Introduction to the SIX Data Analysis Codes

These codes are for a preliminary analysis on the RIXS data collected from SIX beamline, and run in Jupyter notebook based on python3. Currently, there are only three main files: 1) DataRetrive.ipynb (Jupyter file), 2) six_plot.py and Loc_Funcs.py (Python file).

Note: It's highly recommended to put both of .py files into one folder so that python can load them properly.

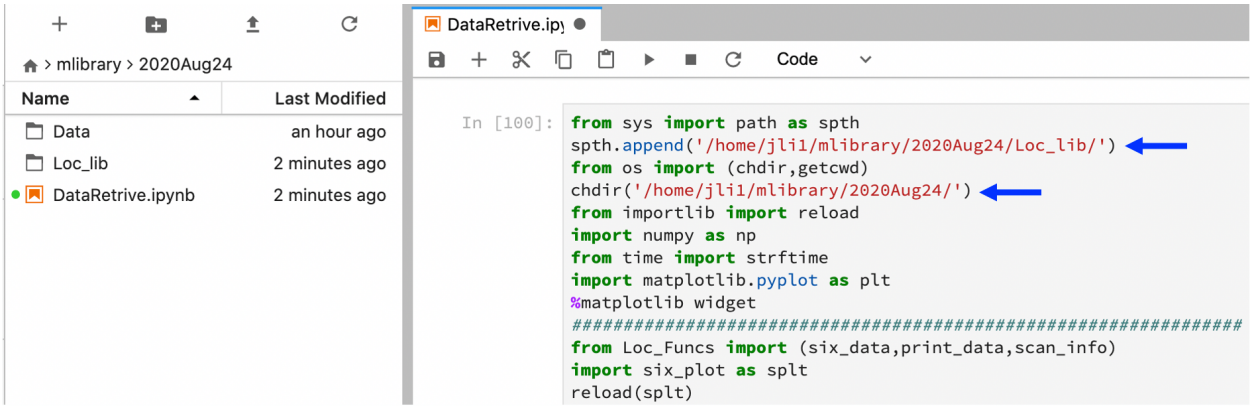
Below, we will focus on the DataRetrive.ipynb to see how the measuring data can be extracted from the raw data in beamline.

****

1) Configure the relating modules/scripts, and specify the folders.

(1) Add the directory of the folder (for example, Loc_lib) which contains .py files to the PYTHONPATH, so that python can find the .py file: spth.append('DIRECTORY')

(2) change the directory where python is currently working in to the one where DataRetrive.ipynb is in. It's not necessary but recommended, cause it can make your working space much cleaner when you have several beamtime, and also the processed data will be saved into this folder later on.



****

2) Check the motor scan

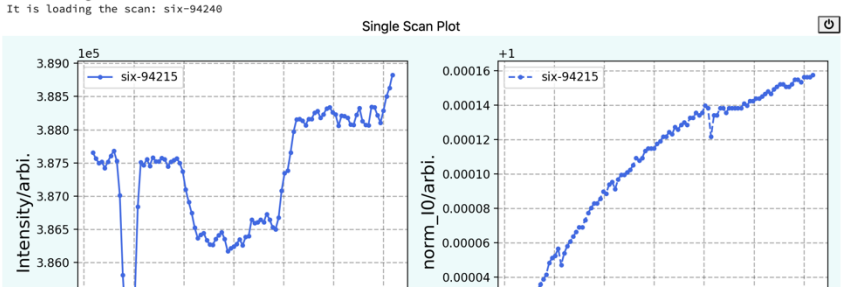
Before RIXS measurements, sample must be aligned properly and the XAS spectrum is collected to find the resonant energy. So in this cell, input the scan number to the 'scan' and also specify the motor (for example, 'x' or 'energy') and detector (generally, it is 'sclr_channels_chan2' or 'rixscam') used. Then run this cell and it will show the results. In 'norm_type', there are two choices for the data normalization: 'none' (by default) and 'I0'.

Single Scan Plot

```
scan = np.array([94216,94241]) # y scan
scan = np.array([94215,94227,94240]) # x scan

#########################################################
vari='x' # vari can be x,y,z,th,energy or any motor names
detector= 'sclr_channels_chan2' # it can be 'sclr_channels_chan2', 'rixscam', 'sclr_channels_chan6' or 'sclr_channels_chan8'
# sample=sample # sample name
#########################################################
splt.scan_plot(scan,vari=vari,dt=detector,plt_close=1)
fig = plt.gcf()
fig.tight_layout()
```

```
It is loading the scan: six-94215
It is loading the scan: six-94227
It is loading the scan: six-94240
```



The results can be saved to .hdf or .txt files from the button ‘save’, with the structure shown below:

Data structure of .hdf file.

E	I	I_norm	norm_I0	norm_type	th	tth	pol	T	x	y	z	slit_h	slit_v	ring_curr
520.0015	260357.0	260357.0	723571.0	None	10.997938	130.052144	LV	38.00465	20.9	4.2	17.88	150.0049	200.0189	399.905966
520.0921	260314.0	260314.0	723569.0	None	10.997938	130.052144	LV	38.00465	20.9	4.2	17.88	150.0049	200.0189	399.905966
520.1980	260395.0	260395.0	723569.0	None	10.997938	130.052144	LV	38.00465	20.9	4.2	17.88	150.0049	200.0189	399.905966
520.2984	260230.0	260230.0	723567.0	None	10.997938	130.052144	LV	38.00465	20.9	4.2	17.88	150.0049	200.0189	399.905966
520.3973	260233.0	260233.0	723568.0	None	10.997938	130.052144	LV	38.00465	20.9	4.2	17.88	150.0049	200.0189	399.905966
520.4983	260242.0	260242.0	723568.0	None	10.997938	130.052144	LV	38.00465	20.9	4.2	17.88	150.0049	200.0189	399.905966
520.5968	260187.0	260187.0	723566.0	None	10.997938	130.052144	LV	38.00465	20.9	4.2	17.88	150.0049	200.0189	399.905966
520.6984	260239.0	260239.0	723567.0	None	10.997938	130.052144	LV	38.00465	20.9	4.2	17.88	150.0049	200.0189	399.905966
520.7985	260231.0	260231.0	723566.0	None	10.997938	130.052144	LV	38.00465	20.9	4.2	17.88	150.0049	200.0189	399.905966
520.8993	260229.0	260229.0	723568.0	None	10.997938	130.052144	LV	38.00465	20.9	4.2	17.88	150.0049	200.0189	399.905966
520.9984	260306.0	260306.0	723567.0	None	10.997938	130.052144	LV	38.00465	20.9	4.2	17.88	150.0049	200.0189	399.905966
521.0985	260253.0	260253.0	723567.0	None	10.997938	130.052144	LV	38.00465	20.9	4.2	17.88	150.0049	200.0189	399.905966
521.1986	260275.0	260275.0	723568.0	None	10.997938	130.052144	LV	38.00465	20.9	4.2	17.88	150.0049	200.0189	399.905966
521.2983	260184.0	260184.0	723567.0	None	10.997938	130.052144	LV	38.00465	20.9	4.2	17.88	150.0049	200.0189	399.905966
521.3994	260228.0	260228.0	723566.0	None	10.997938	130.052144	LV	38.00465	20.9	4.2	17.88	150.0049	200.0189	399.905966
521.4987	260228.0	260228.0	723569.0	None	10.997938	130.052144	LV	38.00465	20.9	4.2	17.88	150.0049	200.0189	399.905966
521.5989	260280.0	260280.0	723570.0	None	10.997938	130.052144	LV	38.00465	20.9	4.2	17.88	150.0049	200.0189	399.905966
521.6906	260298.0	260298.0	723572.0	None	10.997938	130.052144	LV	38.00465	20.9	4.2	17.88	150.0049	200.0189	399.905966
521.7979	260348.0	260348.0	723570.0	None	10.997938	130.052144	LV	38.00465	20.9	4.2	17.88	150.0049	200.0189	399.905966
521.8980	260344.0	260344.0	723569.0	None	10.997938	130.052144	LV	38.00465	20.9	4.2	17.88	150.0049	200.0189	399.905966
521.9967	260406.0	260406.0	723574.0	None	10.997938	130.052144	LV	38.00465	20.9	4.2	17.88	150.0049	200.0189	399.905966
522.0979	260497.0	260497.0	723575.0	None	10.997938	130.052144	LV	38.00465	20.9	4.2	17.88	150.0049	200.0189	399.905966
522.1969	260550.0	260550.0	723574.0	None	10.997938	130.052144	LV	38.00465	20.9	4.2	17.88	150.0049	200.0189	399.905966
522.2976	260534.0	260534.0	723573.0	None	10.997938	130.052144	LV	38.00465	20.9	4.2	17.88	150.0049	200.0189	399.905966
522.3923	260743.0	260743.0	723575.0	None	10.997938	130.052144	LV	38.00465	20.9	4.2	17.88	150.0049	200.0189	399.905966
522.4997	260646.0	260646.0	723573.0	None	10.997938	130.052144	LV	38.00465	20.9	4.2	17.88	150.0049	200.0189	399.905966
522.5978	260743.0	260743.0	723575.0	None	10.997938	130.052144	LV	38.00465	20.9	4.2	17.88	150.0049	200.0189	399.905966
522.6995	260788.0	260788.0	723574.0	None	10.997938	130.052144	LV	38.00465	20.9	4.2	17.88	150.0049	200.0189	399.905966
522.7968	260876.0	260876.0	723576.0	None	10.997938	130.052144	LV	38.00465	20.9	4.2	17.88	150.0049	200.0189	399.905966
522.8978	260867.0	260867.0	723576.0	None	10.997938	130.052144	LV	38.00465	20.9	4.2	17.88	150.0049	200.0189	399.905966

Data structure of .txt file.

****

2) Load data into working space

In this cell, input the data number to scan which can be a list or numpy array. Considering two sensors in detector work together during measurement, we have to specify the border between them. Normally, the border is around 1500~1600. Then run it, and he resulting data is a dictionary whose keys are named as 'six-'+scan_number. In data, each item is also one dictionary which wraps all interesting information from that scan.

Load all RIXS data into working space

```
scan_set = np.arange(99425,99436+1,1) # E-dep at O K-edge & LV
#########################################################################################################
border = 1600
#####################################################################
data = six_data(scan_set,border=border)
print_data(data,scan=scan_set)
print('**********************************************************************************')
print('All scans have been loaded into the working space!')
```

```
--- six-99425 --- points 120 --- split time 5.0s --- total 600.0s --- duration 12.0min --- when Mon Aug 24 04:19:44 2020
--- six-99426 --- points 120 --- split time 5.0s --- total 600.0s --- duration 12.0min --- when Mon Aug 24 04:31:44 2020
--- six-99427 --- points 120 --- split time 5.0s --- total 600.0s --- duration 12.0min --- when Mon Aug 24 04:43:46 2020
--- six-99428 --- points 120 --- split time 5.0s --- total 600.0s --- duration 12.0min --- when Mon Aug 24 04:55:48 2020
--- six-99429 --- points 120 --- split time 5.0s --- total 600.0s --- duration 12.0min --- when Mon Aug 24 05:07:49 2020
--- six-99430 --- points 120 --- split time 5.0s --- total 600.0s --- duration 12.0min --- when Mon Aug 24 05:19:52 2020
--- six-99431 --- points 120 --- split time 5.0s --- total 600.0s --- duration 12.0min --- when Mon Aug 24 05:31:55 2020
--- six-99432 --- points 120 --- split time 5.0s --- total 600.0s --- duration 12.0min --- when Mon Aug 24 05:43:58 2020
--- six-99433 --- points 120 --- split time 5.0s --- total 600.0s --- duration 12.0min --- when Mon Aug 24 05:56:02 2020
--- six-99434 --- points 120 --- split time 5.0s --- total 600.0s --- duration 12.0min --- when Mon Aug 24 06:08:05 2020
--- six-99435 --- points 120 --- split time 5.0s --- total 600.0s --- duration 12.0min --- when Mon Aug 24 06:20:08 2020
--- six-99436 --- points 120 --- split time 5.0s --- total 600.0s --- duration 12.0min --- when Mon Aug 24 06:32:12 2020
+-----------+-------------+------------------+--------------+--------------+----------------+-----------+
| Scan Num. | Theta (deg) | Inc. Energy (eV) | 2Theta (deg) | Polarization | Exit Slit (um) | Temp. (K) |
+-----------+-------------+------------------+--------------+--------------+----------------+-----------+
|   99425   |     60.0    |       529.8      |     149.0    |      LV      |       30.0     |    40.0   |
|   99426   |     60.0    |       529.8      |     149.0    |      LV      |       30.0     |    40.0   |
|   99427   |     60.0    |       529.9      |     149.0    |      LV      |       30.0     |    40.0   |
|   99428   |     60.0    |       529.9      |     149.0    |      LV      |       30.0     |    40.0   |
|   99429   |     60.0    |       529.8      |     149.0    |      LV      |       30.0     |    40.0   |
|   99430   |     60.0    |       529.9      |     149.0    |      LV      |       30.0     |    40.0   |
|   99431   |     60.0    |       529.9      |     149.0    |      LV      |       30.0     |    40.0   |
```

****

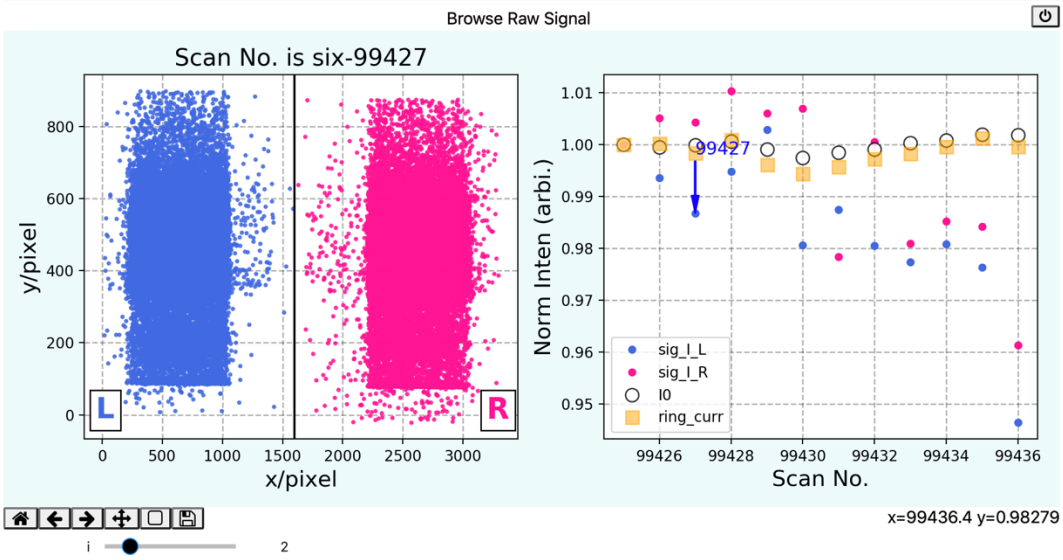
3) Check all data

After loading all data into working space, we can browse them through the following cell.

The result after running this cell is shown below, where the left plot is the signals on detector and the right one is plotting the total photon numbers (sig_I_L and sig_I_R) on both of sensors, ring current and also the I0 intensity during each scan. Sliding the bottom button i can display the results from different scans.

```
scan = scan_set

######################################################################
splt.raw_sig(data,scan=scan,border=border,plt_close=1)
fig = plt.gcf()
fig.tight_layout()
```



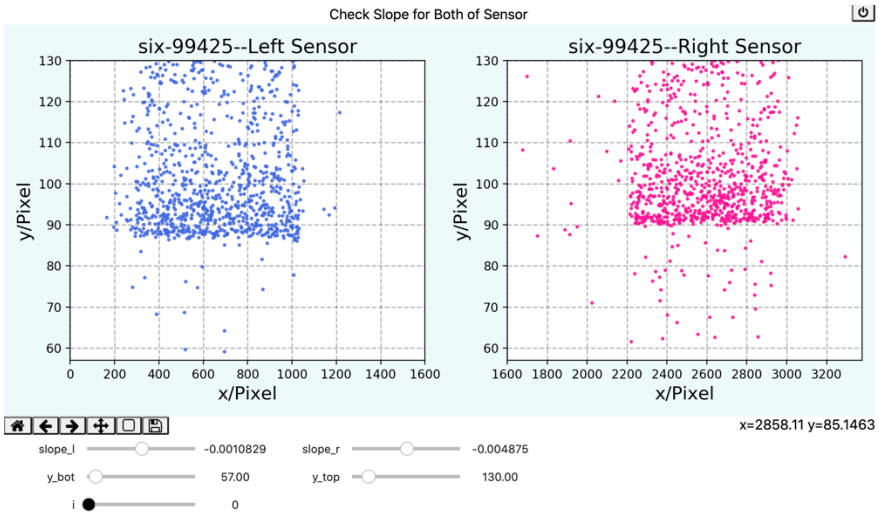
****

4) Check the slope for signals

Normally, the signals on detector are slightly inclined which can be well corrected by a line as found from experience. Input the estimated slopes for both of sensors, and we can then finely tweak them by sliding the buttons ‘slope_l’ and ‘slope_r’ respectively. The current values are shown at the right side of each slider. The ‘y_bot’ and ‘y_top’ are for setting the limit of y-axis. Again, sliding the bottom button i can display the results from different scans.

Check slope for pseudo-2D signal

In [6]:
```
slope_l = -0.0016
slope_r = -0.0044
######################################################################
slope_dict=splt.check_slope(data,scan=scan,slope_l=slope_l,slope_r=slope_r,border=border)
fig = plt.gcf()
fig.tight_layout()
```



****

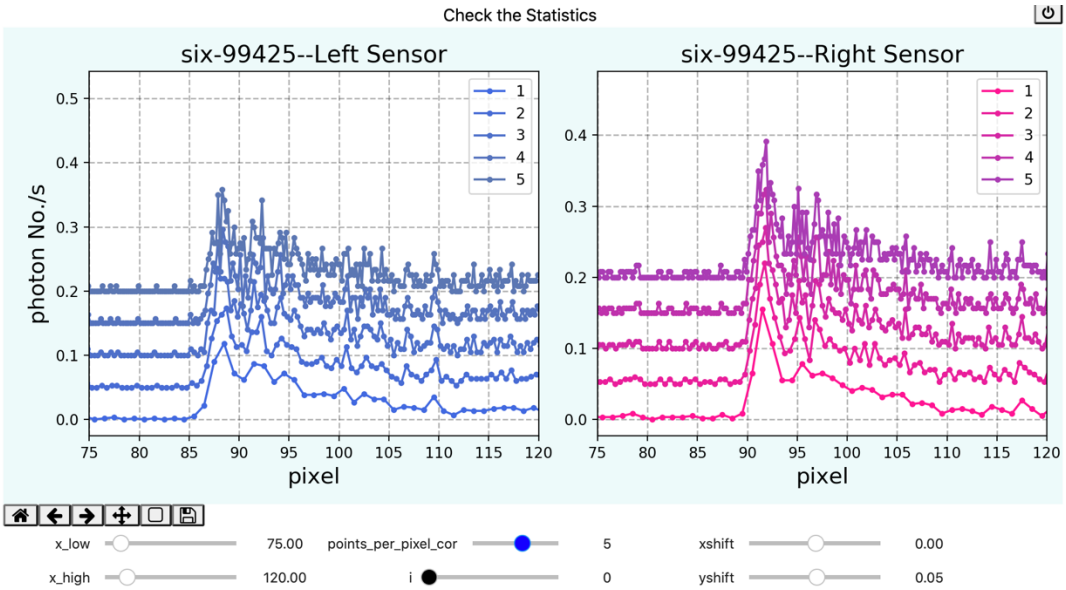
5) Check the points_per_pixel to get reasonable statistics and resolutions

Taking the optimized slope values from previous cell to correct the signal and convert it to 1-dimensional spectra with different points_per_pixel.

The signals shown in previous cells are actually 1-dimensional, therefore we use the concept of histogram to convert it to 1-dimensional spectra through photon distribution. The key parameter is points_per_pixel_cor, which ranges from 1 to 8 with a step of 1. As the resolution and statistics are two conflicting parameters, we have to make a compromise between these two based on the value of points_per_pixel_cor.

Handle the statistics and resolution ---> points_per_pixel

```
slope_l = slope_dict['slope_l'].get_interact_value()
slope_r = slope_dict['slope_r'].get_interact_value()
print(slope_l,slope_r)
#########################################################################
stat_dict = splt.stat_plot(data,scan=scan,slope_l=slope_l,slope_r=slope_r,border=border,yshift=0)
fig = plt.gcf()
fig.tight_layout()
```



****

6) Check the shift between different scans

With above parameters we have found, we can start to align all spectra.

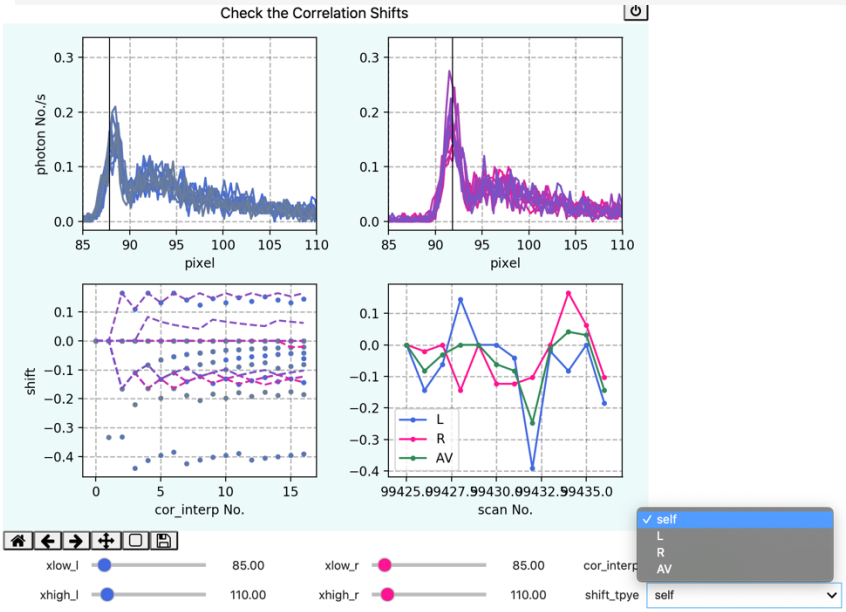
Considering the RIXS measurements are still not so efficient, we have to count a longer time for a particular scan. Practically, we collect several scans under one specific condition, during which the status of whole beamline might be drifted a bit, hence the signal could be shifted slightly between these different scans. To correct this discrepancy, we refer to the concept of cross-correlation shift which had been widely used in signal processing. This method can only make a shift with integer step, so we also use the interpolation to achieve the fractional shift.

'xlow' and 'xhigh' are the signal range for finding the correlation shifts, we can specify them through sliding the buttons, or just input the numbers in the 'cor_roi' where the first two are for left sensor and last two for right one. The 'cor_interp' characterizes the length [cor_interp*len(raw_sig)] of interpolated signal comparing to the raw one. As we have two sensors, the parameter of 'shift_type' gives the choice to make the correlated shifts for different sensors: 'self' → make the spectral shifts based on their own shifts calculated for left and right sensors, 'L'→use the calculated shifts from left sensors to correct both of sensors, 'R'→ use the calculated shifts from right sensors to correct both of sensors, 'AV'→ use the average of left and right shifts to correct both of sensors.

In the figure shown below, the black lines in top two plots depict the estimated elastic peak positions for both of sensors. The bottom left plot shows the correlated shifts as a function of cor_interp for all of scans. The dots are from left sensor, while the dashed line from right one. With increasing the cor_interp, the calculated shifts should converge to one value. The bottom right plot is the final calculated shifts by referring to the first scan for both of sensors as a function of scan.

Check the correlation between different scans

```
slope_l = slope_dict['slope_l'].get_interact_value()
slope_r = slope_dict['slope_r'].get_interact_value()
points_per_pixel_cor = stat_dict['points_per_pixel_cor'].get_interact_value()
print(points_per_pixel_cor)
#################################
cor_roi = np.array([[85,110],[85,110]])
#########################################################################
cor_dict = splt.check_cor(data,scan=scan,border=border,
                slope_l=slope_l,slope_r=slope_r,points_per_pixel=points_per_pixel_cor,
                cor_roi = cor_roi,
                plt_close=1)
#########################################################################
fig = plt.gcf()
fig.tight_layout()
```



****

7) Check the spectra after correlation shifts

After performing the correlation shifts, we can check how the spectra are aligned from this cell. For some weak signals which do not show peak-like features, the correlation shifts between different scans may not work properly, so it's better to check the spectra after correlation shifts. If there are any irregular shapes in the data, we can remove that scan at the beginning.

Check spectra after correlated shifts

```
border = border
slope_l = slope_dict['slope_l'].get_interact_value()
slope_r = slope_dict['slope_r'].get_interact_value()
points_per_pixel = stat_dict['points_per_pixel_cor'].get_interact_value()
################################################################################
cor_roi_l = [cor_dict['xlow_l'].get_interact_value(),cor_dict['xhigh_l'].get_interact_value()]
cor_roi_r = [cor_dict['xlow_r'].get_interact_value(),cor_dict['xhigh_r'].get_interact_value()]
cor_interp=cor_dict['cor_interp'].get_interact_value()
shift_type=cor_dict['shift_type'].get_interact_value()
print(shift_type)
#########################################################################
splt.check_spec_shift(data, scan=scan,border=border,
                      slope_l=slope_l, slope_r=slope_r, points_per_pixel=points_per_pixel,
                      cor_roi_l=cor_roi_l,cor_roi_r=cor_roi_r,cor_interp=cor_interp,shift_type=shift_type,
                      plt_close=1)
#############################################
fig = plt.gcf()
fig.tight_layout()
```

****

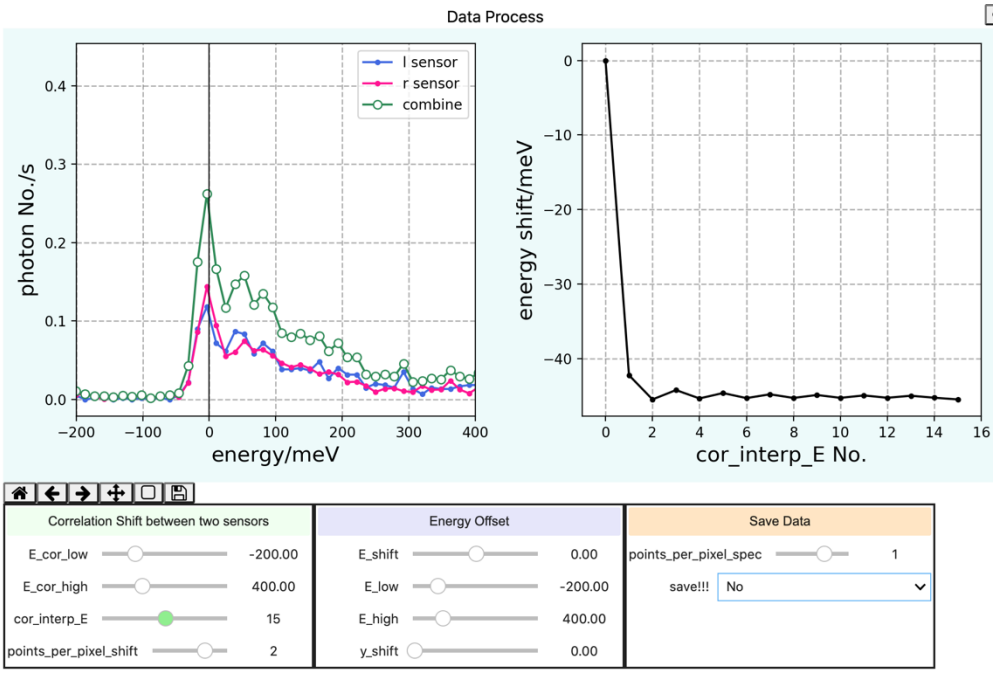
8) Get the final RIXS spectra

With all the key parameters we have optimized and the energy calibration from beamline, run this cell to get the final RIXS spectra from stacking all of scans for both of sensors. Normally, the left spectrum is a bit misaligned from the right one on energy scale, hence we can perform the same correlated shift between two sensors with the reference from left sensor. The 'E_cor_low' and 'E_cor_high' are the energy range for finding correlation shift, the 'cor_interp_E' again characterizes the length of interpolated signal. The 'points_per_pixel_shift'

is used to find a proper energy shift between two sensors within the energy range defined by ‘E_cor_low’ and ‘E_cor_high’. Sometimes the data points within ‘E_cor_low’ and ‘E_cor_high’ are a bit less for finding the correlated shift. So we can use the ‘points_per_pixel_shift’ to increase the data points. It’s ‘points_per_pixel_cor’ by default, but different from ‘points_per_pixel_cor’ that is used to find the correlated shifts between different scans. The ‘E_low’ and ‘E_high’ are the energy limit for the left plot. The ‘points_per_pixel_spec’ is for the final RIXS spectrum and referred to the ‘points_per_pixel_shift’ by default, but can be different from ‘points_per_pixel_shift’ in general. ‘E_shift’ is to correct the misalignment of zero-energy line.

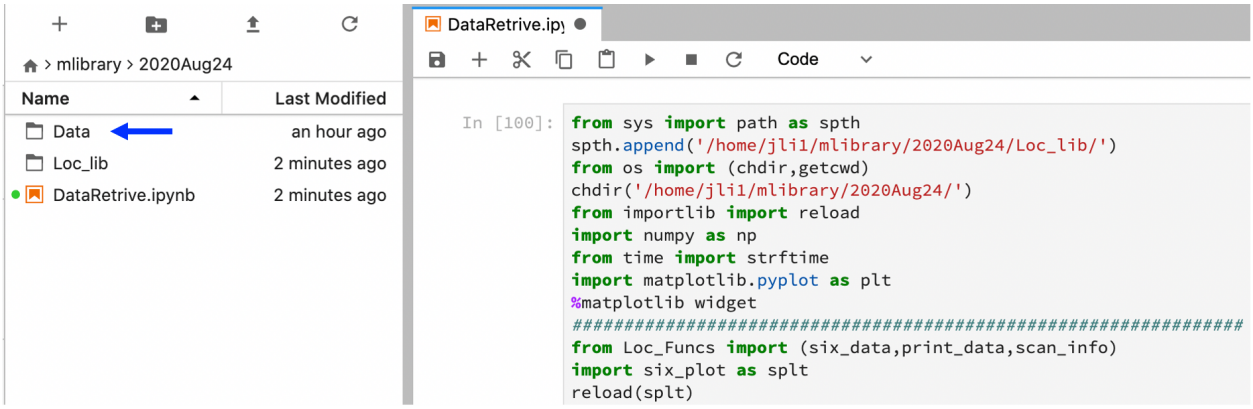
Perform correlation between different sensors (left and right) leading to the sum and save data!

```
E_cali = 21.73 # meV/pixel at Fe L-edge
#############################################################
border = border
slope_l = slope_dict['slope_l'].get_interact_value()
slope_r = slope_dict['slope_r'].get_interact_value()
points_per_pixel = stat_dict['points_per_pixel_cor'].get_interact_value()
#############################################################################
cor_roi_l = [cor_dict['xlow_l'].get_interact_value(),cor_dict['xhigh_l'].get_interact_value()]
cor_roi_r = [cor_dict['xlow_r'].get_interact_value(),cor_dict['xhigh_r'].get_interact_value()]
cor_interp=cor_dict['cor_interp'].get_interact_value()
shift_type=cor_dict['shift_type'].get_interact_value()
print(shift_type)
#####################################################################
splt.spec_cor(data, scan=scan,E_cali=E_cali,border=border,
                    slope_l=slope_l, slope_r=slope_r, points_per_pixel=points_per_pixel,
                    cor_roi_l=cor_roi_l,cor_roi_r=cor_roi_r,cor_interp=cor_interp,shift_type=shift_type,
                    yshift=0)
#########################################
fig = plt.gcf()
fig.tight_layout()
```



After confirming all these parameters, we can set the ‘save’ (it’s ‘No’ by default) to ‘Yes_hdf/Yes_txt’ to save the resulting RIXS spectra to hdf5/txt file in a folder named as ‘Data’. Once the ‘save’ is set to ‘Yes_hdf/Yes_txt’, whenever we change the other parameters, the new

results will automatically overwrite the previous one afterwards. So it's better to keep 'save' to 'No' after saving results.

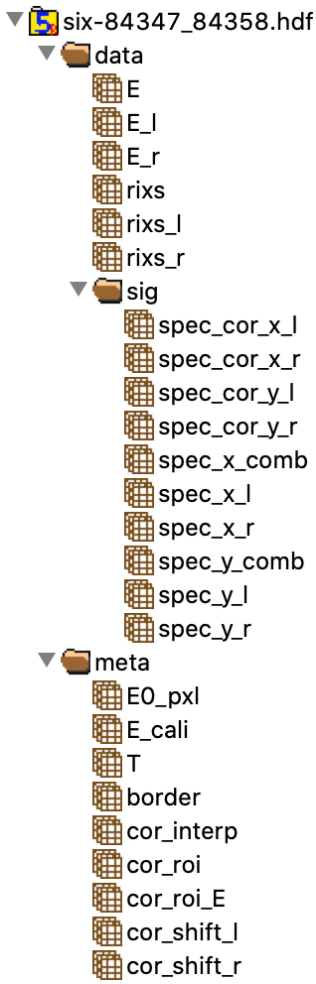


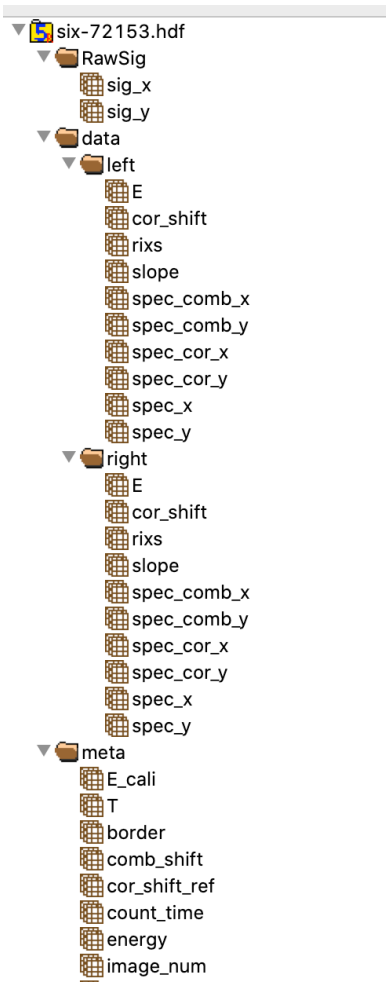
****

9) Data structure

The resulting data is named as 'six-' + first scan number + '_' + last scan number+'.hdf/.txt'.

→ For the .hdf file, the data structure is shown below (the left one): in 'data' folder, the 'rixs' is the combined spectra from both of sensors with 'E' being the energy axis. 'rixs_l' and 'E_l' are from left sensor, and 'rixs_r' and 'E_r' from right sensor. In 'sig' folder, we save all the signals during data process. The interesting parameters from beamline are saved to 'meta' folder.





We also save the result for each individual scan. A typical data structure can be found above (the right figure).

Note: A software to view the .hdf file is HDFView, which is available from below website:

https://www.hdfgroup.org/downloads/hdfview/

→ For .txt file, the data (for the stack of several scans or single scan) looks like:

E	rixs	E_l	rixs_l	E_r	rixs_r	energy	th	tth	T	pol	points_per_pixel	slope_l	slope_r	border	E_cali
-3003.185	0.000639	-3003.185	0.000000	-3003.185	0.000639	931.00005	119.999953	130.042873	38.26685	LV		6.0	-0.001494	-0.003916	1600.0	32.94
-2997.695	0.000766	-2997.695	0.000000	-2997.695	0.000766	931.00005	119.999953	130.042873	38.26685	LV		6.0	-0.001494	-0.003916	1600.0	32.94
-2992.205	0.000435	-2992.205	0.000000	-2992.205	0.000435	931.00005	119.999953	130.042873	38.26685	LV		6.0	-0.001494	-0.003916	1600.0	32.94
-2986.715	0.000084	-2986.715	0.000000	-2986.715	0.000084	931.00005	119.999953	130.042873	38.26685	LV		6.0	-0.001494	-0.003916	1600.0	32.94
-2981.225	0.000482	-2981.225	0.000000	-2981.225	0.000482	931.00005	119.999953	130.042873	38.26685	LV		6.0	-0.001494	-0.003916	1600.0	32.94
-2975.735	0.000836	-2975.735	0.000000	-2975.735	0.000836	931.00005	119.999953	130.042873	38.26685	LV		6.0	-0.001494	-0.003916	1600.0	32.94
-2970.245	0.000548	-2970.245	0.000000	-2970.245	0.000548	931.00005	119.999953	130.042873	38.26685	LV		6.0	-0.001494	-0.003916	1600.0	32.94
-2964.755	0.000749	-2964.755	0.000000	-2964.755	0.000749	931.00005	119.999953	130.042873	38.26685	LV		6.0	-0.001494	-0.003916	1600.0	32.94
-2959.265	0.001000	-2959.265	0.000000	-2959.265	0.001000	931.00005	119.999953	130.042873	38.26685	LV		6.0	-0.001494	-0.003916	1600.0	32.94
-2953.775	0.001535	-2953.775	0.000000	-2953.775	0.001535	931.00005	119.999953	130.042873	38.26685	LV		6.0	-0.001494	-0.003916	1600.0	32.94
-2948.285	0.001415	-2948.285	0.000000	-2948.285	0.001415	931.00005	119.999953	130.042873	38.26685	LV		6.0	-0.001494	-0.003916	1600.0	32.94
-2942.795	0.000435	-2942.795	0.000000	-2942.795	0.000435	931.00005	119.999953	130.042873	38.26685	LV		6.0	-0.001494	-0.003916	1600.0	32.94
-2937.305	0.001097	-2937.305	0.000000	-2937.305	0.001097	931.00005	119.999953	130.042873	38.26685	LV		6.0	-0.001494	-0.003916	1600.0	32.94
-2931.815	0.000819	-2931.815	0.000000	-2931.815	0.000819	931.00005	119.999953	130.042873	38.26685	LV		6.0	-0.001494	-0.003916	1600.0	32.94
-2926.325	0.000819	-2926.325	0.000735	-2926.325	0.000084	931.00005	119.999953	130.042873	38.26685	LV		6.0	-0.001494	-0.003916	1600.0	32.94
-2920.835	0.000532	-2920.835	0.000265	-2920.835	0.000268	931.00005	119.999953	130.042873	38.26685	LV		6.0	-0.001494	-0.003916	1600.0	32.94
-2915.345	0.000997	-2915.345	0.000000	-2915.345	0.000997	931.00005	119.999953	130.042873	38.26685	LV		6.0	-0.001494	-0.003916	1600.0	32.94
-2909.855	0.001053	-2909.855	0.000000	-2909.855	0.001053	931.00005	119.999953	130.042873	38.26685	LV		6.0	-0.001494	-0.003916	1600.0	32.94
-2904.365	0.000861	-2904.365	0.000312	-2904.365	0.000548	931.00005	119.999953	130.042873	38.26685	LV		6.0	-0.001494	-0.003916	1600.0	32.94
-2898.875	0.001437	-2898.875	0.000688	-2898.875	0.000749	931.00005	119.999953	130.042873	38.26685	LV		6.0	-0.001494	-0.003916	1600.0	32.94
-2893.385	0.001476	-2893.385	0.001091	-2893.385	0.000384	931.00005	119.999953	130.042873	38.26685	LV		6.0	-0.001494	-0.003916	1600.0	32.94
-2887.895	0.000909	-2887.895	0.000909	-2887.895	0.000000	931.00005	119.999953	130.042873	38.26685	LV		6.0	-0.001494	-0.003916	1600.0	32.94
-2882.405	0.000202	-2882.405	0.000202	-2882.405	0.000000	931.00005	119.999953	130.042873	38.26685	LV		6.0	-0.001494	-0.003916	1600.0	32.94
-2876.915	0.001111	-2876.915	0.001111	-2876.915	0.000000	931.00005	119.999953	130.042873	38.26685	LV		6.0	-0.001494	-0.003916	1600.0	32.94
-2871.425	0.000741	-2871.425	0.000688	-2871.425	0.000054	931.00005	119.999953	130.042873	38.26685	LV		6.0	-0.001494	-0.003916	1600.0	32.94
-2865.935	0.000595	-2865.935	0.000000	-2865.935	0.000595	931.00005	119.999953	130.042873	38.26685	LV		6.0	-0.001494	-0.003916	1600.0	32.94
-2860.445	0.001553	-2860.445	0.001202	-2860.445	0.000351	931.00005	119.999953	130.042873	38.26685	LV		6.0	-0.001494	-0.003916	1600.0	32.94
-2854.955	0.002312	-2854.955	0.002312	-2854.955	0.000000	931.00005	119.999953	130.042873	38.26685	LV		6.0	-0.001494	-0.003916	1600.0	32.94
-2849.465	0.002741	-2849.465	0.002688	-2849.465	0.000054	931.00005	119.999953	130.042873	38.26685	LV		6.0	-0.001494	-0.003916	1600.0	32.94
-2843.975	0.002905	-2843.975	0.001534	-2843.975	0.001372	931.00005	119.999953	130.042873	38.26685	LV		6.0	-0.001494	-0.003916	1600.0	32.94

****

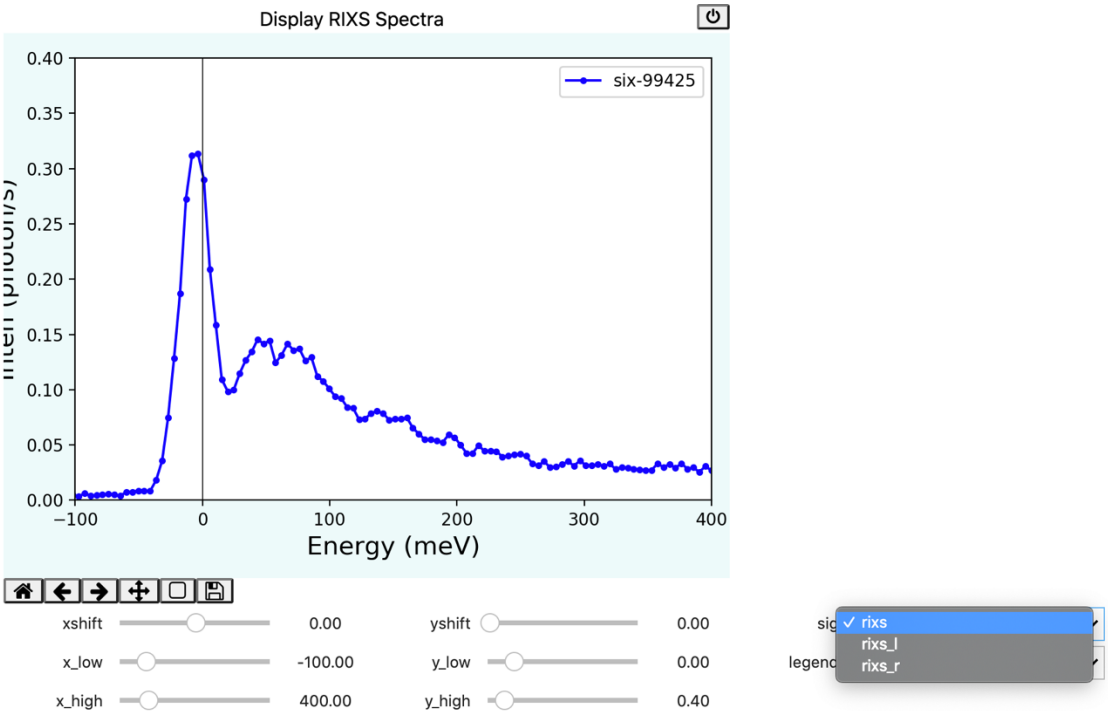
10) View the resulting RIXS spectra

To see the RIXS spectra, we can run the next cell, in which we have to specify the data folder, input the scan number and data type (hdf or txt). In the pop-out figure, we can check different signals from 'sig': 'rixs' (default) is the combination of both sensors, 'rixs_l' is the left sensor while 'rixs_r' is the right one. The 'xshift' and 'yshift' can shift the spectra horizontally or vertically.

Display RIXS Spectra

```
data_folder = getcwd()+'/Data/points_per_pixel-3/'
#####################################################
scan = np.array([99425])


data_type = 'hdf' # it can be 'hdf' or 'txt'
#########################################################################################################################
# plt.close()
# fig = plt.figure('Display RIXS Spectra', figsize=(3, 7))
splt.rixs1d(data_folder=data_folder,scan=scan,data_type=data_type,plt_close=1)
fig = plt.gcf()
fig.tight_layout()
```



****

11) View the 2D RIXS map [**This cell only works for .hdf files!!!**]

We can also check the 2D RIXS map from the following cell, where five parameters have to be initialized before running it: 'data_folder', 'scan', 'vari'(normally, this kind of results is a function of θ-angle or incident photon energy), 'vari_offset' (characterize the potential offset

for the choosing ‘vari’) and ‘cut_type’ (determine which kind of cut (vertical or horizaontal) is going to be shown in the right plot.

In the pop-out figure, we can re-size the 2D map from ‘v_low, v_high, h_low, h_high’ and change the color scale from ‘c_low’ and ‘c_high’. The ‘sig’ provides the option to choose which set of data we can view: ‘rixs’ (default) is the combination of both sensors, ‘rixs_l’ is the left sensor while ‘rixs_r’ is the right one. ‘E_shift’ corrects the potential offset for the zero-energy in all of RIXS spectra. ‘cut’ will be the data shown in the right plot. ‘cut_int_wid’ gives the choice to see the integration of a certain range of spectra (starting from ‘cut’). ‘x_low’ and ‘x_high’ are the horizonta limits of right plot. The last parameter ‘sig_interp’ will do the interpolation for the spectra in right plot, which can be useful for the cut with a lower statistics.

Display RIXS Map

```
data_folder = getcwd()+'/Data/RawData/'
#####################################################
scan = np.arange(72132,72143,1)
#####################################################
vari = 'energy' # This parameter can be 'energy' (incident photon energy) or 'th'.
vari_offset = 0 # The potential offset for vari
cut_type = 'V' # There are two choices: 'H' or 'V' !!!
#########################################################################################
splt.rixs2d(data_folder=data_folder,scan=scan,vari=vari,vari_offset = vari_offset,
            cut_type=cut_type,colormap='jet')
fig = plt.gcf()
fig.tight_layout()
```