Diffraction_image_visualization

January 14, 2016

1 Diffraction image visualization

This exercise is about single crystal diffraction images (protein crystallography) and highlights difficulties one can encounter when trying to display such image.

The image we will try to display, run2_1_00147.cbf, is part of the insulin test-set provided by the manufacturer (Dectris) to advertise the quality of its Pilatis 6M detectors.

1.1 Initialization of the scientific Python stack

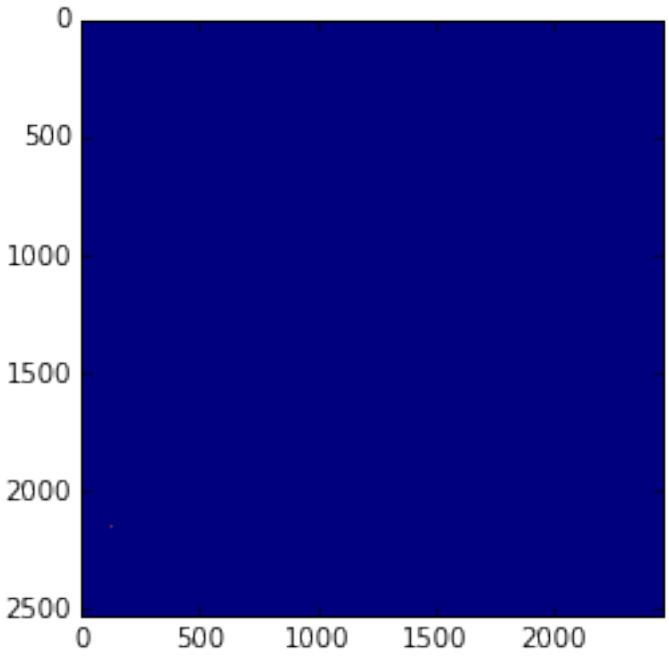
The first difficulty one may encounter is the compressed form of this image. Fortunately, both PyMca and FabIO support the CIF Binary Format and the byte-offset compression scheme used by Dectris.

```
In [1]: %pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

```
In [2]: import fabio
        img = fabio.open("run2_1_00148.cbf")
        imshow(img.data)
```

```
Out[2]: <matplotlib.image.AxesImage at 0x7f7d0c897550>
```



When trying to display such image, it appears empty. One has to check the content to be sure it is not:

```
In [3]: print("shape: (%s, %s)"% img.data.shape)
        print("max: %s"% img.data.max())
        print("min: %s"% img.data.min())
        print("mean: %s"% img.data.mean())
        print("std: %s"% img.data.std())

shape: (2527, 2463)
max: 664762
min: -2
mean: 6.4045405841
std: 473.992411656
```

Diffraction data exhibit a very large dynamical range, here from 0 to 7e5 but most datapoint are arround 0. One solution, at least, one part of the solution is to display the image in a logarithmic color-scale.

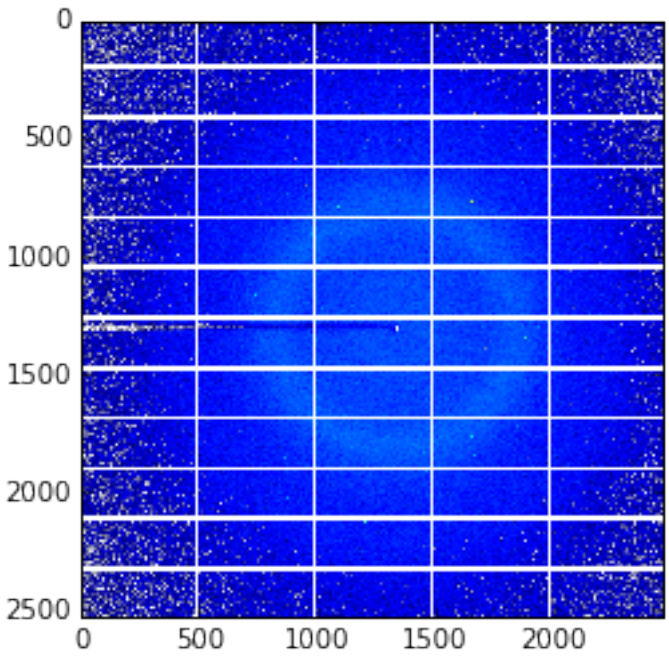
1.2 Logarithmic colors

Simply display the logarithm of the intensity (intensities <=0 will raise a warning and be transparent)

```
In [4]: imshow(np.log(img.data))

-c:1: RuntimeWarning: divide by zero encountered in log
-c:1: RuntimeWarning: invalid value encountered in log

Out[4]: <matplotlib.image.AxesImage at 0x7f7d0c7eff90>
```



Taking the log of the image helps a lot: one clearly sees now the 5x12 Pilatus modules with their gaps, the beamstop and the diffuse scattering attributed to water (ice-ring). But no diffraction spots are visible, actually a few of them, but not enough to assess the quality of diffraction signal.

Here the images displayed in the IPython Notebook is about 326x333 while the input image was 2463x2527, so one pixel on the screen is representing 10x10 pixels on the detector. Because the Pilatus is a pixel-detector with a point spread function of 1 pixel, only 1% of the information is displayed. Two solution exists::

```
* make peaks 10x larger
* bin the image by a factor 10
```

The latter solution has already been seen and is called "binning"

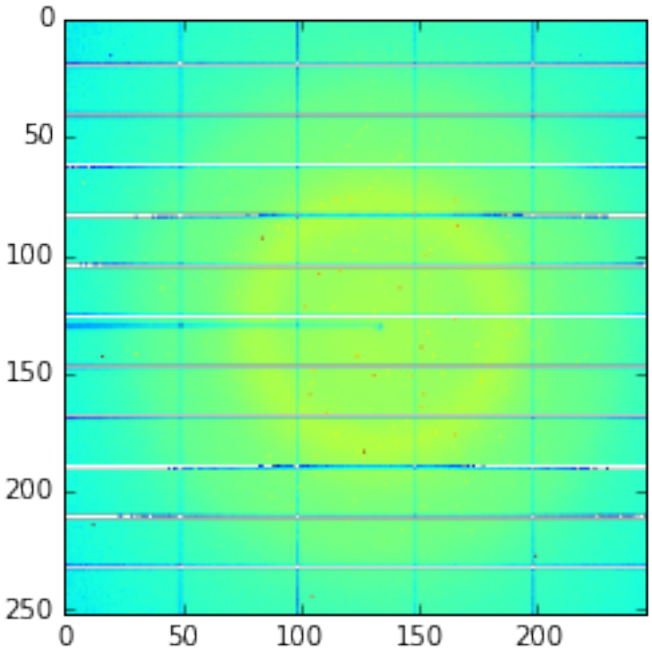
1.3 Image binning

To bin the image, it needs to be a full multiple of the binning factor, here 10. Moreover to avoid side effects with integers, we will convert the to float. We use here one of the two binning techics developped in the numpy exercise.

```
In [5]: chunk = img.data[:2520,:2460].astype("float")
        chunk.shape = 252, 10, 246, 10
        binned = chunk.sum(axis=-1).sum(axis=1)
        imshow(np.log(binned))
```

```
-c:4: RuntimeWarning: divide by zero encountered in log
-c:4: RuntimeWarning: invalid value encountered in log
```

```
Out[5]: <matplotlib.image.AxesImage at 0x7f7d0c730f90>
```



Note the gaps have almost disapeared due to the binning of the image ... but a dozen of diffraction spots are now clearly visible.

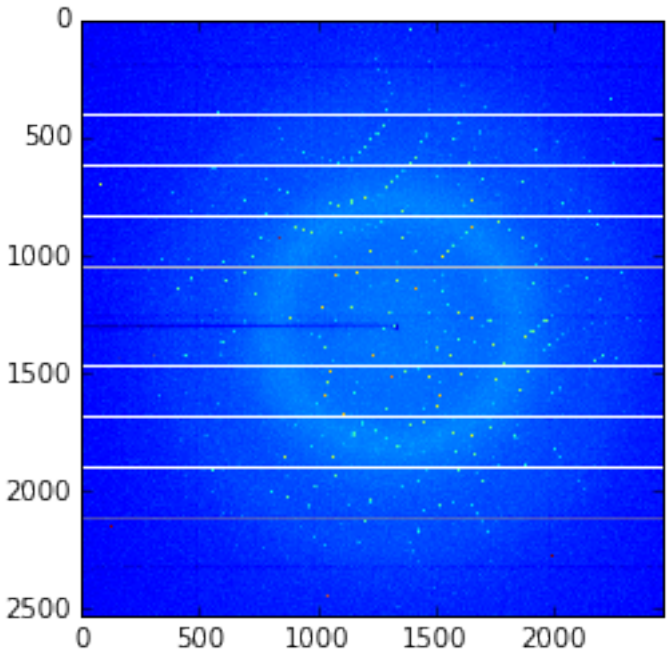
1.4 Increase the size of spots

This can be done via "grey-scale" morphology from scipy.ndimage. More details about the mathematics behind grayscale morphology can be found in: http://en.wikipedia.org/wiki/Mathematical_morphology We will use "grey_dilation" which actually will transform a peak of 1 pixel wide into a disc of 10 pixels.

```
In [6]: from scipy import ndimage

In [7]: imshow(np.log(ndimage.grey_dilation(img.data,10)))

Out[7]: <matplotlib.image.AxesImage at 0x7f7d0c59ad90>
```



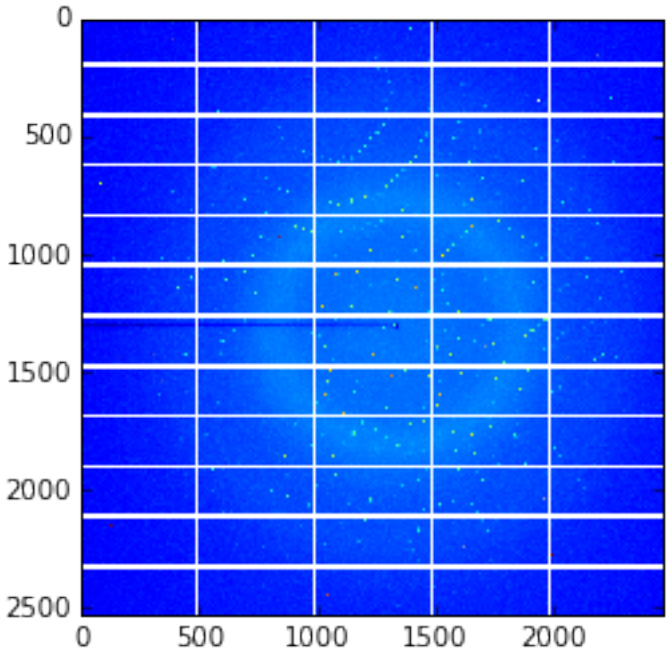
In this picture, one can clearly see hundreeds of diffraction spots up to a very wide diffraction angle (exhibiting the very good resolution for the protein diffractionist).

The last step is now to resume the masked pixelsfrom the original image:

```
In [8]: dil = ndimage.grey_dilation(img.data,10)
        dil[img.data<0] = 0
        imshow(np.log(dil))

-c:3: RuntimeWarning: divide by zero encountered in log

Out[8]: <matplotlib.image.AxesImage at 0x7f7d0014fb90>
```

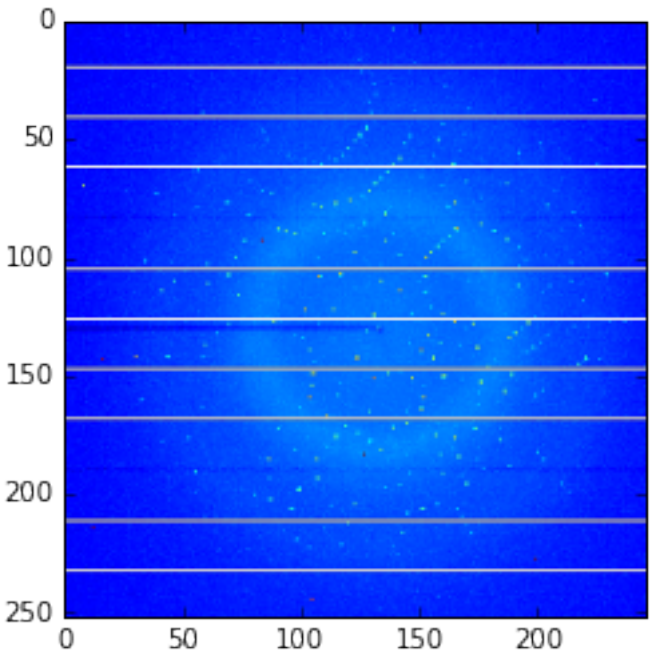


1.5 Alternative solution

Instead of using sum in the binning use the max to highlight peaks

```
In [9]: chunk = img.data[:2520,:2460]
        chunk.shape = 252, 10, 246, 10
        binned = chunk.max(axis=-1).max(axis=1)
        imshow(np.log(binned))
```

```
Out[9]: <matplotlib.image.AxesImage at 0x7f7d0007f650>
```



2 Conclusion

With such thumbnail image, of only 300x300, a protein crystalographer can actually assess the quality of the data-collection ongoing. Such technique is implemented in an EDNA plugin to generate the thumbnail displayed into ISPyB.