Appendix C

Analysis example in Python

C.1 Using the LOKI library

I developed a modular Python package called LOKI that can be downloaded at

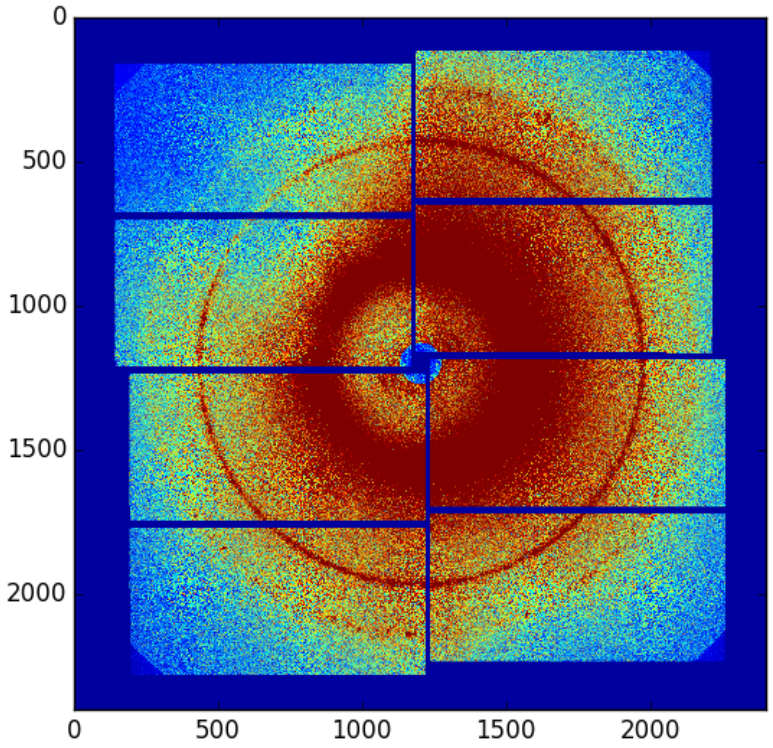
https://github.com/dermen/loki

and can help in analysis of 2D X-ray data. It is installed by running the usual "setup.py install" command with the Python interpreter. This simple tutorial describes how to interpolate a cartesian image using LOKI such that correlations can then be computed. In the root directory of the LOKI Github repository there is a file in called

test_img_and_mask.hdf5

which contain a test image and image mask that will be used below.

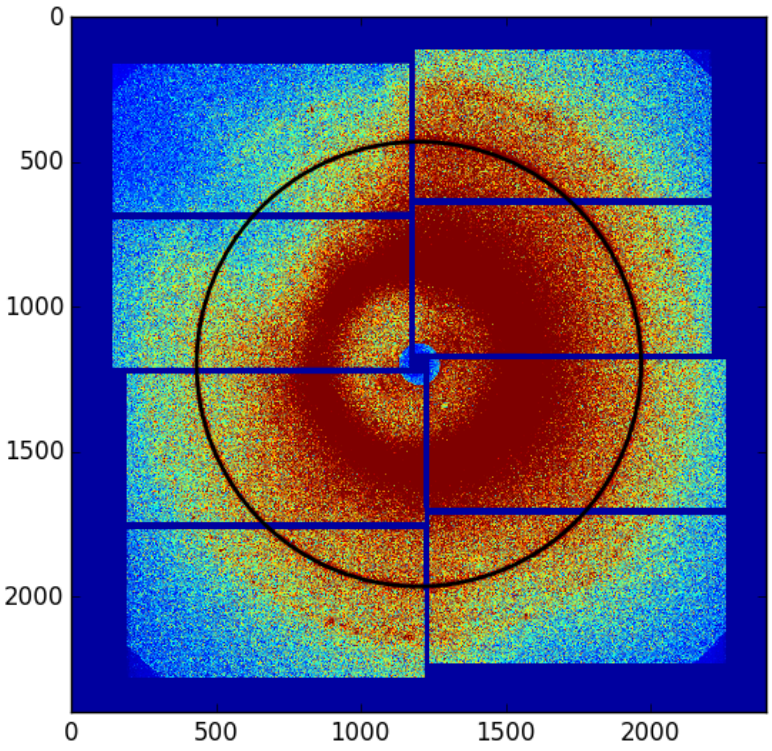
```
# first load and display the image
from pylab import *
import h5py
h5_f = h5py.File('test_img_and_mask.hdf5', 'r')
img = h5_f['img'].value.astype(float32)
imshow(img, vmax = 1000)
show()
```



This shows a nice ring pattern. It is an image of gold NP scattering recorded at the SACLA beamline in Japan. Using the pylab GUI, one can guess the radial position of the 111 Bragg ring (measured from the center) to be 765 pixel units. A good guess for the detector origin of a symmetric diffraction image like this one is just the center of the detector, however we can optimize the position using LOKI :

```
# initial guess for the detector origin
x_i  = img.shape[1]/2. # fast dimension of image
y_i = img.shape[0]/2. # slow dimension of image
q_i = 765. # 111 Bragg ring

# import the RingFit module from Loki
from loki.RingData import RingFit
```

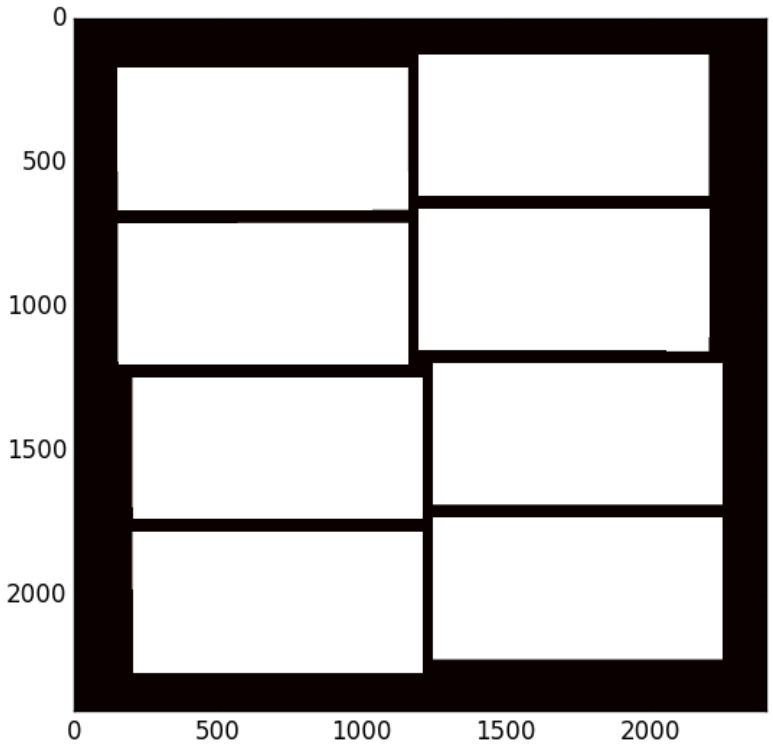
```
RF = RingFit( img )
(x_center, y_center, q_ring), _ , _ = RF.fit_circle( beta_i=( x_i ,y_i , q_i ) )

# plot the result
imshow(img, vmax = 1000)
gca().add_patch(Circle( xy=(x_center,y_center), radius=q_ring , ec = 'k', fc = 'none', lw=
    2,) )
show()
```



Now, load a user defined mask file which masks the beamstop and the gaps between panels. For LOKI, masks are two dimensional boolean arrays that have the same shape as the images. In a masked array, the value "1" is means the corresponding pixel is valid, and a "0" mean it is masked,

meaning it won't be included in computations:

```
mask = h5_f['mask']

imshow(mask, cmap='hot')

show()
```



In the following snippet we show how to produce a polar image in LOKI using a simple nearest neighbor interpolation

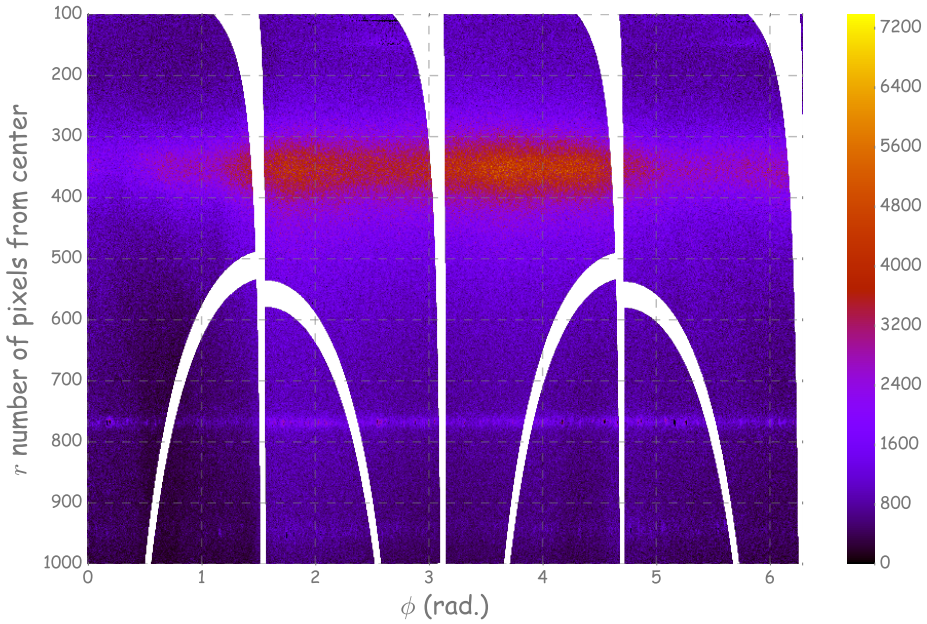
```
from loki.RingData import InterpSimple


qRmax = 1000

qRmin = 100
```

```
nphi  =  int( 2 * pi * qRmax ) # single pixel resolution at maximum q

# initialize the interpolater
interpolater = InterpSimple( x_center, y_center, qRmax=qRmax, qRmin=qRmin, nphi=nphi,
    raw_img_shape = mask.shape )

# make a polar image mask
pmask  = interpolater.nearest( mask , dtype=bool ).round()

# Make the masked polar image
polar_img = pmask * interpolater.nearest( img )
```

Let's have a look at the gold Bragg ring image in polar coordinates:

```
# let's first remove high pixel values
from loki.utils.postproc_helper import is_outlier

# lets make a masked array object for convenience
polar_img_m = np.ma.masked_equal( polar_img, 0)

# remove pixels that are 5 median-standard deviations away from median
for i_ring, ring in enumerate(polar_img_m): # iterate over rings
    bad_pix = is_outlier(ring, 5) # bad pixels in ring
    polar_img_m[i_ring, bad_pix] = 0 # make them 0

# display the masked image
# note how we do not have to specify vmax as we removed outliers that would otherwise skew
    the color map
imshow( polar_img_m, aspect='auto' , extent=(0,2*pi,qRmax, qRmin), cmap='gnuplot' )
colorbar()
ylabel(r'$r$ number of pixels from center',fontsize=18)
xlabel(r'$\phi$ (rad.)',fontsize=18)
```

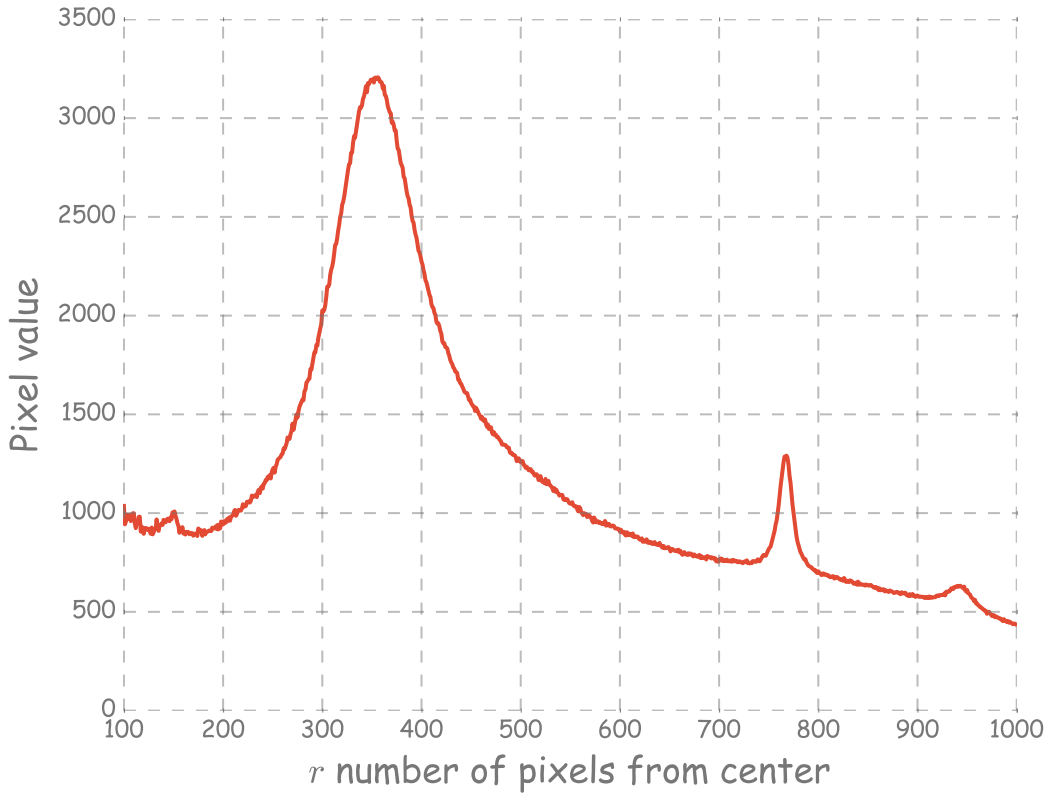
```
show()
```



Note how the 111 Bragg ring appears as a line in polar coordinates. Now, to correlate this image, we use the DiffCorr class from LOKI. Usually, DiffCorr takes a list of exposure differences with mean 0, and this is important (see section B.5.9), but in this case we only have a single image. Nevertheless we can make a quick hack to show how to use the DiffCorr package:

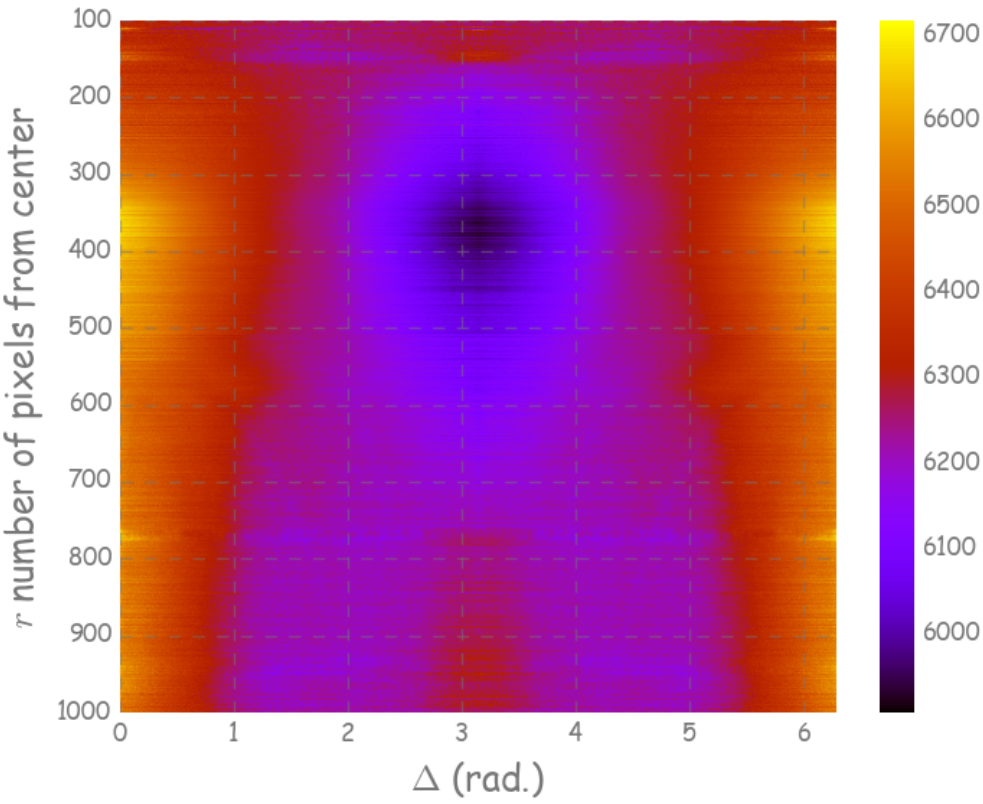
```
# The LOKI DiffCorr class does not use a mask, as it
# typically works with difference intensities which have average value 0
# hence the masked values are automatically filled in with the average value
# Since, however, we are not doing a difference correlation, we have to fill in the masked
    regions
# with some average pixel value.
# We use radial profile to fill in the masked pixels with an average value
rad_pro = polar_img_m.mean(1)
plot( np.arange( qRmin, qRmax), rad_pro, lw=2)
```

```
xlabel(r'$r$ number of pixels from center',fontsize=18)
ylabel('Pixel value', fontsize=18)
# The 111 and 200 Bragg reflections are visible at around r=750 and r=950, respectively.
# (note the solvent scatter peak around 350 pixels)
```




```
# now we use radial profile to fill in the polar_img masked regions
fill_in = np.logical_not(pmask) * rad_pro[:,None]
polar_img_filled = fill_in.data + polar_img_m.data

# now we can do the correlation using DiffCorr
from loki.RingData import DiffCorr
dc = DiffCorr([polar_img_filled])
```

```
cor = dc.autocorr()[0] # returns a list of auto-correlations, in this case we only have one

# normalize to account for intensity variation with radius
cor /= (rad_pro[:,None]**2) # [:,None] is called broadcasting in numpy

# plot the correlation, trim the edges near Delta=0 and 2*pi to avoid skewing the color
    scheme
imshow( cor[:,10:-10], aspect='auto' , extent=(0,2*pi,qRmax, qRmin), cmap='gnuplot')
xlabel(r'$\Delta$ (rad.)',fontsize=18)
ylabel(r'$r$ number of pixels from center',fontsize=18)
colorbar()
show()
```

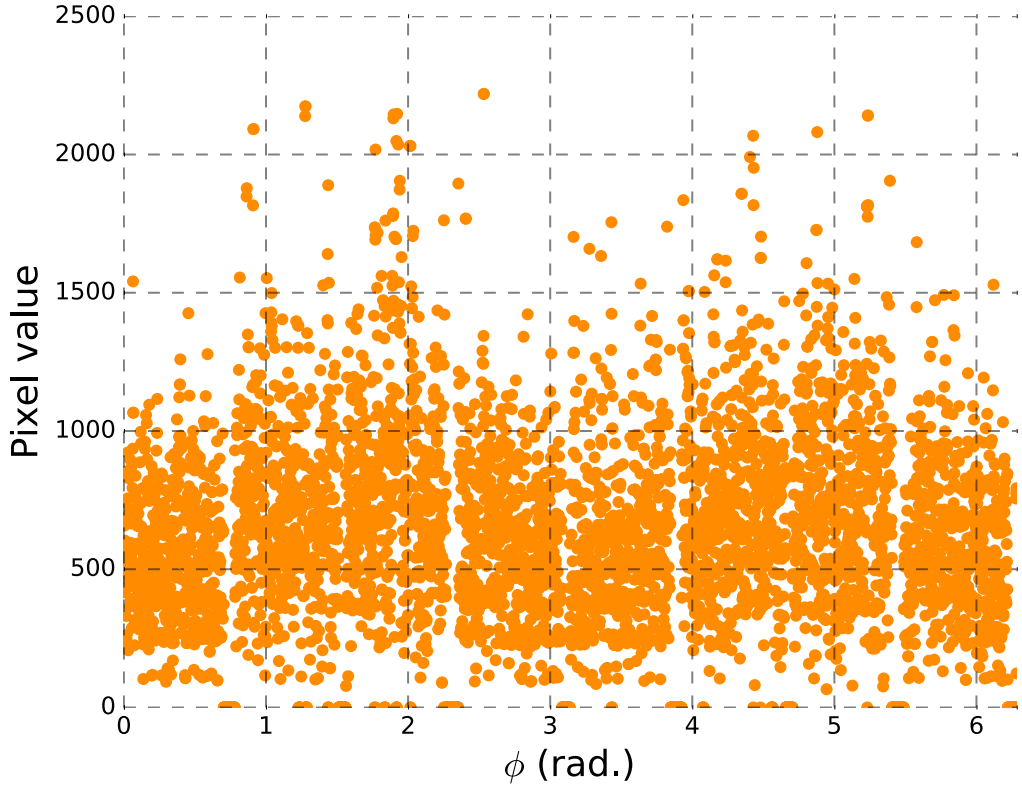


As evident in the above figure, artifactual correlations dominate this single shot auto-correlation, which should otherwise be relatively noisy and isotropic, as it just represents a single snapshot. This is why we use difference correlations - to suppress these artifactual correlations. In what follows we will demonstrate how to use difference correlations to suppress these anisotropies. You will need to use the file

bragg_rings.hdf5

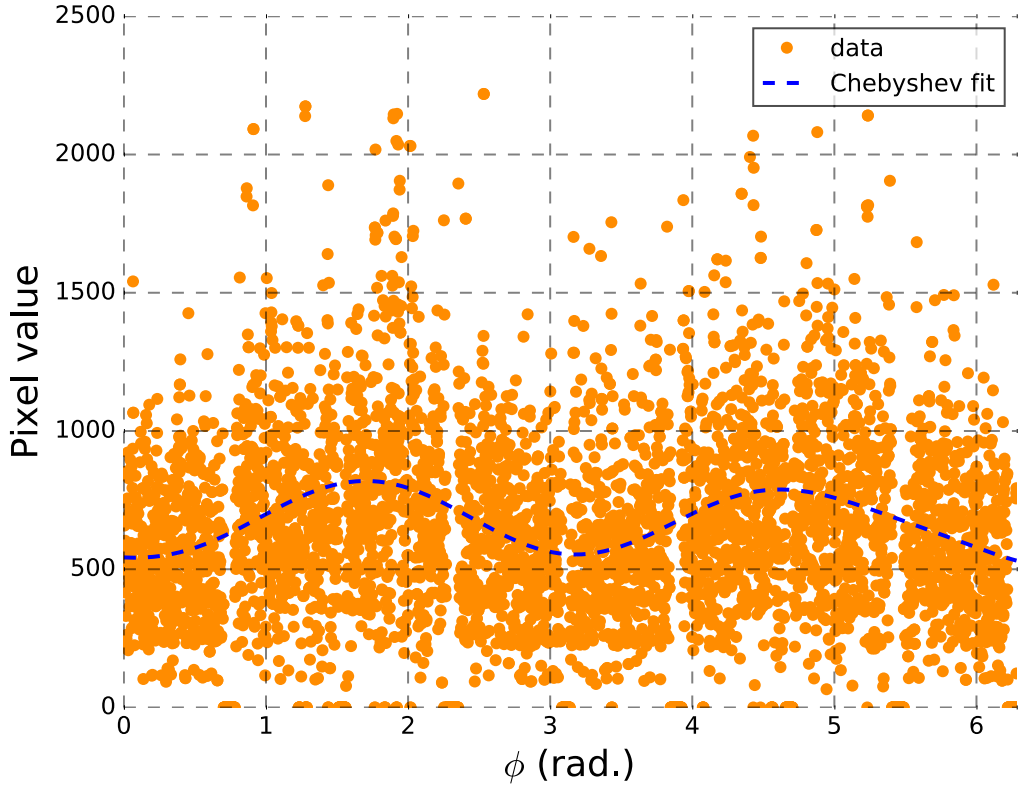
found in the root of the LOKI Github repository.

```
#to begin, load the test Bragg rings
# NOTE: Crucially, these Bragg rings have all been set to the same overall
# intensity scale, so that they can eventually be subtracted from one another.
# This is an important step in the analysis of XFEL data where
# shot-to-shot intensity fluctuates significantly
rings = h5py.File('bragg_rings.hdf5','r')['111_Bragg_rings'].value.astype(float32)
print (rings.shape)
# (14, 5000)

# There are 14 Bragg rings here that all occurred on the same
# pixel radius from the detector origin, around r=750 pixels
# Each Bragg ring is a set of 5000 interpolated pixel values
# that spans the bragg ring from 0 to 2*pi

phis = np.arange( rings.shape[1]) * 2 * pi / rings.shape[1] #5000 phi values

plot( phis, rings[0], 'o', mec='none', color='Darkorange')
xlabel(r'$\phi$ (rad.)', fontsize=18)
ylabel(r'Pixel value', fontsize=18)
xlim(0,2*pi)
show()
```



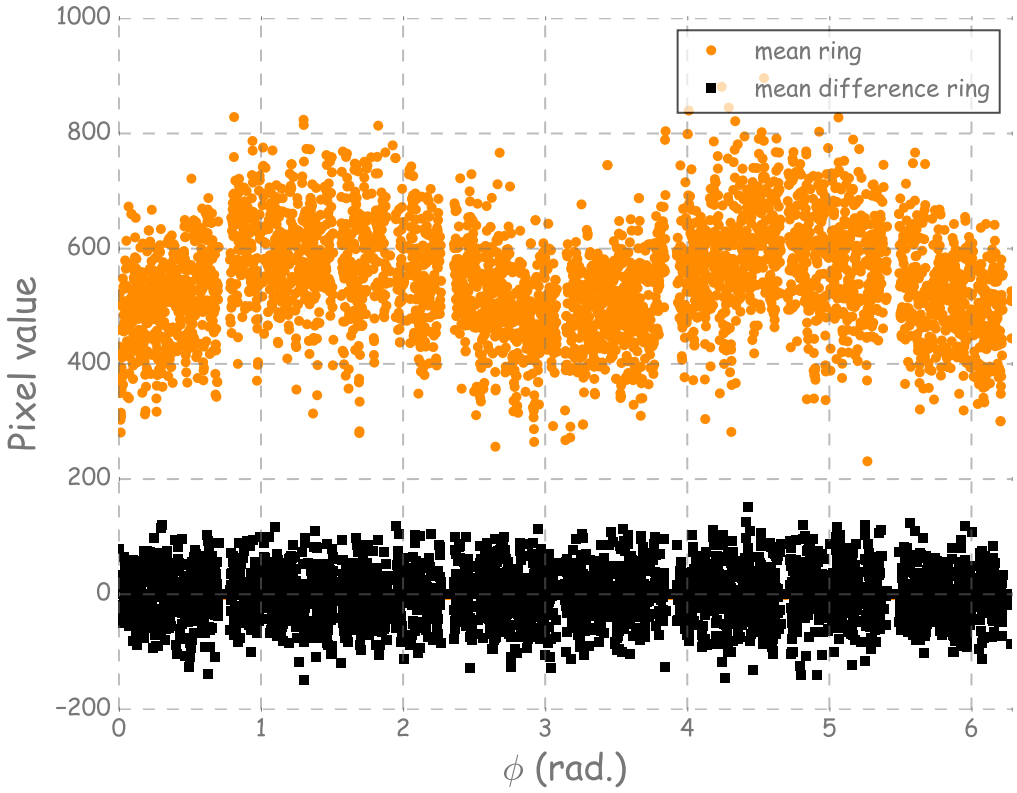

```
# Our goal is to pair exposures in a way that minimizes artifactual azimuthal
    anisotropies, which are apparent in the above Bragg ring plot
# We can quantify the anisotropy by fitting a low-order polynomial to the Bragg ring
    profile, using LOKI

from loki.utils.postproc_helper import fit_periodic

# this function takes a mask. Fortunately, these Bragg rings are implicitly masked, 0
    values are masked values
# lets fit a Chebyshev polynomial to each Bragg ring

coefs = []
```

```
for r in rings:
    mask = r > 0
    cheby_coef, _, _ = fit_periodic(r, mask, 15)
    coefs.append( cheby_coef)

# the domain of the polynomial is actually just the azimuthal indices
poly_domain = np.arange( rings.shape[1])
cheby_fits = np.array( [ np.polynomial.chebyshev.chebval( poly_domain, c) for c in coefs ]
    )

# With these polynomials we can quantify the azimuthal anisotropy
plot( phis, rings[0] , 'o', color='Darkorange', mec='none', label='data')
plot( phis, cheby_fits[0], '--', lw=2, color='b', label='Chebyshev fit')
xlabel(r'$\phi$ (rad.)', fontsize=18)
ylabel(r'Pixel value', fontsize=18)
xlim(0,2*pi)
legend()
show()
```



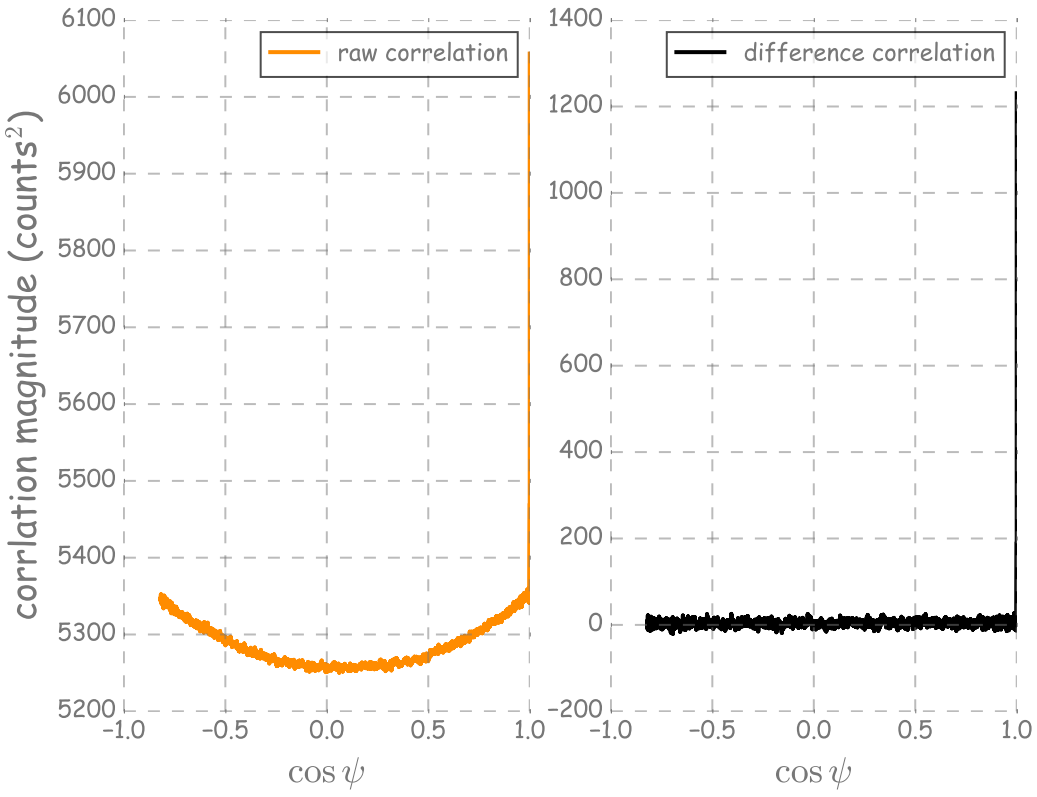

```
# now that we have polynomial fits, we can use LOKI to pair the Bragg rings according to
    their fits
from loki.utils.postproc_helper import pair_fits

pairs, _ = pair_fits(cheby_fits)
#----Performing distance calc...
#----Pairing...

print (pairs)
# [[0, 4], [1, 6], [2, 5], [3, 8], [7, 11], [9, 13], [10, 12]]
#what this is saying is that ring 0 matches closest to ring 4, ring 1 matches closest to
    ring 6, and so forth
```

```
# Before continuing, lets use LOKI to remove bright Bragg peaks from the large crystal
    reflections
from loki.utils.postproc_helper import remove_peaks
from itertools import izip
for i_r, ( ring, coef) in enumerate( izip( rings, coefs )):
    mask = r > 0
    new_ring, _, _ = remove_peaks( ring, mask, thick=2, peak_thresh=2.5, coef=coef)
    rings[i_r] = new_ring

# Now that we have a pair mapping that groups rings according to similar anisotropy, we
    can compute the difference correlations
diffs = np.zeros( (len(pairs), rings.shape[1]))
for i_diff, (i,j) in enumerate( pairs):
    r1 = rings[i] # data
    r2 = rings[j]
    m1 = r1 > 0 # masks
    m2 = r2 > 0

    m_tot = m1 * m2 # total mask
    diffs[i_diff] = (r1 - r2)*m_tot

plot( phis, rings.mean(0), 'o',color='Darkorange', mec='none', label='mean ring', ms=5)
plot( phis, diffs.mean(0), 's', color='k', mec='none', label='mean difference ring', ms=5)
xlabel(r'$\phi$ (rad.)', fontsize=18)
ylabel(r'Pixel value', fontsize=18)
xlim(0,2*pi)
legend()
show()
```




```
# now we can use DiffCorr class from LOKI to correlate the difference rings

from loki.RingData import DiffCorr # if not already imported
dc = DiffCorr(diffs)
cors = dc.autocorr()

# lets use the PSI representation of the angle
# We need the wavelength of the photons, 1.42 angstrom, as well as the momentum transfer
    magnitude of the gold 111 Bragg ring, 2.67 inverse angstrom
theta_111 = arcsin( 2.67 * 1.42 / 4 / pi ) # this is half of the Bragg angle
cos_psi = np.cos( phis) * np.cos(theta_111)**2 + np.sin(theta_111) **2
```

```
# the average 111 correlation
cm = cors.mean(0)

# lets normalize by average 111 intensity
norm = ma.masked_equal( rings,0).mean()
cm /= norm**2

# Lets compare this with the raw correlation of Bragg rings
# As before, in order to use DiffCorr to compute a non-difference correlation, we will
    need to fill in the masked values with the mean
# Let's use the polynomial fits to fill in the gaps..

fill_ins = []
for ring ,fit in izip( rings, cheby_fits):
    gaps = (ring == 0)* fit
    fill_ins.append( ring + gaps)

dc2 = DiffCorr(fill_ins)
cors2 = dc2.autocorr()
cm2 = cors2.mean(0)
cm2 /= norm**2

subplot(122)
plot( cos_psi, cm, 'k' , label='difference correlation')
xlabel(r'$\cos \,\psi$', fontsize=18)
legend()
subplot(121)
plot( cos_psi, cm2, 'Darkorange' , label='raw correlation')
xlabel(r'$\cos \,\psi$', fontsize=18)
ylabel(r'corrlation magnitude (counts$^2$)', fontsize=18)
legend()
show()
```



With only 14 exposures, we do not expect significant CXS signal, just an isotropic noise, however the raw correlations, $C(\cos\psi)$ show a dominant artifactual signal. We are able to suppress this signal by making use of the difference correlation $D(\cos\psi)$.