analysis3

June 11, 2021

1 Obtain spatial resolved grain maps of PARADIM-2

```
%run -i ../tomography/utils.py
%run -i startup2.py
```

```
Namespace: ['In', 'Out', 'ProgressBar', 'annotate_peaks', 'assign_Q_to_atlas',
'create_atlas', 'create_atlas_dask', 'create_dataset', 'create_grain_maps',
'create_windows_from_size', 'create_windows_from_width', 'dask', 'data',
'db_ana', 'db_cdf', 'db_csv', 'db_raw', 'df_uid', 'draw_windows', 'exit',
'facet', 'get_ipython', 'get_vlim', 'grain', 'grains', 'image_data', 'metadata',
'min_and_max_along_time', 'mpl', 'my_print', 'np', 'pd', 'peaks', 'pixel_to_Q',
'plot_grain_maps', 'plot_real_aspect', 'plt', 'quit', 'reformat_data',
'reshape', 'reshape_to_matrix', 'reshape_to_xarray', 'run', 'set_real_aspect',
'set_verbose', 'tp', 'track_peaks', 'typing', 'windows', 'xr']
```

1.1 Shadow method

1.1.1 Load and reformat the data

```
run = db_raw[df_uid['uid'][0]]
data = reformat_data(run.xarray_dask())
image_data = data["dexela_image"]
```

```
data
```

```
[4]: <xarray.Dataset>
     Dimensions:                 (dim_0: 1, dim_1: 3888, dim_2: 3072, frame: 4887)
     Coordinates:
       * frame                   (frame) int64 0 1 2 3 4 5 … 4882 4883 4884 4885 4886
     Dimensions without coordinates: dim_0, dim_1, dim_2
     Data variables:
         dexela_stats1_total     (frame) float64 dask.array<chunksize=(1,),
     meta=np.ndarray>
         dexela_image            (frame, dim_0, dim_1, dim_2) float64
     dask.array<chunksize=(1, 1, 3888, 3072), meta=np.ndarray>
         mXBase                  (frame) float64 dask.array<chunksize=(1,),
meta=np.ndarray>
    mXBase_user_setpoint  (frame) float64 dask.array<chunksize=(1,),
meta=np.ndarray>
    mYBase                (frame) float64 dask.array<chunksize=(1,),
meta=np.ndarray>
    mYBase_user_setpoint  (frame) float64 dask.array<chunksize=(1,),
meta=np.ndarray>
```

1.1.2 Test the method

We iterate the images along the time series and find the maximum and minimum values on each pixel. The images of the maximum values are like a photo of all shadows of the Bragg peaks on the detector while the ones of the minimum values are like the record of the lowest background intensities on the images.

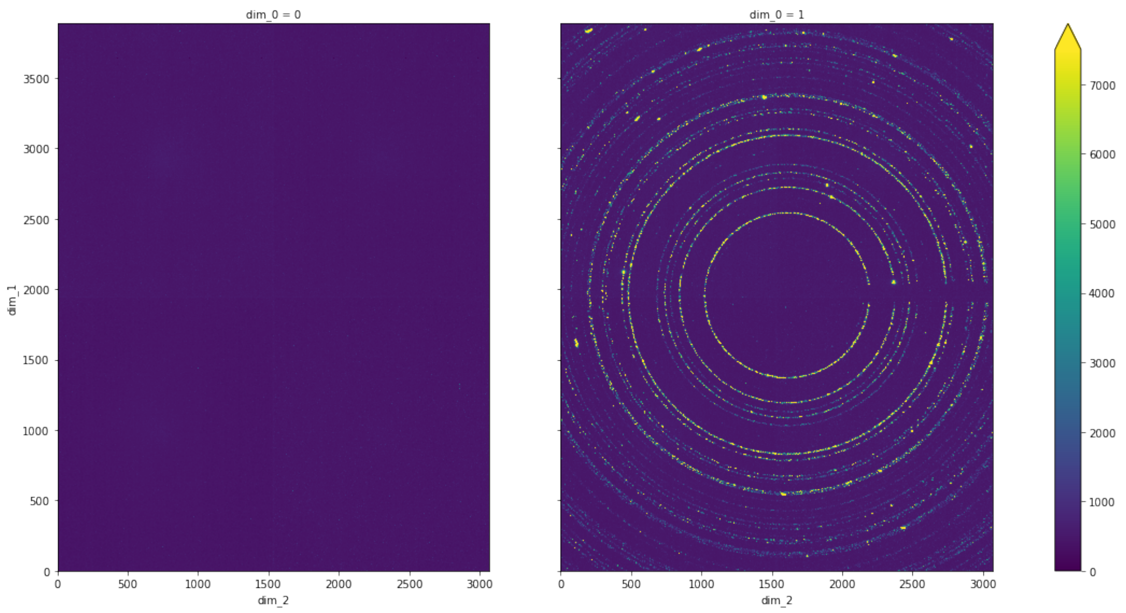
This method is implemented in `min_and_max_along_time`.

1.1.3 Analyze the result

The method will catch the powder diffraction in it. We need to filter the images and only apply the method on the images of non-powder part.

```
shadows = xr.load_dataarray("/Volumes/STAO_EXT/test_with_powder_frames_array.
↪nc")
```

```
facet = shadows.plot(col="dim_0", vmax=7500, size=8)
set_real_aspect(facet.axes)
```

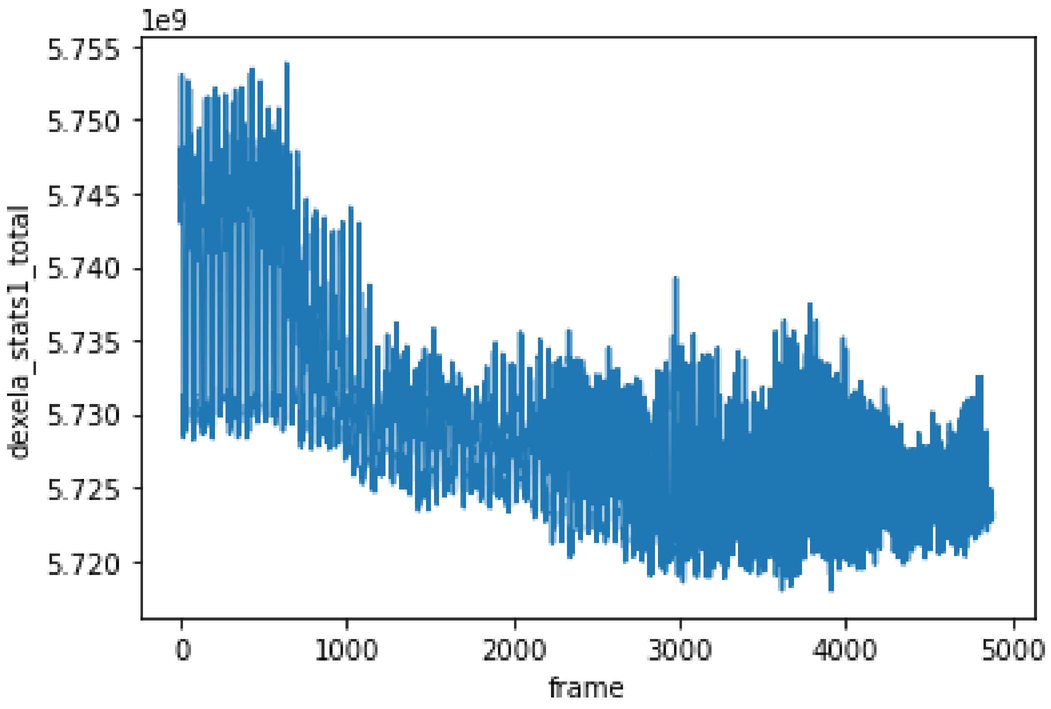


```
del shadows, facet
```

1.1.4 Filter the images

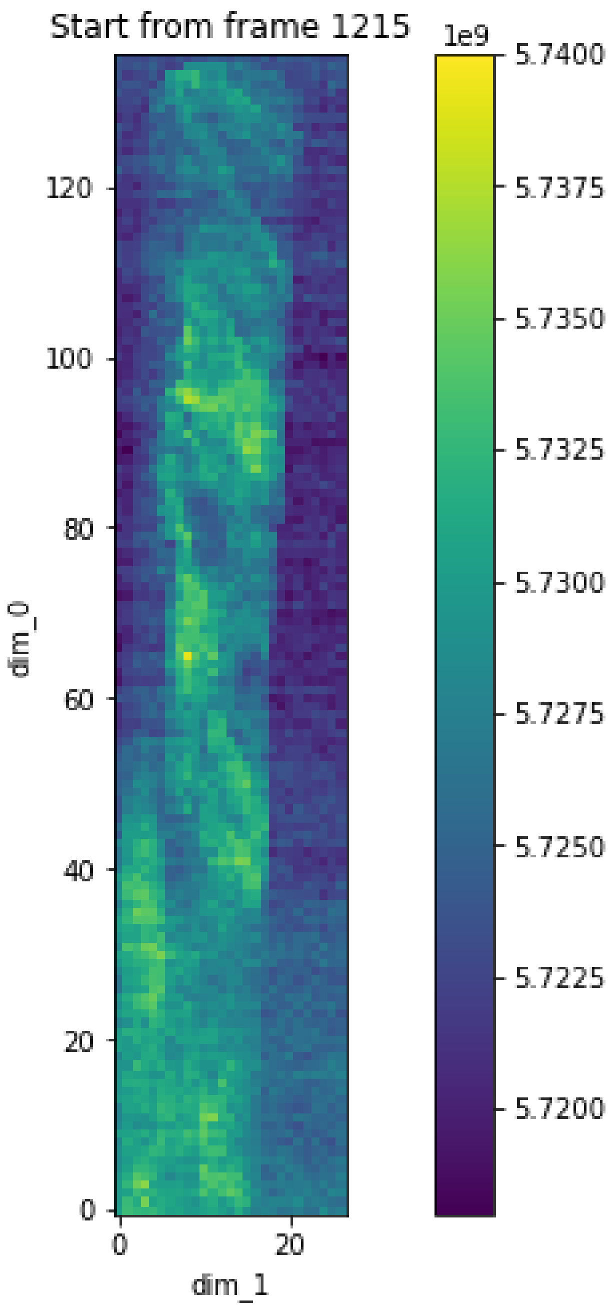
We filter the images according to the `dexela_stats1_total` which is the sum of intensity on the image. The powder diffraction image has much higher intensity than the single crystal diffraction image and the air scattering image.

```
image_sum_data = data['dexela_stats1_total'].compute()
image_sum_data.plot();
```



If row >= 45, the frames are all from the single crystal diffraction images. We can use the frame index of them to select the valid images.

```
image_sum_matrix = reshape_to_xarray(image_sum_data, run.start)
facet = image_sum_matrix[45:].plot(size=8, vmax=5.74e9)
facet.axes.set_title("Start from frame {}".format(image_sum_matrix[45].frame.
 ↪values.min()))
set_real_aspect(facet.axes)
```



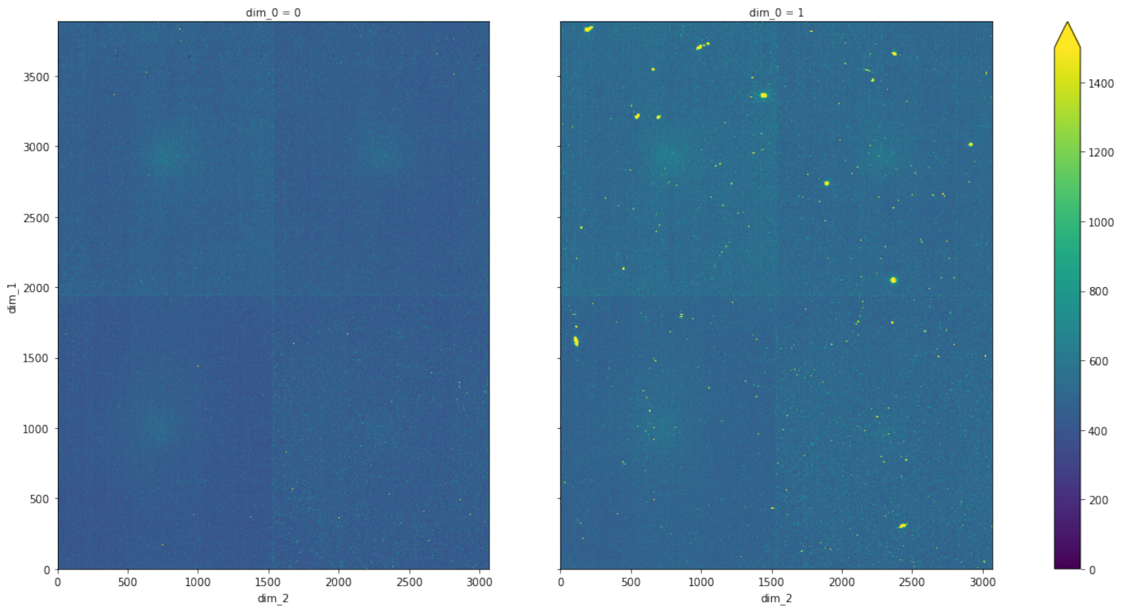
```
del image_sum_data, image_sum_matrix, facet
```

1.1.5 Run again with the filtered images

Get the maximum and minimum of pixel values for the images of the non-powder part.

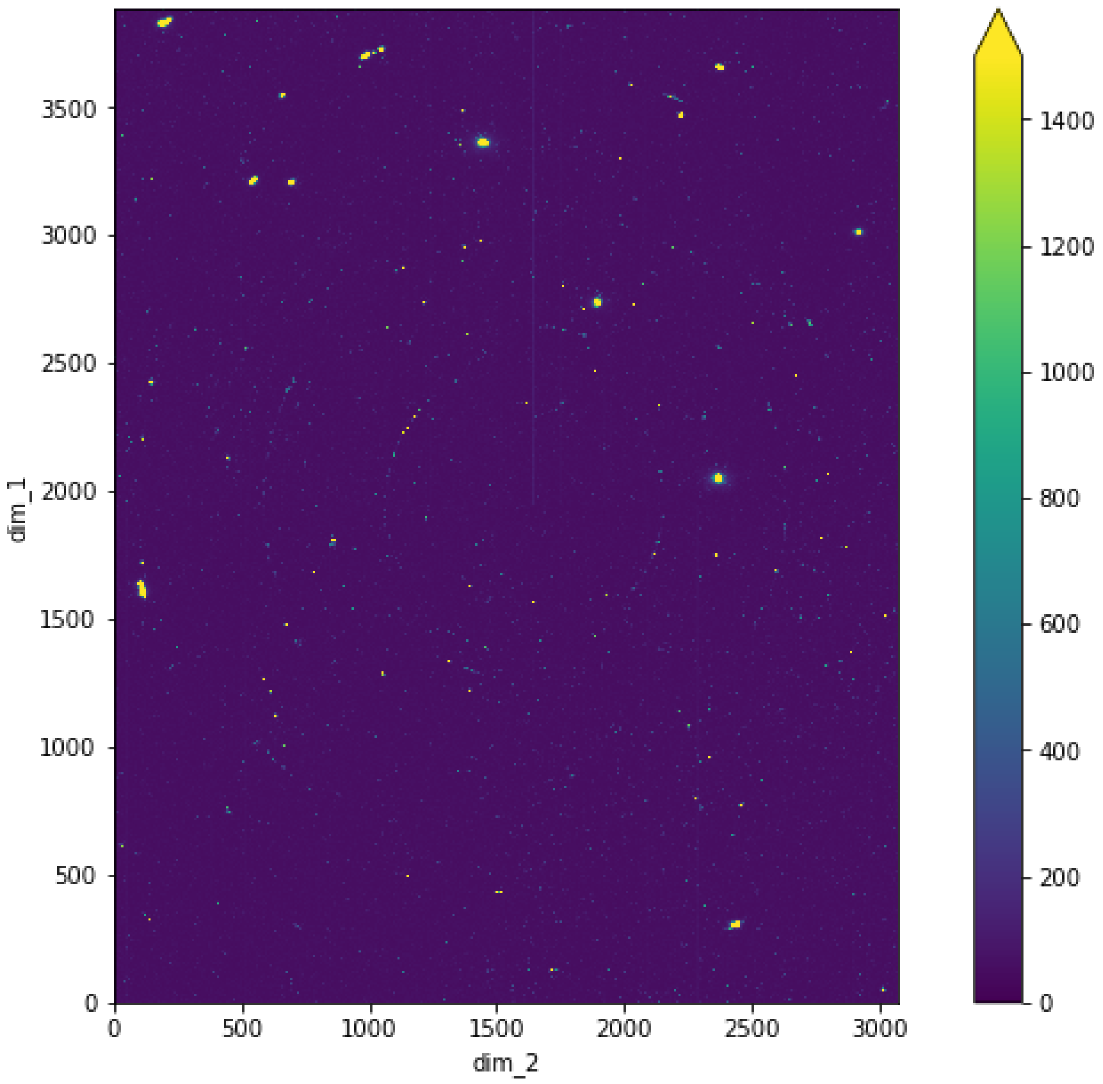
```
shadows = xr.load_dataarray("/Volumes/STAO_EXT/test_filtered_frames_array.nc")
```

```
facet = shadows.plot(col="dim_0", size=8, vmax=1500)
set_real_aspect(facet.axes)
```



The result shows all the shinning points that ever appeared in any of the frames. Some look like Bragg peaks while the other look like a part of powder diffraction rings.

```
subtracted_shadow = shadows[1] - shadows[0]
facet = subtracted_shadow.plot(size=8, vmax=1500)
set_real_aspect(facet.axes)
```



1.2 Locate the peak positions and track it

1.2.1 Locate the spots

Use `trackpy.locate` to find the spots on the image.

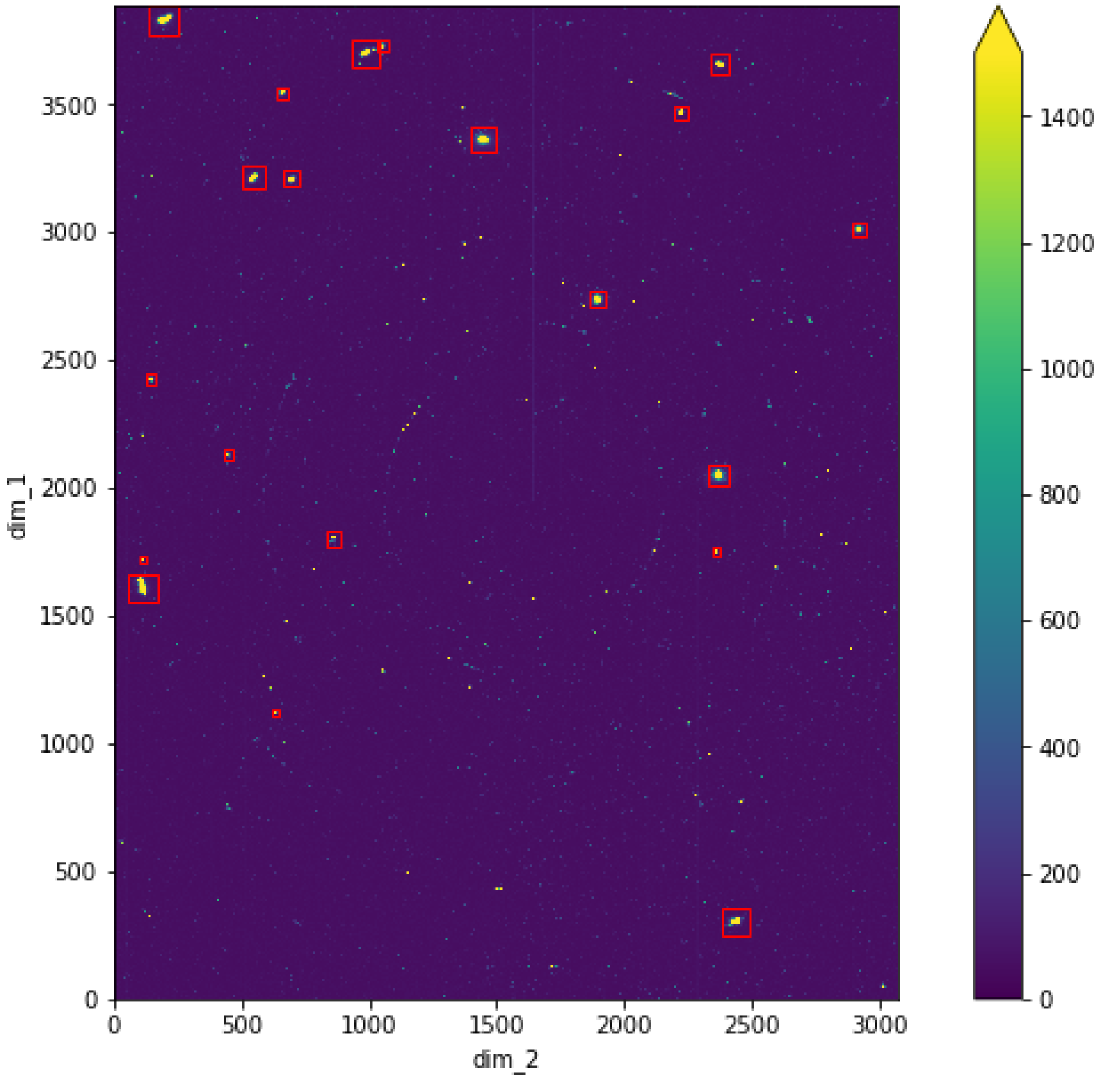
1.2.2 Select the ones to track

We select the 20 largest peaks to track.

```
peaks = db_csv.get_df("60c140527b072673151c761e")
peaks = peaks.sort_values("mass", ascending=False).iloc[:20]
```

```
windows = create_windows_from_size(peaks, 4.5)
```

```
facet = subtracted_shadow.plot(size=8, vmax=1500)
set_real_aspect(facet.axes)
draw_windows(windows, facet.axes)
plt.show()
```



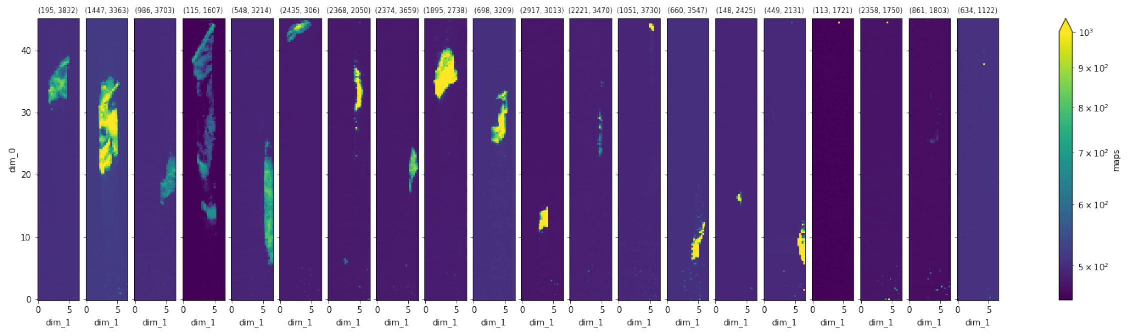
1.2.3 Run the tracking

1.2.4 Visualize the grain maps

This is the final result of 20 grain maps. They are visualized in a log scale color map.

```
grains = xr.load_dataset("/Volumes/STAO_EXT/grain_maps_first_try.nc")
grains = grains.set_index({"grain": ["x", "y"]})
```

```
facet = plot_grain_maps(grains, norm=mpl.colors.LogNorm(vmax=1000))
facet.set_titles(template="{value}", size=8);
```

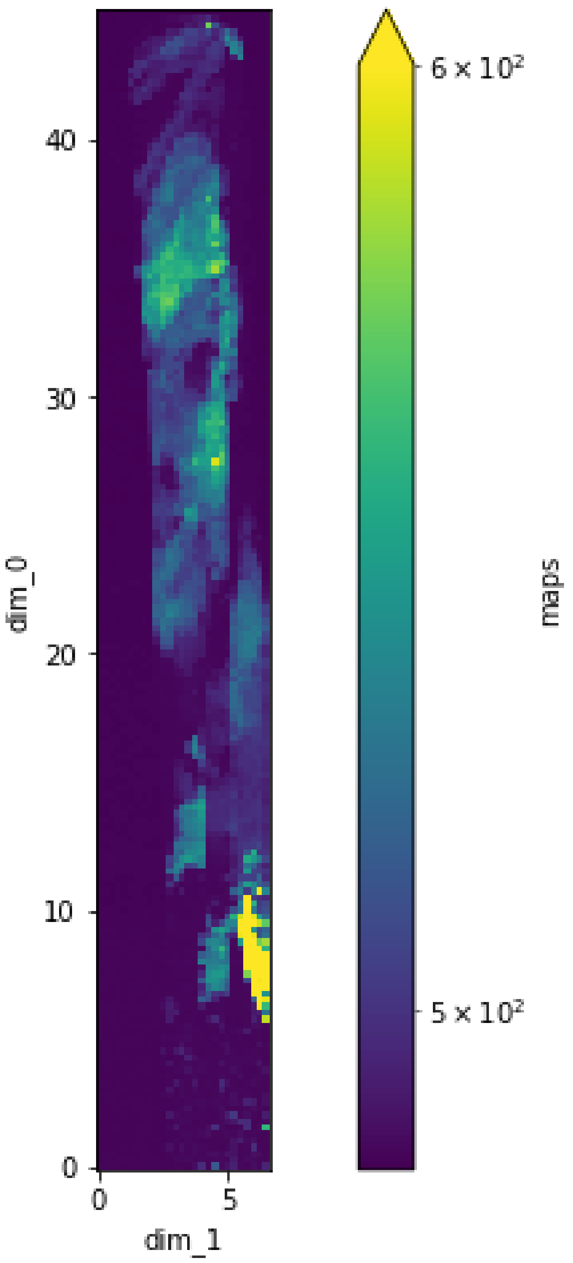


1.2.5 The mean of the grain maps

It shows the total distribution of all 20 grains.

```
mean_map = grains["maps"].mean(axis=2)
```

```
facet = mean_map.plot.pcolormesh(size=8, norm=mpl.colors.LogNorm(vmax=600))
set_real_aspect(facet.axes)
```


40
30
20
10
0
dim_0
0
5
dim_1
6 × 10^2
5 × 10^2
maps