWARPED configuration

```
hexrd.xrd.xrutil.pullSpots(pd, detector_params, grain_params, reader, ome_period=(-
    3.141592653589793, 3.141592653589793), eta_range=[(-
    3.141592653589793, 3.141592653589793)], panel_dims=[(-204.8,
    -204.8), (204.8, 204.8)], panel_buff=[20, 20], pixel_pitch=(0.2,
    0.2), distortion=(<function dummy at 0x7fad8697caa0>, []),
    tth_tol=0.15, eta_tol=1.0, ome_tol=1.0, npdiv=1, thresh-
    old=10, doClipping=False, filename=None, save_spot_list=False,
    use_closest=False)
```

```
hexrd.xrd.xrutil.validateQVecAngles(*args, **kwargs)
```



## h

- `hexrd.arrayutil`, 2
- `hexrd.fileutil`, 2
- `hexrd.gridutil`, 3
- `hexrd.matrixutil`, 4
- `hexrd.orientations`, 7
- `hexrd.pfigutil`, 10
- `hexrd.plotwrap`, 10
- `hexrd.quadrature.q1db`, 12
- `hexrd.quadrature.q2db`, 12
- `hexrd.quadrature.q3db`, 12
- `hexrd.tens`, 13
- `hexrd.valunits`, 15
- `hexrd.xrd.crystallography`, 16
- `hexrd.xrd.detector`, 20
- `hexrd.xrd.distortion`, 31
- `hexrd.xrd.experiment`, 31
- `hexrd.xrd.fitting`, 36
- `hexrd.xrd.grain`, 37
- `hexrd.xrd.hydra`, 39
- `hexrd.xrd.indexer`, 40
- `hexrd.xrd.material`, 41
- `hexrd.xrd.rotations`, 42
- `hexrd.xrd.spacegroup`, 44
- `hexrd.xrd.spotfinder`, 46
- `hexrd.xrd.symmetry`, 52
- `hexrd.xrd.transforms`, 52
- `hexrd.xrd.transforms_CAPI`, 57
- `hexrd.xrd.xrdbase`, 59
- `hexrd.xrd.xrutil`, 60





## Symbols

- `__init__()` (hexrd.fileutil.FileDescr method), 2
- `__init__()` (hexrd.fileutil.FileForm method), 2
- `__init__()` (hexrd.fileutil.FileLink method), 2
- `__init__()` (hexrd.fileutil.FileLinkWild method), 2
- `__init__()` (hexrd.fileutil.Log method), 2
- `__init__()` (hexrd.orientations.BungeEuler method), 7
- `__init__()` (hexrd.orientations.CanovaEuler method), 7
- `__init__()` (hexrd.orientations.Fiber method), 8
- `__init__()` (hexrd.orientations.KocksEuler method), 7
- `__init__()` (hexrd.orientations.Quat method), 7
- `__init__()` (hexrd.orientations.RotInv method), 7
- `__init__()` (hexrd.orientations.RotationParameterization method), 7
- `__init__()` (hexrd.orientations.SymmGroup method), 8
- `__init__()` (hexrd.plotwrap.PlotWin method), 11
- `__init__()` (hexrd.plotwrap.PlotWinLite method), 11
- `__init__()` (hexrd.plotwrap.PlotWinP method), 11
- `__init__()` (hexrd.plotwrap.PlotWrap method), 11
- `__init__()` (hexrd.tens.T2Svec method), 13
- `__init__()` (hexrd.tens.T2SvecP method), 13
- `__init__()` (hexrd.tens.T2Symm method), 13
- `__init__()` (hexrd.tens.T2Vecds method), 13
- `__init__()` (hexrd.valunits.valWUnit method), 15
- `__init__()` (hexrd.xrd.crystallography.PlaneData method), 16
- `__init__()` (hexrd.xrd.detector.Detector2DRC method), 26
- `__init__()` (hexrd.xrd.detector.DetectorBase method), 26
- `__init__()` (hexrd.xrd.detector.DetectorGeomFrelon method), 29
- `__init__()` (hexrd.xrd.detector.DetectorGeomGE method), 28
- `__init__()` (hexrd.xrd.detector.DetectorGeomMar165 method), 28
- `__init__()` (hexrd.xrd.detector.DetectorGeomQuadGE method), 29
- `__init__()` (hexrd.xrd.detector.FmtCoordIdeal method), 20
- `__init__()` (hexrd.xrd.detector.FrameWriter method), 21
- `__init__()` (hexrd.xrd.detector.Framer2DRC method), 21
- `__init__()` (hexrd.xrd.detector.LineStyles method), 23
- `__init__()` (hexrd.xrd.detector.MultiRingBinned method), 24
- `__init__()` (hexrd.xrd.detector.MultiRingEval method), 25
- `__init__()` (hexrd.xrd.detector.Peak1DAtLoc method), 23
- `__init__()` (hexrd.xrd.detector.PeakGauss1DAtLoc method), 24
- `__init__()` (hexrd.xrd.detector.PeakLorentzian1DAtLoc method), 24
- `__init__()` (hexrd.xrd.detector.PeakPV1DAtLoc method), 23
- `__init__()` (hexrd.xrd.detector.ReadGE method), 22
- `__init__()` (hexrd.xrd.detector.ReadGeneric method), 21
- `__init__()` (hexrd.xrd.detector.ReadMar165 method), 23
- `__init__()` (hexrd.xrd.detector.ReadMar165NB1 method), 23
- `__init__()` (hexrd.xrd.detector.ReadMar165NB2 method), 23
- `__init__()` (hexrd.xrd.detector.ReadMar165NB3 method), 23
- `__init__()` (hexrd.xrd.detector.ReadMar165NB4 method), 23
- `__init__()` (hexrd.xrd.detector.ThreadReadFrame method), 20
- `__init__()` (hexrd.xrd.experiment.CalibrationInput method), 35
- `__init__()` (hexrd.xrd.experiment.DetectorInfo method), 35
- `__init__()` (hexrd.xrd.experiment.Experiment method), 31
- `__init__()` (hexrd.xrd.experiment.IndexOptions method), 36
- `__init__()` (hexrd.xrd.experiment.PolarRebinOpts method), 36
- `__init__()` (hexrd.xrd.experiment.ReaderInput method), 34
- `__init__()` (hexrd.xrd.experiment.SpotOptions method), 36
- `__init__()` (hexrd.xrd.grain.Grain method), 37
- `__init__()` (hexrd.xrd.hydra.Hydra method), 39

[\\_\\_init\\_\\_\(\)](#) (hexrd.xrd.indexer.GrainSpotter method), 40  
[\\_\\_init\\_\\_\(\)](#) (hexrd.xrd.material.Material method), 42  
[\\_\\_init\\_\\_\(\)](#) (hexrd.xrd.spacegroup.SpaceGroup method), 45  
[\\_\\_init\\_\\_\(\)](#) (hexrd.xrd.spotfinder.FitFailedError method), 47  
[\\_\\_init\\_\\_\(\)](#) (hexrd.xrd.spotfinder.IntensityFunc2D method), 46  
[\\_\\_init\\_\\_\(\)](#) (hexrd.xrd.spotfinder.IntensityFunc3D method), 46  
[\\_\\_init\\_\\_\(\)](#) (hexrd.xrd.spotfinder.IntensityFuncGauss2D method), 47  
[\\_\\_init\\_\\_\(\)](#) (hexrd.xrd.spotfinder.IntensityFuncGauss3D method), 46  
[\\_\\_init\\_\\_\(\)](#) (hexrd.xrd.spotfinder.IntensityFuncGauss3DGenerate method), 46  
[\\_\\_init\\_\\_\(\)](#) (hexrd.xrd.spotfinder.IntensityFuncMulti2D method), 47  
[\\_\\_init\\_\\_\(\)](#) (hexrd.xrd.spotfinder.IntensityFuncMulti3D method), 46  
[\\_\\_init\\_\\_\(\)](#) (hexrd.xrd.spotfinder.Spot method), 47  
[\\_\\_init\\_\\_\(\)](#) (hexrd.xrd.spotfinder.Spots method), 49  
[\\_\\_init\\_\\_\(\)](#) (hexrd.xrd.spotfinder.SpotsIterator method), 50  
[\\_\\_init\\_\\_\(\)](#) (hexrd.xrd.spotfinder.UnfitableError method), 47  
[\\_\\_init\\_\\_\(\)](#) (hexrd.xrd.xrutil.CollapseOmeEta method), 60  
[\\_\\_init\\_\\_\(\)](#) (hexrd.xrd.xrutil.FormatEtaOme method), 60  
[\\_\\_init\\_\\_\(\)](#) (hexrd.xrd.xrutil.GrainPoles method), 61  
[\\_\\_init\\_\\_\(\)](#) (hexrd.xrd.xrutil.MultiSlopeFunc method), 61  
[\\_\\_init\\_\\_\(\)](#) (hexrd.xrd.xrutil.MultiSlopeFuncSmooth method), 61  
[\\_\\_init\\_\\_\(\)](#) (hexrd.xrd.xrutil.OmeEtaPfig method), 60

## A

[active\\_img](#) (hexrd.xrd.experiment.Experiment attribute), 32  
[activeImage](#) (hexrd.xrd.experiment.Experiment attribute), 32  
[activeMaterial](#) (hexrd.xrd.experiment.Experiment attribute), 32  
[activeReader](#) (hexrd.xrd.experiment.Experiment attribute), 32  
[add\\_to\\_img\\_list\(\)](#) (hexrd.xrd.experiment.Experiment method), 32  
[aggMode](#) (hexrd.xrd.experiment.ReaderInput attribute), 35  
[aggModeOp](#) (hexrd.xrd.experiment.ReaderInput attribute), 35  
[angCOM\(\)](#) (hexrd.xrd.spotfinder.Spot method), 47  
[angleAxisOfRotMat\(\)](#) (in module hexrd.xrd.rotations), 43  
[anglesToGVec\(\)](#) (in module hexrd.xrd.transforms), 53

[angOnDetector\(\)](#) (hexrd.xrd.detector.Detector2DRC method), 26  
[angToXYIdeal\(\)](#) (in module hexrd.xrd.detector), 30  
[angToXYO\(\)](#) (hexrd.xrd.detector.Detector2DRC method), 26  
[angToXYO\\_V\(\)](#) (hexrd.xrd.detector.Detector2DRC method), 26  
[angToXYOBBBox\(\)](#) (hexrd.xrd.detector.Detector2DRC method), 26  
[angularDifference\(\)](#) (in module hexrd.xrd.transforms), 55  
[angularDifference\(\)](#) (in module hexrd.xrd.transforms\_CAPI), 59  
[angularDifference\\_opt\(\)](#) (in module hexrd.xrd.rotations), 44  
[angularDifference\\_orig\(\)](#) (in module hexrd.xrd.rotations), 44  
[angularPixelSize\(\)](#) (in module hexrd.xrd.xrutil), 65  
[append\(\)](#) (hexrd.xrd.spotfinder.Spot method), 47  
[applySym\(\)](#) (in module hexrd.xrd.symmetry), 52  
[arccosSafe\(\)](#) (in module hexrd.orientations), 8  
[arccosSafe\(\)](#) (in module hexrd.xrd.rotations), 42  
[arccosSafe\(\)](#) (in module hexrd.xrd.transforms), 55  
[arccosSafe\(\)](#) (in module hexrd.xrd.transforms\_CAPI), 58  
[archiveDir\(\)](#) (in module hexrd.fileutil), 3  
[archiveFile\(\)](#) (in module hexrd.fileutil), 3  
[argListToStr\(\)](#) (in module hexrd.fileutil), 3  
[argToPW\(\)](#) (in module hexrd.plotwrap), 11  
[arrayToString\(\)](#) (in module hexrd.arrayutil), 2  
[autoTicks\(\)](#) (in module hexrd.plotwrap), 11

## B

[bboxIntersect3D\(\)](#) (in module hexrd.xrd.xrutil), 63  
[beamEnergy](#) (hexrd.xrd.material.Material attribute), 42  
[beamXYD\\_from\\_tVec\\_d\(\)](#) (in module hexrd.xrd.xrutil), 64  
[blockSparseOfMatArray\(\)](#) (in module hexrd.matrixutil), 4  
[bMat](#) (hexrd.xrd.grain.Grain attribute), 38  
[BungeEuler](#) (class in hexrd.orientations), 7  
[bungeToMat\(\)](#) (in module hexrd.orientations), 8

## C

[cakeArgs](#) (hexrd.xrd.experiment.CalibrationInput attribute), 35  
[calculateBiotStrain\(\)](#) (in module hexrd.xrd.xrutil), 62  
[calData](#) (hexrd.xrd.experiment.CalibrationInput attribute), 35  
[calibrate\(\)](#) (hexrd.xrd.experiment.DetectorInfo method), 35  
[calibrate\(\)](#) (hexrd.xrd.experiment.Experiment method), 32  
[CalibrationInput](#) (class in hexrd.xrd.experiment), 35  
[calInput](#) (hexrd.xrd.experiment.Experiment attribute), 32  
[calMat](#) (hexrd.xrd.experiment.CalibrationInput attribute), 35

- CanovaEuler (class in hexrd.orientations), 7  
cartesianCoordsOfPixelIndices()  
    (hexrd.xrd.detector.Detector2DRC method), 26  
catList() (in module hexrd.fileutil), 2  
cellCentroids() (in module hexrd.gridutil), 4  
cellConnectivity() (in module hexrd.gridutil), 4  
cellIndices() (in module hexrd.gridutil), 3  
checkClaim() (hexrd.xrd.spotfinder.Spots method), 49  
checkClaims() (hexrd.xrd.grain.Grain method), 38  
checkClaims() (hexrd.xrd.spotfinder.Spots method), 49  
claimSpots() (hexrd.xrd.grain.Grain method), 38  
claimSpots() (hexrd.xrd.spotfinder.Spots method), 49  
cleanFit() (hexrd.xrd.spotfinder.Spot method), 47  
cleanMulti() (hexrd.xrd.spotfinder.Spots method), 49  
clear\_reader() (hexrd.xrd.experiment.Experiment method), 32  
clear\_spots() (hexrd.xrd.experiment.Experiment method), 32  
clearLines() (hexrd.xrd.xrutil.OmeEtaPfig method), 60  
collapse() (in module hexrd.xrd.xrutil), 63  
CollapseOmeEta (class in hexrd.xrd.xrutil), 60  
colorbar() (hexrd.plotwrap.PlotWrap method), 11  
columnNorm() (in module hexrd.matrixutil), 4  
columnNorm() (in module hexrd.xrd.transforms), 55  
columnNorm() (in module hexrd.xrd.transforms\_CAPI), 59  
computeArea() (in module hexrd.gridutil), 4  
computeIntersection() (in module hexrd.gridutil), 4  
convertUToRotMat() (in module hexrd.xrd.indexer), 40  
copyBox() (in module hexrd.xrd.spotfinder), 51  
cosineXform() (in module hexrd.xrd.crystallography), 17  
cross() (in module hexrd.matrixutil), 6  
cullSpots() (hexrd.xrd.spotfinder.Spot static method), 47  
cullSpotUsingBin() (in module hexrd.xrd.spotfinder), 51  
curFrameNumber (hexrd.xrd.experiment.Experiment attribute), 32
- ## D
- d\_dCenters() (hexrd.xrd.detector.Peak1DAAtLoc method), 23  
d\_dCenters() (hexrd.xrd.detector.PeakGauss1DAAtLoc method), 24  
d\_dp() (hexrd.xrd.detector.Peak1DAAtLoc method), 23  
d\_dx() (hexrd.xrd.detector.Peak1DAAtLoc method), 23  
d\_dx() (hexrd.xrd.detector.PeakGauss1DAAtLoc method), 24  
d\_dx() (hexrd.xrd.detector.PeakLorentzian1DAAtLoc method), 24  
d\_dx() (hexrd.xrd.detector.PeakPV1DAAtLoc method), 23  
dAiAoH\_svecList() (in module hexrd.tens), 14  
darkFile (hexrd.xrd.experiment.ReaderInput attribute), 35  
darkFromStack() (in module hexrd.xrd.xrutil), 62  
dataToFrame() (in module hexrd.xrd.xrdbase), 59  
destroy() (hexrd.plotwrap.PlotWrap method), 11  
detector (hexrd.xrd.experiment.Experiment attribute), 32  
Detector2DRC (class in hexrd.xrd.detector), 26  
DetectorBase (class in hexrd.xrd.detector), 26  
DetectorGeomFrelon (class in hexrd.xrd.detector), 29  
DetectorGeomGE (class in hexrd.xrd.detector), 28  
DetectorGeomMar165 (class in hexrd.xrd.detector), 28  
DetectorGeomQuadGE (class in hexrd.xrd.detector), 29  
DetectorInfo (class in hexrd.xrd.experiment), 35  
detectorList() (in module hexrd.xrd.detector), 30  
determinant3() (in module hexrd.matrixutil), 7  
deval() (hexrd.xrd.detector.MultiRingEval method), 25  
dgDummy (hexrd.xrd.detector.DetectorGeomQuadGE attribute), 29  
dictToDefs() (in module hexrd.fileutil), 3  
dilateObj() (in module hexrd.xrd.spotfinder), 51  
display() (hexrd.xrd.detector.Detector2DRC method), 26  
display() (hexrd.xrd.detector.ReadGE class method), 22  
display() (hexrd.xrd.spotfinder.Spot method), 47  
display() (hexrd.xrd.xrutil.CollapseOmeEta method), 60  
display() (hexrd.xrd.xrutil.OmeEtaPfig method), 60  
displayEtaOme() (in module hexrd.xrd.xrutil), 63  
displayFlat() (hexrd.xrd.spotfinder.Spot method), 48  
displayFrames() (hexrd.xrd.spotfinder.Spot method), 48  
displayIdeal() (hexrd.xrd.detector.Detector2DRC method), 27  
displayIdeal() (hexrd.xrd.detector.DetectorGeomQuadGE method), 29  
displayPathVariants() (in module hexrd.xrd.xrutil), 61  
displaySparse() (in module hexrd.xrd.xrutil), 63  
displaySub() (hexrd.xrd.detector.DetectorGeomQuadGE method), 29  
distanceToFiber() (in module hexrd.xrd.rotations), 43  
distBetweenFibers() (hexrd.orientations.Fiber method), 8  
doFit() (hexrd.xrd.detector.MultiRingBinned method), 24  
doFit() (hexrd.xrd.detector.MultiRingEval method), 25  
doItMP() (in module hexrd.xrd.xrutil), 62  
doMap (hexrd.xrd.spotfinder.Spot attribute), 48  
doSpotThis() (in module hexrd.xrd.spotfinder), 51  
drawLines() (hexrd.xrd.xrutil.OmeEtaPfig method), 60  
drawLines() (in module hexrd.pfigutil), 10  
drawRings() (hexrd.xrd.detector.Detector2DRC method), 27  
drawRings() (hexrd.xrd.detector.DetectorGeomQuadGE method), 29  
drawRingsGUI() (hexrd.xrd.detector.Detector2DRC method), 27  
drEtaOme() (in module hexrd.xrd.xrutil), 63  
dummy() (in module hexrd.xrd.distortion), 31  
dump\_grainList() (hexrd.xrd.experiment.Experiment method), 32
- ## E
- emptyBox() (in module hexrd.xrd.spotfinder), 51

enslaveSpotData() (hexrd.xrd.spotfinder.Spots static method), 49  
etas (hexrd.xrd.xrutil.CollapseOmeEta attribute), 60  
eval() (hexrd.xrd.detector.MultiRingBinned method), 24  
eval() (hexrd.xrd.detector.MultiRingEval method), 25  
eval() (hexrd.xrd.detector.Peak1DATLoc method), 23  
eval() (hexrd.xrd.spotfinder.IntensityFunc2D method), 46  
eval() (hexrd.xrd.spotfinder.IntensityFunc3D method), 46  
exceptionOnFit() (hexrd.xrd.spotfinder.Spot method), 48  
Experiment (class in hexrd.xrd.experiment), 31  
export\_grainList() (hexrd.xrd.experiment.Experiment method), 32

## F

Fiber (class in hexrd.orientations), 8  
fiberSearch() (in module hexrd.xrd.indexer), 40  
FileDescr (class in hexrd.fileutil), 2  
FileForm (class in hexrd.fileutil), 2  
FileLink (class in hexrd.fileutil), 2  
FileLinkWild (class in hexrd.fileutil), 2  
fileToForm() (in module hexrd.fileutil), 2  
finalize() (hexrd.xrd.spotfinder.Spot method), 48  
find\_raw\_spots() (hexrd.xrd.experiment.Experiment method), 32  
findDistinct() (hexrd.orientations.SymmGroup method), 8  
findDuplicateVectors() (in module hexrd.matrixutil), 6  
findGrainsNewSpots() (in module hexrd.xrd.xrutil), 62  
findSpotsOmegaStack() (hexrd.xrd.spotfinder.Spots static method), 49  
fit() (hexrd.xrd.grain.Grain method), 38  
fit() (hexrd.xrd.spotfinder.Spot method), 48  
fitDG() (in module hexrd.xrd.xrutil), 61  
fitDGX() (in module hexrd.xrd.xrutil), 61  
FitFailedError (class in hexrd.xrd.spotfinder), 47  
fitFloatingCenter() (hexrd.xrd.detector.Peak1DATLoc method), 23  
fitHasFailed() (hexrd.xrd.spotfinder.Spots method), 50  
fitLParm() (in module hexrd.xrd.xrutil), 61  
FitModes (class in hexrd.xrd.experiment), 31  
fitPrecession() (hexrd.xrd.grain.Grain method), 38  
fitProcedureA() (hexrd.xrd.detector.DetectorGeomQuadGE method), 29  
fitRMats (hexrd.xrd.experiment.Experiment attribute), 32  
fitSpots() (hexrd.xrd.spotfinder.Spots method), 50  
fitSpotsMulti() (hexrd.xrd.spotfinder.Spots method), 50  
fitType (hexrd.xrd.experiment.CalibrationInput attribute), 35  
fitWrap() (hexrd.xrd.spotfinder.Spot method), 48  
fixQuat() (in module hexrd.xrd.rotations), 42  
flattenOmega() (hexrd.xrd.spotfinder.Spot method), 48  
floatingCentersIJ (hexrd.xrd.detector.MultiRingBinned attribute), 24  
fMat (hexrd.xrd.grain.Grain attribute), 38

FmtCoordIdeal (class in hexrd.xrd.detector), 20  
FormatEtaOme (class in hexrd.xrd.xrutil), 60  
frameInRange() (in module hexrd.xrd.detector), 30  
Framer2DRC (class in hexrd.xrd.detector), 21  
FrameWriter (class in hexrd.xrd.detector), 21  
fromSouthern() (in module hexrd.pfigutil), 10

## G

GE\_41RT() (in module hexrd.xrd.distortion), 31  
geomParamsToInput() (in module hexrd.xrd.fitting), 37  
get2DFunc() (hexrd.xrd.spotfinder.IntensityFuncGauss3DGenElI class method), 46  
get\_hkls() (hexrd.xrd.crystallography.PlaneData method), 17  
get\_spots\_ind() (hexrd.xrd.experiment.Experiment method), 32  
getAlignmentRotation() (hexrd.xrd.grain.Grain method), 38  
getAngCoords() (hexrd.xrd.spotfinder.Spots method), 50  
getAngPixelSize() (hexrd.xrd.detector.Detector2DRC method), 27  
getAxes() (hexrd.plotwrap.PlotWin method), 11  
getBankNames() (in module hexrd.fileutil), 3  
getBin() (in module hexrd.xrd.spotfinder), 51  
getCellVolume() (hexrd.xrd.grain.Grain method), 38  
getCentered() (in module hexrd.xrd.detector), 30  
getCMap() (in module hexrd.xrd.detector), 30  
getDark() (hexrd.xrd.detector.Framer2DRC method), 21  
getDark() (hexrd.xrd.detector.ReadGeneric method), 21  
getDD\_tThs\_lparms() (hexrd.xrd.crystallography.PlaneData method), 16  
getDeltaOmega() (hexrd.xrd.detector.Framer2DRC method), 21  
getDoMap() (hexrd.xrd.spotfinder.Spot method), 48  
getDparms() (in module hexrd.xrd.crystallography), 20  
getDtype() (in module hexrd.xrd.spotfinder), 51  
getEmptyMask() (hexrd.xrd.detector.Framer2DRC method), 21  
getEtaResolution() (in module hexrd.xrd.xrutil), 63  
getFitResid() (hexrd.xrd.grain.Grain method), 38  
getFrameOmega() (hexrd.xrd.detector.Framer2DRC method), 21  
getFrameOmega() (hexrd.xrd.detector.ReadGE method), 22  
getFrameOmega() (hexrd.xrd.detector.ReadGeneric method), 21  
getFrames() (hexrd.xrd.spotfinder.Spot method), 48  
getFrameUseMask() (hexrd.xrd.detector.ReadGE method), 22  
getFriedelPair() (in module hexrd.xrd.crystallography), 19  
getFromPipe() (in module hexrd.fileutil), 3  
getGaussNDParams() (in module hexrd.xrd.xrdbase), 59

[getHKLID\(\)](#) (hexrd.xrd.crystallography.PlaneData method), [16](#)  
[getHKLs\(\)](#) (hexrd.xrd.crystallography.PlaneData method), [16](#)  
[getHKLs\(\)](#) (hexrd.xrd.spacegroup.SpaceGroup method), [45](#)  
[getHKLSpots\(\)](#) (hexrd.xrd.spotfinder.Spots method), [50](#)  
[getHostName\(\)](#) (in module hexrd.fileutil), [3](#)  
[getImCOM\(\)](#) (in module hexrd.xrd.spotfinder), [51](#)  
[getIndices\(\)](#) (in module hexrd.xrd.spotfinder), [51](#)  
[getIntegratedIntensity\(\)](#) (hexrd.xrd.spotfinder.Spots method), [50](#)  
[getIterHKL\(\)](#) (hexrd.xrd.spotfinder.Spots method), [50](#)  
[getIterPhase\(\)](#) (hexrd.xrd.spotfinder.Spots method), [50](#)  
[getLatticeOperators\(\)](#) (hexrd.xrd.crystallography.PlaneData method), [16](#)  
[getLatticeParams\(\)](#) (hexrd.xrd.grain.Grain method), [38](#)  
[getLatticeType\(\)](#) (hexrd.xrd.crystallography.PlaneData method), [16](#)  
[getLatticeVectors\(\)](#) (hexrd.xrd.grain.Grain method), [38](#)  
[getLaueGroup\(\)](#) (hexrd.xrd.crystallography.PlaneData method), [16](#)  
[getLorenz\(\)](#) (in module hexrd.xrd.xrutil), [63](#)  
[getMem\(\)](#) (in module hexrd.arrayutil), [2](#)  
[getNCorePerNode\(\)](#) (in module hexrd.fileutil), [3](#)  
[getNFrames\(\)](#) (hexrd.xrd.detector.Framer2DRC method), [21](#)  
[getNFrames\(\)](#) (hexrd.xrd.detector.ReadGE method), [22](#)  
[getNFramesFromBytes\(\)](#) (in module hexrd.xrd.detector), [30](#)  
[getNhklRef\(\)](#) (hexrd.xrd.crystallography.PlaneData method), [16](#)  
[getNParams\(\)](#) (hexrd.xrd.detector.PeakGauss1DAtLoc method), [24](#)  
[getNParams\(\)](#) (hexrd.xrd.detector.PeakLorentzian1DAtLoc method), [24](#)  
[getNumberOfFrames\(\)](#) (hexrd.xrd.experiment.ReaderInput method), [35](#)  
[getObjSize\(\)](#) (in module hexrd.xrd.spotfinder), [51](#)  
[getOmegaMMReaderList\(\)](#) (in module hexrd.xrd.detector), [30](#)  
[getParamScalings\(\)](#) (hexrd.xrd.detector.Detector2DRC method), [27](#)  
[getParamScalings\(\)](#) (hexrd.xrd.detector.DetectorBase method), [26](#)  
[getPhaseID\(\)](#) (hexrd.xrd.crystallography.PlaneData method), [16](#)  
[getPixelIsInSpots\(\)](#) (hexrd.xrd.spotfinder.Spots method), [50](#)  
[getPlaneNormals\(\)](#) (hexrd.xrd.crystallography.PlaneData method), [16](#)  
[getPlaneSpacings\(\)](#) (hexrd.xrd.crystallography.PlaneData method), [16](#)  
[getPRBOverlay\(\)](#) (hexrd.xrd.detector.Detector2DRC method), [27](#)  
[getPVecScaling\(\)](#) (hexrd.xrd.detector.DetectorBase method), [26](#)  
[getRandQuat\(\)](#) (hexrd.orientations.Quat static method), [7](#)  
[getReciprocalAlignmentRotation\(\)](#) (hexrd.xrd.grain.Grain method), [38](#)  
[getReciprocalLatticeVectors\(\)](#) (hexrd.xrd.grain.Grain method), [38](#)  
[getReferenceLatticeParams\(\)](#) (hexrd.xrd.grain.Grain method), [38](#)  
[getRefineFlagsDflt\(\)](#) (hexrd.xrd.detector.DetectorGeomQuadGE class method), [30](#)  
[getRightStretchTensor\(\)](#) (hexrd.xrd.grain.Grain method), [38](#)  
[getRings\(\)](#) (hexrd.xrd.detector.Detector2DRC method), [27](#)  
[getSavedReader\(\)](#) (hexrd.xrd.experiment.Experiment method), [32](#)  
[getScratchBaseDir\(\)](#) (in module hexrd.fileutil), [3](#)  
[getSpot\(\)](#) (in module hexrd.xrd.spotfinder), [51](#)  
[getSpotFromPixels\(\)](#) (in module hexrd.xrd.spotfinder), [51](#)  
[getStretchTensor\(\)](#) (hexrd.xrd.grain.Grain method), [38](#)  
[getSymHKLs\(\)](#) (hexrd.xrd.crystallography.PlaneData method), [16](#)  
[getSysType\(\)](#) (in module hexrd.fileutil), [3](#)  
[getTThErrors\(\)](#) (hexrd.xrd.detector.MultiRingBinned method), [25](#)  
[getTThMax\(\)](#) (hexrd.xrd.detector.DetectorGeomQuadGE method), [30](#)  
[getTThRanges\(\)](#) (hexrd.xrd.crystallography.PlaneData method), [16](#)  
[getTThResolution\(\)](#) (in module hexrd.xrd.xrutil), [63](#)  
[getVal\(\)](#) (hexrd.valunits.valWUnit method), [15](#)  
[getValuesOnly\(\)](#) (in module hexrd.xrd.spotfinder), [51](#)  
[getVScale\(\)](#) (hexrd.xrd.detector.Detector2DRC method), [27](#)  
[getWtrShd\(\)](#) (in module hexrd.xrd.spotfinder), [51](#)  
[getXYOCoords\(\)](#) (hexrd.xrd.spotfinder.Spots method), [50](#)  
[Grain](#) (class in hexrd.xrd.grain), [37](#)  
[GrainPoles](#) (class in hexrd.xrd.xrutil), [61](#)  
[grainPolesGUI\(\)](#) (in module hexrd.xrd.xrutil), [62](#)  
[grainSpotsFromStack\(\)](#) (in module hexrd.xrd.xrutil), [63](#)  
[GrainSpotter](#) (class in hexrd.xrd.indexer), [40](#)  
[guessXVec\(\)](#) (hexrd.xrd.spotfinder.IntensityFuncGauss3DGenEll method), [46](#)  
[guessXVec\(\)](#) (hexrd.xrd.spotfinder.IntensityFuncMulti2D method), [47](#)  
[guessXVecPureNDImage\(\)](#) (hexrd.xrd.spotfinder.IntensityFuncMulti3D method), [46](#)

## H

[HallSymbol](#) (hexrd.xrd.spacegroup.SpaceGroup attribute), [45](#)



- hasImages (hexrd.xrd.experiment.ReaderInput attribute), 35
- haveXLabels() (hexrd.plotwrap.PlotWinLite method), 11
- haveXLabels() (hexrd.plotwrap.PlotWinP method), 11
- haveYLabels() (hexrd.plotwrap.PlotWinLite method), 11
- haveYLabels() (hexrd.plotwrap.PlotWinP method), 11
- hermannMauguin (hexrd.xrd.spacegroup.SpaceGroup attribute), 45
- hexagonalIndicesFromRhombohedral() (in module hexrd.xrd.crystallography), 19
- hexrd.arrayutil (module), 2
- hexrd.fileutil (module), 2
- hexrd.gridutil (module), 3
- hexrd.matrixutil (module), 4
- hexrd.orientations (module), 7
- hexrd.pfigutil (module), 10
- hexrd.plotwrap (module), 10
- hexrd.quadrature.q1db (module), 12
- hexrd.quadrature.q2db (module), 12
- hexrd.quadrature.q3db (module), 12
- hexrd.tens (module), 13
- hexrd.valunits (module), 15
- hexrd.xrd.crystallography (module), 16
- hexrd.xrd.detector (module), 20
- hexrd.xrd.distortion (module), 31
- hexrd.xrd.experiment (module), 31
- hexrd.xrd.fitting (module), 36
- hexrd.xrd.grain (module), 37
- hexrd.xrd.hydra (module), 39
- hexrd.xrd.indexer (module), 40
- hexrd.xrd.material (module), 41
- hexrd.xrd.rotations (module), 42
- hexrd.xrd.spacegroup (module), 44
- hexrd.xrd.spotfinder (module), 46
- hexrd.xrd.symmetry (module), 52
- hexrd.xrd.transforms (module), 52
- hexrd.xrd.transforms\_CAPI (module), 57
- hexrd.xrd.xrdbase (module), 59
- hexrd.xrd.xrutil (module), 60
- hist2D() (in module hexrd.plotwrap), 12
- histoFit() (in module hexrd.arrayutil), 2
- hkIMax (hexrd.xrd.material.Material attribute), 42
- hkls (hexrd.xrd.crystallography.PlaneData attribute), 17
- hkIToStr() (in module hexrd.xrd.crystallography), 17
- Hydra (class in hexrd.xrd.hydra), 39
- hydra (hexrd.xrd.experiment.Experiment attribute), 32
- I**
- iEtaBin (hexrd.xrd.xrutil.CollapseOmeEta attribute), 60
- ImageModes (class in hexrd.xrd.experiment), 31
- imageNames (hexrd.xrd.experiment.ReaderInput attribute), 35
- img\_list (hexrd.xrd.experiment.Experiment attribute), 33
- img\_names (hexrd.xrd.experiment.Experiment attribute), 33
- index\_opts (hexrd.xrd.experiment.Experiment attribute), 33
- IndexOptions (class in hexrd.xrd.experiment), 36
- indices (hexrd.xrd.xrutil.CollapseOmeEta attribute), 60
- indicesToMask() (hexrd.xrd.detector.ReadGE method), 22
- inputToGeomParams() (in module hexrd.xrd.fitting), 37
- IntensityFunc2D (class in hexrd.xrd.spotfinder), 46
- IntensityFunc3D (class in hexrd.xrd.spotfinder), 46
- IntensityFuncGauss2D (class in hexrd.xrd.spotfinder), 47
- IntensityFuncGauss3D (class in hexrd.xrd.spotfinder), 46
- IntensityFuncGauss3DGenEll (class in hexrd.xrd.spotfinder), 46
- IntensityFuncMulti2D (class in hexrd.xrd.spotfinder), 47
- IntensityFuncMulti3D (class in hexrd.xrd.spotfinder), 46
- invertQuat() (in module hexrd.xrd.rotations), 42
- invToM() (in module hexrd.orientations), 8
- invToQuat() (in module hexrd.orientations), 8
- invToRodr() (in module hexrd.orientations), 8
- isAngle() (hexrd.valunits.valWUnit method), 15
- isEnergy() (hexrd.valunits.valWUnit method), 15
- isinside() (in module hexrd.gridutil), 4
- isLength() (hexrd.valunits.valWUnit method), 15
- isMarked() (hexrd.xrd.spotfinder.Spot method), 48
- J**
- jQP (hexrd.xrd.detector.MultiRingEval attribute), 25
- K**
- KocksEuler (class in hexrd.orientations), 7
- kwargs (hexrd.xrd.experiment.PolarRebinOpts attribute), 36
- L**
- latticeParameters (hexrd.xrd.grain.Grain attribute), 38
- latticeParameters (hexrd.xrd.material.Material attribute), 42
- latticeParameters() (in module hexrd.xrd.crystallography), 17
- latticePlanes() (in module hexrd.xrd.crystallography), 17
- latticeToSampleT2() (in module hexrd.orientations), 9
- latticeToSampleV() (in module hexrd.orientations), 10
- latticeType (hexrd.xrd.spacegroup.SpaceGroup attribute), 45
- latticeVectors() (in module hexrd.xrd.crystallography), 18
- latVecOps (hexrd.xrd.crystallography.PlaneData attribute), 17
- laueGroup (hexrd.xrd.spacegroup.SpaceGroup attribute), 45
- LineStyles (class in hexrd.xrd.detector), 23
- listFiles() (in module hexrd.fileutil), 3

- loadDetector() (hexrd.xrd.experiment.Experiment method), 33
- loadExp() (in module hexrd.xrd.experiment), 36
- loadImages() (hexrd.xrd.hydra.Hydra method), 39
- loadMaterialList() (hexrd.xrd.experiment.Experiment method), 33
- loadMaterialList() (in module hexrd.xrd.material), 42
- loadRawSpots() (hexrd.xrd.experiment.Experiment method), 33
- loadReaderList() (hexrd.xrd.experiment.Experiment method), 33
- loadSpots() (hexrd.xrd.spotfinder.Spot static method), 48
- Log (class in hexrd.fileutil), 2
- logfile (hexrd.xrd.detector.MultiRingBinned attribute), 25
- ltypeOfLaueGroup() (in module hexrd.xrd.symmetry), 52
- ## M
- main() (in module hexrd.plotwrap), 12
- main() (in module hexrd.xrd.spotfinder), 51
- makeBinaryRotMat() (in module hexrd.xrd.transforms), 56
- makeBinaryRotMat() (in module hexrd.xrd.transforms\_CAPI), 59
- makeDetectorRotMat() (in module hexrd.xrd.transforms), 55
- makeDetectorRotMat() (in module hexrd.xrd.transforms\_CAPI), 59
- makeEtaFrameRotMat() (in module hexrd.xrd.transforms), 56
- makeEtaFrameRotMat() (in module hexrd.xrd.transforms\_CAPI), 59
- makeGVector() (in module hexrd.xrd.transforms), 52
- makeGVector() (in module hexrd.xrd.transforms\_CAPI), 57
- makeHist2D() (in module hexrd.plotwrap), 12
- makeIndicesTThRanges() (hexrd.xrd.detector.Detector2DRC method), 27
- makeMaskTThRanges() (hexrd.xrd.detector.Detector2DRC method), 27
- makeMeasuredScatteringVectors() (in module hexrd.xrd.xrutil), 62
- makeMNConn() (in module hexrd.xrd.xrutil), 62
- makeNew() (hexrd.xrd.detector.ReadGE method), 22
- makeNew() (hexrd.xrd.detector.ReadGeneric method), 21
- makeNVecs() (in module hexrd.xrd.xrutil), 62
- makeOscillRotMat() (in module hexrd.xrd.transforms), 56
- makeOscillRotMat() (in module hexrd.xrd.transforms\_CAPI), 59
- makePathVariantPoles() (in module hexrd.xrd.xrutil), 61
- makePlaneData() (hexrd.xrd.crystallography.PlaneData static method), 17
- makeQuatsBall() (in module hexrd.orientations), 10
- makeQuatsComponents() (in module hexrd.orientations), 10
- makeReader() (hexrd.xrd.experiment.ReaderInput method), 35
- makeRMatList() (in module hexrd.xrd.xrutil), 62
- makeRotMatOfExpMap() (in module hexrd.xrd.transforms), 56
- makeRotMatOfExpMap() (in module hexrd.xrd.transforms\_CAPI), 59
- makeRotMatOfQuat() (in module hexrd.xrd.transforms\_CAPI), 59
- makeScatteringVectors() (hexrd.xrd.crystallography.PlaneData static method), 17
- makeSynthFrames() (in module hexrd.xrd.xrutil), 64
- makeSynthSpots() (in module hexrd.xrd.xrutil), 61
- mapAngCen() (in module hexrd.xrd.xrutil), 64
- mapAngle() (in module hexrd.xrd.rotations), 44
- mapAngle() (in module hexrd.xrd.transforms), 55
- mapAngle() (in module hexrd.xrd.transforms\_CAPI), 59
- mapAngs() (in module hexrd.xrd.detector), 30
- mar165IDim() (in module hexrd.xrd.detector), 30
- matDict (hexrd.xrd.experiment.Experiment attribute), 33
- Material (class in hexrd.xrd.material), 42
- matList (hexrd.xrd.experiment.Experiment attribute), 33
- matNames (hexrd.xrd.experiment.Experiment attribute), 33
- matToCanova() (in module hexrd.orientations), 8
- matToQuat() (in module hexrd.orientations), 8
- matToThetaN() (in module hexrd.orientations), 9
- matxToSkew() (in module hexrd.tens), 13
- matxToSvec() (in module hexrd.tens), 14
- matxToSymm() (in module hexrd.tens), 13
- maxVal() (hexrd.xrd.detector.Framer2DRC class method), 21
- maxVal() (hexrd.xrd.detector.ReadGE class method), 22
- merge() (hexrd.xrd.spotfinder.Spot method), 48
- mergeSpotsOmegaStack() (hexrd.xrd.spotfinder.Spots static method), 50
- millerBravais2Normal() (in module hexrd.orientations), 10
- minimizeFiberDistance() (hexrd.xrd.grain.Grain method), 38
- misorAng() (hexrd.orientations.Quat method), 7
- misorientation() (in module hexrd.xrd.rotations), 43
- MultiRingBinned (class in hexrd.xrd.detector), 24
- MultiRingEval (class in hexrd.xrd.detector), 25
- MultiSlopeFunc (class in hexrd.xrd.xrutil), 61
- MultiSlopeFuncSmooth (class in hexrd.xrd.xrutil), 61
- multMatArray() (in module hexrd.matrixutil), 6
- MVCOBMatrix() (in module hexrd.tens), 13
- MVvecToSymm() (in module hexrd.tens), 13
- ## N
- n2eap() (in module hexrd.pfigutil), 10

n2sph() (in module hexrd.pfigutil), 10  
name (hexrd.xrd.material.Material attribute), 42  
newDetector() (hexrd.xrd.experiment.Experiment method), 33  
newDetector() (in module hexrd.xrd.detector), 30  
newGenericDetector() (in module hexrd.xrd.detector), 30  
newGenericReader() (in module hexrd.xrd.detector), 30  
newGrain() (hexrd.xrd.grain.Grain method), 39  
newMaterial() (hexrd.xrd.experiment.Experiment method), 33  
newName() (in module hexrd.xrd.experiment), 36  
newReader() (hexrd.xrd.experiment.Experiment method), 33  
normalize() (hexrd.orientations.Quat method), 8  
normalized() (hexrd.orientations.Quat method), 8  
normalized() (in module hexrd.matrixutil), 6  
normalizeQuat() (hexrd.orientations.Quat static method), 8  
NormalProjectionOfMV() (in module hexrd.tens), 14  
normvec() (in module hexrd.matrixutil), 6  
normvec3() (in module hexrd.matrixutil), 6  
nrmlProjOfVecMV() (in module hexrd.matrixutil), 5  
nullSpace() (in module hexrd.matrixutil), 4  
numFramesTotal (hexrd.xrd.experiment.Experiment attribute), 33

## O

objFun\_tVec\_d() (in module hexrd.xrd.xrutil), 64  
objFunc() (in module hexrd.xrd.experiment), 36  
objFuncFitGrain() (in module hexrd.xrd.fitting), 37  
objFuncSX() (in module hexrd.xrd.fitting), 37  
omeEtaGridToDA() (in module hexrd.xrd.xrutil), 63  
omeEtaGridToNVecs() (in module hexrd.xrd.xrutil), 62  
OmeEtaPfig (class in hexrd.xrd.xrutil), 60  
omeRangeToFrameRange() (in module hexrd.xrd.detector), 30  
orthogonalize() (in module hexrd.orientations), 8  
ownCanvas (hexrd.plotwrap.PlotWrap attribute), 11

## P

paintGrid() (in module hexrd.xrd.indexer), 40  
paintGridThis() (in module hexrd.xrd.indexer), 41  
Peak1DAAtLoc (class in hexrd.xrd.detector), 23  
PeakGauss1DAAtLoc (class in hexrd.xrd.detector), 24  
PeakLorentzian1DAAtLoc (class in hexrd.xrd.detector), 24  
PeakPV1DAAtLoc (class in hexrd.xrd.detector), 23  
pfigFromSpots() (in module hexrd.xrd.xrutil), 64  
pgRefine() (in module hexrd.xrd.indexer), 40  
pixelIndicesOfCartesianCoords() (hexrd.xrd.detector.Detector2DRC method), 27  
PlaneData (class in hexrd.xrd.crystallography), 16  
planeData (hexrd.xrd.material.Material attribute), 42  
plotByRingEta() (hexrd.xrd.detector.MultiRingBinned method), 25

plotHist2D() (in module hexrd.plotwrap), 12  
PlotWin (class in hexrd.plotwrap), 11  
PlotWinLite (class in hexrd.plotwrap), 11  
PlotWinP (class in hexrd.plotwrap), 11  
PlotWrap (class in hexrd.plotwrap), 11  
pointGroup (hexrd.xrd.spacegroup.SpaceGroup attribute), 45  
polarRebin() (hexrd.xrd.detector.Detector2DRC method), 28  
PolarRebinOpts (class in hexrd.xrd.experiment), 35  
pollImg (hexrd.xrd.detector.MultiRingBinned attribute), 25  
prbkw (hexrd.xrd.detector.MultiRingBinned attribute), 25  
printTestName() (in module hexrd.xrd.rotations), 44  
processWavelength() (in module hexrd.xrd.crystallography), 17  
pullFromStack() (in module hexrd.xrd.xrutil), 63  
pullSpots() (in module hexrd.xrd.xrutil), 65  
putInShelf() (in module hexrd.xrd.xrutil), 63  
pwList (hexrd.plotwrap.PlotWinP attribute), 11

## Q

qLoc() (in module hexrd.quadrature.q1db), 12  
qLoc() (in module hexrd.quadrature.q2db), 12  
qLoc() (in module hexrd.quadrature.q3db), 13  
qloc1() (in module hexrd.quadrature.q1db), 12  
qloc1() (in module hexrd.quadrature.q2db), 12  
qloc1() (in module hexrd.quadrature.q3db), 12  
qloc2() (in module hexrd.quadrature.q1db), 12  
qloc27() (in module hexrd.quadrature.q3db), 12  
qloc3() (in module hexrd.quadrature.q1db), 12  
qloc4() (in module hexrd.quadrature.q1db), 12  
qloc4() (in module hexrd.quadrature.q2db), 12  
qloc5() (in module hexrd.quadrature.q1db), 12  
qloc8() (in module hexrd.quadrature.q1db), 12  
qloc8() (in module hexrd.quadrature.q3db), 12  
qloc9() (in module hexrd.quadrature.q2db), 12  
qLocFrom1D() (in module hexrd.quadrature.q2db), 12  
qLocFrom1D() (in module hexrd.quadrature.q3db), 12  
Quat (class in hexrd.orientations), 7  
quat\_distance() (in module hexrd.xrd.transforms), 56  
quat\_distance() (in module hexrd.xrd.transforms\_CAPI), 59  
quat\_product\_matrix() (in module hexrd.xrd.transforms), 56  
quatOfAngleAxis() (in module hexrd.xrd.rotations), 43  
quatOfExpMap() (in module hexrd.xrd.rotations), 43  
quatOfLaueGroup() (in module hexrd.xrd.symmetry), 52  
quatOfRotMat() (in module hexrd.xrd.rotations), 43  
quatProduct() (in module hexrd.xrd.rotations), 43  
quatProductMatrix() (in module hexrd.xrd.rotations), 43  
quatToInv() (in module hexrd.orientations), 9  
quatToMat() (in module hexrd.orientations), 9  
quatToProdMat() (in module hexrd.orientations), 9



## R

- radialDistortion() (hexrd.xrd.detector.DetectorGeomFrelon method), 29
- radialDistortion() (hexrd.xrd.detector.DetectorGeomGE method), 28
- radialDistortion() (hexrd.xrd.detector.DetectorGeomMar165 method), 28
- radialFitXVec() (hexrd.xrd.detector.MultiRingEval method), 25
- radialPlotData() (hexrd.xrd.detector.MultiRingEval method), 26
- rankOneMatrix() (in module hexrd.matrixutil), 5
- raw\_spots (hexrd.xrd.experiment.Experiment attribute), 33
- rawRead() (hexrd.xrd.detector.ReadGE method), 22
- RC (hexrd.xrd.experiment.ReaderInput attribute), 35
- read() (hexrd.xrd.detector.Framer2DRC method), 21
- read() (hexrd.xrd.detector.ReadGE method), 22
- read() (hexrd.xrd.detector.ReadGeneric method), 21
- readBBox() (hexrd.xrd.detector.ReadGE method), 22
- readDark() (hexrd.xrd.detector.ReadGE class method), 22
- readDataFlat() (in module hexrd.fileutil), 3
- ReaderInput (class in hexrd.xrd.experiment), 34
- readerListAddCurrent() (hexrd.xrd.experiment.Experiment method), 33
- readerNames (hexrd.xrd.experiment.Experiment attribute), 34
- readFloatData() (in module hexrd.fileutil), 3
- readFloatDataA() (in module hexrd.fileutil), 2
- readFLT() (in module hexrd.fileutil), 3
- readFrameStack\_multiproc() (in module hexrd.xrd.xrdutil), 62
- ReadGE (class in hexrd.xrd.detector), 21
- ReadGeneric (class in hexrd.xrd.detector), 21
- readImage() (hexrd.xrd.experiment.Experiment method), 33
- ReadMar165 (class in hexrd.xrd.detector), 23
- ReadMar165NB1 (class in hexrd.xrd.detector), 23
- ReadMar165NB2 (class in hexrd.xrd.detector), 23
- ReadMar165NB3 (class in hexrd.xrd.detector), 23
- ReadMar165NB4 (class in hexrd.xrd.detector), 23
- readRaw() (hexrd.xrd.detector.ReadGE class method), 22
- referenceLatticeParameters (hexrd.xrd.grain.Grain attribute), 39
- refine\_grains() (hexrd.xrd.experiment.Experiment method), 34
- refineDetector() (in module hexrd.xrd.experiment), 36
- refineFlags (hexrd.xrd.experiment.Experiment attribute), 34
- reg\_grid\_indices() (in module hexrd.xrd.transforms), 55
- renderEAProj() (in module hexrd.pfigutil), 10
- renderIdeal() (hexrd.xrd.detector.Detector2DRC method), 28
- reqParams (hexrd.xrd.spacegroup.SpaceGroup attribute), 45
- resetDetectorGeom() (hexrd.xrd.spotfinder.Spots method), 50
- resolveWild() (in module hexrd.fileutil), 2
- revertOutputDegrees() (in module hexrd.xrd.crystallography), 17
- rhombohedralIndicesFromHexagonal() (in module hexrd.xrd.crystallography), 19
- rhombohedralParametersFromHexagonal() (in module hexrd.xrd.crystallography), 19
- rhoPxRange (hexrd.xrd.detector.MultiRingBinned attribute), 25
- rmDirF() (in module hexrd.fileutil), 3
- rmSafe() (in module hexrd.fileutil), 3
- rmWild() (in module hexrd.fileutil), 3
- rmWorkDir() (in module hexrd.fileutil), 3
- rodrToInv() (in module hexrd.orientations), 8
- rodrToQuat() (in module hexrd.orientations), 8
- rotate\_vecs\_about\_axis() (in module hexrd.xrd.transforms), 56
- rotate\_vecs\_about\_axis() (in module hexrd.xrd.transforms\_CAPI), 59
- RotationParameterization (class in hexrd.orientations), 7
- RotInv (class in hexrd.orientations), 7
- rotMatOfExpMap\_opt() (in module hexrd.xrd.rotations), 43
- rotMatOfExpMap\_orig() (in module hexrd.xrd.rotations), 43
- rotMatOfQuat() (in module hexrd.xrd.rotations), 43
- rotMatrixFromCrystalVectors() (in module hexrd.orientations), 10
- roty90() (in module hexrd.xrd.xrdutil), 62
- rowNorm() (in module hexrd.matrixutil), 4
- rowNorm() (in module hexrd.xrd.transforms), 55
- rowNorm() (in module hexrd.xrd.transforms\_CAPI), 59
- run\_indexer() (hexrd.xrd.experiment.Experiment method), 34

## S

- sampleToLatticeT2() (in module hexrd.orientations), 9
- save() (hexrd.plotwrap.PlotWrap method), 11
- saveDetector() (hexrd.xrd.experiment.Experiment method), 34
- savedReaders (hexrd.xrd.experiment.Experiment attribute), 34
- saveExp() (in module hexrd.xrd.experiment), 36
- saveRawSpots() (hexrd.xrd.experiment.Experiment method), 34
- saveReaderList() (hexrd.xrd.experiment.Experiment method), 34
- saveRMats() (hexrd.xrd.experiment.Experiment method), 34
- set\_pVec() (hexrd.xrd.grain.Grain method), 39

- setCentersFromRef() (hexrd.xrd.detector.DetectorGeomQuadGen method), 30  
 setFuncXVecList() (hexrd.xrd.detector.MultiRingEval method), 26  
 setOmegaInfo() (hexrd.xrd.experiment.ReaderInput method), 35  
 setQuadOffsets() (hexrd.xrd.detector.DetectorGeomQuadGen method), 30  
 setStretchTensor() (hexrd.xrd.grain.Grain method), 39  
 setupMulti() (hexrd.xrd.spotfinder.Spot method), 48  
 setupQuardFromFWHM() (hexrd.xrd.spotfinder.Spot method), 48  
 sgnum (hexrd.xrd.material.Material attribute), 42  
 sgnum (hexrd.xrd.spacegroup.SpaceGroup attribute), 45  
 SgOps (hexrd.xrd.spacegroup.SpaceGroup attribute), 45  
 simulateGVecs() (in module hexrd.xrd.xrutil), 65  
 sixLatticeParams() (hexrd.xrd.spacegroup.SpaceGroup method), 45  
 skew() (in module hexrd.matrixutil), 5  
 skewMatrixOfVector() (in module hexrd.matrixutil), 5  
 skewOfMatx() (in module hexrd.tens), 13  
 SpaceGroup (class in hexrd.xrd.spacegroup), 45  
 spaceGroup (hexrd.xrd.material.Material attribute), 42  
 sph2n() (in module hexrd.pfigutil), 10  
 Spot (class in hexrd.xrd.spotfinder), 47  
 spot\_readers (hexrd.xrd.experiment.Experiment attribute), 34  
 spotFinderSingle() (in module hexrd.xrd.spotfinder), 51  
 spotFromDataList() (hexrd.xrd.spotfinder.Spot static method), 48  
 spotFromStack() (in module hexrd.xrd.xrutil), 63  
 SpotOptions (class in hexrd.xrd.experiment), 36  
 spotOpts (hexrd.xrd.experiment.Experiment attribute), 34  
 Spots (class in hexrd.xrd.spotfinder), 49  
 spots\_for\_indexing (hexrd.xrd.experiment.Experiment attribute), 34  
 SpotsIterator (class in hexrd.xrd.spotfinder), 50  
 storeSpots() (hexrd.xrd.spotfinder.Spot static method), 49  
 stretchToLV() (in module hexrd.xrd.xrutil), 62  
 strip() (hexrd.xrd.grain.Grain method), 39  
 stripQuatList() (in module hexrd.orientations), 10  
 structuredSort() (in module hexrd.arrayutil), 2  
 subtractDark (hexrd.xrd.detector.ReadGeneric attribute), 21  
 sutherlandHodgman() (in module hexrd.gridutil), 4  
 svecpToSvec() (in module hexrd.tens), 14  
 svecToMatx() (in module hexrd.tens), 14  
 svecToSvecP() (in module hexrd.tens), 14  
 svecToSymm() (in module hexrd.tens), 14  
 svecToVecds() (in module hexrd.tens), 14  
 symm() (in module hexrd.matrixutil), 5  
 SymmGroup (class in hexrd.orientations), 8  
 symmOfMatx() (in module hexrd.tens), 13  
 symmPlusI() (in module hexrd.tens), 14  
 symmToMVvec() (in module hexrd.tens), 13  
 symmToSvec() (in module hexrd.tens), 14  
 symmToVecds() (in module hexrd.tens), 13  
 symmToVecMV() (in module hexrd.matrixutil), 4
- ## T
- T() (hexrd.orientations.Quat method), 7  
 T2Svec (class in hexrd.tens), 13  
 T2SvecP (class in hexrd.tens), 13  
 T2Symm (class in hexrd.tens), 13  
 T2Vecds (class in hexrd.tens), 13  
 tempSetOutputDegrees() (in module hexrd.xrd.crystallography), 17  
 testHKLs() (in module hexrd.xrd.spacegroup), 46  
 testRotMatOfExpMap() (in module hexrd.xrd.rotations), 44  
 testSingle() (in module hexrd.xrd.spotfinder), 51  
 testSpotFinder() (in module hexrd.xrd.spotfinder), 51  
 testThisQ() (in module hexrd.xrd.indexer), 40  
 textureToSpots() (in module hexrd.xrd.xrutil), 61  
 ThreadReadFrame (class in hexrd.xrd.detector), 20  
 thresholdStackDisplay() (in module hexrd.xrd.xrutil), 62  
 ticMethod (hexrd.xrd.detector.MultiRingBinned attribute), 25  
 toArray() (in module hexrd.arrayutil), 2  
 toBunge() (hexrd.orientations.KocksEuler method), 7  
 toFloat() (in module hexrd.valunits), 15  
 toFundamentalRegion() (in module hexrd.xrd.symmetry), 52  
 toKocks() (hexrd.orientations.BungeEuler method), 7  
 toMatrix() (hexrd.orientations.RotationParameterization method), 7  
 trace3() (in module hexrd.tens), 13  
 traceToAng() (in module hexrd.orientations), 8  
 traceToVecdsS() (in module hexrd.tens), 13  
 transpose() (hexrd.orientations.Quat method), 8  
 transposed() (hexrd.orientations.Quat method), 8  
 transposeQuats() (in module hexrd.orientations), 10  
 tryFromShelf() (in module hexrd.xrd.xrutil), 63  
 tThRanges (hexrd.xrd.xrutil.CollapseOmeEta attribute), 60  
 tVec\_d\_from\_old\_parfile() (in module hexrd.xrd.xrutil), 64
- ## U
- uMat (hexrd.xrd.grain.Grain attribute), 39  
 UNames (class in hexrd.valunits), 15  
 UnfitableError (class in hexrd.xrd.spotfinder), 47  
 uniqueVectors() (in module hexrd.matrixutil), 6  
 unitRowVector() (in module hexrd.transforms\_CAPI), 59  
 unitVector() (in module hexrd.matrixutil), 4  
 unitVector() (in module hexrd.xrd.transforms), 55  
 updateGVecs() (hexrd.xrd.grain.Grain method), 39

useThreading (hexrd.xrd.detector.ReadGE attribute), 22

## V

validateAngleRanges() (in module hexrd.xrd.transforms), 56

validateAngleRanges() (in module hexrd.xrd.transforms\_CAPI), 59

validateAngleRanges() (in module hexrd.xrd.xrutil), 64

validateQVecAngles() (in module hexrd.xrd.xrutil), 65

valWithDflt() (in module hexrd.valunits), 16

valWUnit (class in hexrd.valunits), 15

vecdsSToTrace() (in module hexrd.tens), 13

vecdsToSymm() (in module hexrd.tens), 13

vecdvToVecds() (in module hexrd.tens), 13

vecMVCOBMatrix() (in module hexrd.matrixutil), 5

vecMVToSymm() (in module hexrd.matrixutil), 4

vectorOfSkewMatrix() (in module hexrd.matrixutil), 6

vMat (hexrd.xrd.grain.Grain attribute), 39

vol (hexrd.xrd.grain.Grain attribute), 39

## W

wQP (hexrd.xrd.detector.MultiRingBinned attribute), 25

write\_old\_parfile() (in module hexrd.xrd.xrutil), 64

writeArray() (in module hexrd.arrayutil), 2

writeGVE() (in module hexrd.xrd.indexer), 41

writeQuats() (in module hexrd.orientations), 10

## X

xyoCOM() (hexrd.xrd.spotfinder.Spot method), 49

xyoToAng() (hexrd.xrd.detector.Detector2DRC method), 28

xyoToAng\_V() (hexrd.xrd.detector.Detector2DRC method), 28

xyoToAngAll() (hexrd.xrd.detector.Detector2DRC method), 28

xyoToAngCorners() (hexrd.xrd.detector.Detector2DRC method), 28

xyoToAngMap() (hexrd.xrd.detector.Detector2DRC method), 28