Program for XMLD and XMCD spectra analysis

Technical documentation

1. Libraries and modules

1.1. Numpy

Used for conversion of .txt file containing data to approachable to work with format of data.

1.2. Os

Used for file handling in the process of uploading and saving.

1.3. Matplotlib

Used for graphical visualization of data in the form of various plots.

1.4. Re

Used for searching the file to find individual measurements.

1.5. CSV

Used for writing results to file in a form enabling further analysis.

1.6. Tkinter

Used for creating a graphical user interface for the program.

2. Functions

2.1. Uploading

2.1.1. Uploading a file

A standard data set is a .dat file that contains numbers of measurements. Every measurement consists of several columns proceded by several lines of comments describing measurement. Essential for analysis is the first line of comment which contain a number of measurement and its type and two columns of the data.

To load a file *select_file()* function is used. It opens File Explorer which allows for choosing a file. If the file is correct the program writes the file line by line to variable *Lines* and closes the file

To find all measurements existing in the whole data set the function *list of measurement()* is used.

```
#searching file to find number of measurements
def list_of_measurements():
    #pattern for searching first and list line of measurement in the file
    pattern_all = r'#S\s\d+'
    pattern_rscan = r'#S\s\d+\srscan'

    total = 0
    #searching data from specified measurement
    for L in Lines:
        if re.search(pattern_all, L):
            total += 1

    rscans.clear()
    #only numbers of rscan
    for L in Lines:
        search_res = re.search(pattern_rscan, L)
        if search_res:
            text = search_res.group()
            number = re.search(r'\d+',text)
            rscans.append(int(number.group()))

    return total
```

Figure 1. *List of measurement()* function.

There are two regular expressions used:

- pattern_all – finding all measurements
- pattern_rscan – finding only measurements made as a function of energy (subject of analysis)

Variable *total* is the number of all measurements in the data set and is used in the program for creating a list of available measurements (measurements are saved in the data set sequentially from the first to the last).

Measurement numbers made as a function of energy are saved in list *rscans*.

2.1.2. Opening specific measurement

For opening specific measurements for the purpose of plotting, for example, the function *search in file(nb)* is used.

```
pattern_first = r'#S\s{}'.format(nb) #start of data
pattern_last = r'#C\sAcquisition\sended' #end of data
```

Figure 2. Regular expressions in *search in file* function.

There are two regular expressions used (Figure 2.):

- pattern_first – the first line of commentary of a given measurement (*nb* – number of the measurement you are looking for)
- pattern_last – the last line of data of a given measurement

To obtain the number of the first line of data of a given measurement, the number of comment lines is added to the number of the first line of commentary (Figure 3.).

```
for L in Lines: #searching for the beginning of data in uploaded file
    if re.search(pattern_first, L):
        break
    first_line +=1
data_start = first_line + nb_lines_comment
```

Figure 3. Searching for the first line of data in the *search in file* function.

The function returns a tuple with a number of the first and the last line of data.

2.2. Displaying plots

2.2.1. Plotting spectra

For displaying spectra *Plotter* function is used. The function takes five parameters, including two optional ones:

- *measurement_nb* – the number of spectra that you want to plot
- *ax* – matplotlib Axes object that you want to use to plot spectrum
- *mode* – the function has two modes of spectra displaying which are activated by passing to the function proper string:
  - 'virgin': In this mode the raw data is used to draw the graph. This mode is used in the first stages of the program.

  - 'sub and normalized': In this mode data after normalization and eventual background removal is plotted.

- *w* (optional) – first parameter of fitted background (strip background width)
- *iter_nb* (optional) – second parameter of fitted background (strip background number of iterations)

The function writes data for specific measurement to a temporary file and extracts two columns from it and save to ndarray *data* (Figure 4.).

```
#convert data into two columns
filew = open('temp.txt', 'w')
filew.writelines(Lines[line_range[0]:line_range[1]])
filew.close()
data = np.genfromtxt('temp.txt', usecols = (1,3))
```

Figure 4. Preparing data for the plotting.

The creation of variable *plot_obj* allows for getting settings used for plotting.

2.2.2. Plotting results

For displaying results the argumentless function *plot_results* is used.

```
#Plotting results
def plot_results():
    if if_specify.get() == 1:
        x_title = x_title_ent.get()
        x_tab = x_val_ent.get().split()
        x_int = list(map(float, x_tab)) #x values for the plots entered by user

    #user want to specify x title and x values for the plots
    if if_specify.get() == 1 and len(srot0el) == len(srot60el) == len(x_int):
        x_int_sr0 = x_int #x values for plotting RL coefficient for SROT 0
        x_int_sr60 = x_int #x values for plotting RL coefficient for SROT 60
    else: #default values (from 0 to max nb of element)
        if if_specify.get() == 0: #no specified title
            x_title = 'x'
        x_int = range(len(dRL3_tab))
        x_int_sr0 = range(len(srot0el)) #xvalues from 0 to nb of measurements
        x_int_sr60 = range(len(srot60el))
```

Figure 5. Determination of x values.

If the user wants to specify values for the x-axis, the title, and the values will be passed to variables *x_tittle* and *x_tab*. They are both strings, so elements in *x_tab* have to be converted to float numbers. After that, they are saved in the list *x_int*.

If the user doesn't want to specify values, *x_title* is set as 'x' and *x_int* as a list of integers from 0 to the number of the measurements.

Number of ticks entered by user must be equal to the number of measurements for SROT 0 and SROT 60.

```
for el in accepted_p:
    m_nb = el
    w = accepted_p[el][0]
    iter_nb = accepted_p[el][1]
    plotter(m_nb, plt.gca(), 'sub and normalized', w, iter_nb)
```

Figure 6. Plotting all spectra.

The first plot displayed shows all analyzed spectra. For plotting the function *plotter* is used. Measurement numbers and background parameters are received from the *accepted_p* dictionary (Figure 6.).

RL for both SROT and ΔRL coefficients are plotted based on results from appropriate lists: *RL3_0_tab*, *RL3_0_tab*, *dRL3_tab*.

2.3. Background removal

2.3.1. Background fitting algorithm

For background fitting the program uses a strip background model. It is an iterative procedure based on two parameters:

- Strip background width (*Width*)
- Strip background number of iterations (*Iterations*)

At each iteration, if the contents of channel i, y(i), is above the average of the contents of the channels at w channels of distance, y(i-w) and y(i+w), y(i) is replaced by the average. At the end of the process, we are left with something that resembles a spectrum in which the peaks have been "stripped".

2.3.2. Function for background fitting - *background*

```
#calculating background fit
def background(channel, w, iter_nb):
    y_mean = data_y.copy()
    temp = data_y.copy()
    for nb in range(iter_nb):
        for i in range(w, channel-w):
            mean = (temp[i-w] + temp[i+w])/2
            if y_mean[i] > mean:
                temp[i] = mean
        y_mean = temp.copy()
    return y_mean #background calculated based on w and number of iterations
```

Figure 7. Fitting the background – *background* function.

The function has three parameters:

- Number of measurement channels (*channel*)
- Strip background width (*w*)
- Strip background number of iterations (*iter_nb*)

As a first step, two lists are created:

- *y_mean* –list with intensity values for individual measurement channels (*y_mean(i)* in the code is equivalent of y(i) in the description of the algorithm from the previous section)
- *temp* – temporary list for storing data for the duration of loop iteration

Using two for loops strip background procedure is implemented in the function.

2.3.3. Accepting background removal settings – *acc_all* function.

a) No background removal or using the same parameters for all spectra.

If the user doesn't want to subtract the background or use the same background removal parameters for all spectra, he ticks the proper check button which results in calling the appropriate function (*noback* or *oneback*).

In the *acc_all function*, if one of the above options has been selected, all spectra are plotted using for loop and measurement numbers obtained from the list *accepted* (Figure 8.).

```
if if_noback.get() == 1 or if_oneback.get() == 1: #no background removal
    for nb in accepted[0]: #plotting normalized srot 0 measurements
        plotter(nb, ax_calc, 'sub and normalized', w, iter_nb)
    for nb in accepted[1]: #plotting normalized srot 60 measurements
        plotter(nb, ax_calc, 'sub and normalized', w, iter_nb)
```

Figure 8. The first case in the *acc_all* function.

b) Individual selection of background removal parameters for each spectrum.

```
else: #subtract the unique backgroung from every measurement
    for el in accepted_p:
        nb = el
        w = accepted_p[el][0]
        iter_nb = accepted_p[el][1]
        plotter(nb, ax_calc, 'sub and normalized', w, iter_nb) #plotting normalized spectra after bckg removing
ax_calc.set_xlabel('Energy (ev)')
ax_calc.set_ylabel('Intensity')
ax_calc.legend()
canvas_calc.draw()
```

Figure 9. The second case in the *acc_all* function.

If the user wants to select background removal parameters for each spectrum individually procedures after *else* are executed (Figure 9.). The spectra are plotted based on the measurement numbers and background removal parameters obtained from the dictionary *accepted_p*.

2.4. Detection of the peaks

2.4.1. Function peaks_detection

The function has three parameters, including two optional:

- *nb* – measurement number
- *w* (optional) - strip background width
- *iter_nb* (optional) - strip background number of iterations

```
#convert data into two columns
filew = open('temp.txt', 'w')
filew.writelines(Lines[line_range[0]:line_range[1]])
filew.close()
data = np.genfromtxt('temp.txt', usecols = (1,3))

#split imported data to X and Y
global data_x, data_y, no_peaks
data_x.clear()
data_y.clear()
no_peaks = False  #variable for automatic mode exception (false - no exception)
for point in data:
    data_x.append(point[0])
    data_y.append(point[1])

if w and iter_nb: #background is removed
    channel_nb = line_range[1] - line_range[0]
    data_y = list(map(sub, data_y, background(channel_nb, w, iter_nb)))

#normalization
global norm_data_y
norm_data_y = normalization_vec(data_y)
```

Figure 10. Preparing spectra for detection in the *peaks_detection* function.

In the first step, data is converted to a two-column form as in the *plotter* function (3.1). Both columns are saved to the lists: *data_x* (x values) and *data_y* (y values). In the next step, if background removal parameters were provided to the function, the background is subtracted. As a final step of the spectrum preparation for peak detection, spectra are normalized using the *normalization_vec* function.

a) automatic mode

```
#finding peaks
peak, _ = find_peaks(norm_data_y, height = 0.3)

#preparing peaks for plotting
peak_x.clear()
peak_y.clear()
for peak_nb in peak:
    peak_x.append(data_x[peak_nb])
    peak_y.append(norm_data_y[peak_nb])

if mode.get() == 'automatic':
    try:
        RL3_tab.append(RL3_auto(peak_y)) #adding auto-searched peaks to results
    except:
        no_peaks = True #less than 2 peaks found in automatic mode
```

Figure 11. Automatic detection is implemented in the *peaks_detection* function.

To find peaks the function *find_peaks* from the scipy module is used. Founded peaks in the form of channel numbers are saved in the *peak* variable. Next, x and y values for these channels are saved in *peak_x* and *peak_y* lists.

If the automatic mode is on, the function saved RL results in the *RL3_tab* list, unless the program failed to find the peaks. In this case, variable *no_peaks* is set as *True*. This is information for the program that using manual mode is necessary.

b) Manual mode.

If manual mode is on, the function *picking_points*, which is described in the next paragraph, is used.

2.4.2. Function *picking_points*

The function has three parameters, including two optional:

- *nb* – measurement number
- *w* (optional) - strip background width
- *iter_nb* (optional) - strip background number of iterations

In the first step, like in *find_peaks* (5.1.1.), the correctness of automatic peak detection is checked (Figure 12.).

```
no_peaks = False #variable for automatic mode exception (false - no exception)
try:  #2 peaks were found in automatic mode
    a_peaks_x = [peak_x[0],peak_x[-1]]
    a_peaks_y = [peak_y[0],peak_y[-1]]
except:
    no_peaks = True
```

Figure 12. Checking if automatic detection was successful.

In the next step, global variables, which will be used in the *clicking* function, are created (Figure 13.).

```
global actual_nb, actual_w , actual_iter_nb #nb of actual displayed spectra (essential for "clicking" function)
actual_nb = nb
actual_w = w
actual_iter_nb = iter_nb
```

Figure 13. Creating variables necessary for the *clicking* function.

After that, the spectrum with the provided number is plotted together with automatically found peaks (if automatic detection was successful). After that, the function *clicking* connected with the right-clicking event is called (Figure 14.). This function is described in the next paragraph.

```
plotter(nb,ax_detect,'sub and normalized', w, iter_nb) #plotting spectra
#plotting automatic searched-peaks
if not no_peaks:
    ax_detect.plot(a_peaks_x, a_peaks_y, '*', color = 'orange', label='Automatic-searched peaks')
ax_detect.set_xlabel('Energy (eV)')
ax_detect.set_ylabel('Intensity (a.u.)')
ax_detect.set_title('Meusurement number {}'.format(nb))
ax_detect.legend(handles=[legend_auto, legend_man, legend_spectra])

fig_detect.canvas.mpl_connect("button_press_event", clicking) #mark points on the graph
```

Figure 14. The last part of the *picking_points* function.

2.4.3. Function *clicking*

The function allows you to select points on the chart and delete them.

a) Selecting the points

```
if event.button == 1: #picking a point
    closest_val = min(enumerate(data_x), key=lambda x: abs(x[1] - event.xdata)) #closest x value in plot to clicked point
    cl_val_index = closest_val[0]
    x_man.append(data_x[cl_val_index]) #tabs with choosed peaks
    y_man.append(norm_data_y[cl_val_index])
    ax_detect.plot(data_x[cl_val_index], norm_data_y[cl_val_index], '*', color = 'green', label = 'Choosen peak')
    canvas_detect.draw()

    act_xposition.set(str(data_x[cl_val_index])) #display coordiantes of last selected point
    act_yposition.set(str(norm_data_y[cl_val_index]))
```

Figure 15. Selecting points in the *clicking* function.

If some point is right-clicked, the closest x value and its index in the actually displayed spectrum are saved to the *closest_val*. That variable is created through the use of *enumerate*, *min,* and a *lambda* function as a key. The search method is as follows:

- enumerate(data_x): This part of the code uses the enumerate function to iterate through the elements of the data_x along with their corresponding indices. It creates pairs of (index, element) for each element in data_x
- key = lambda x: abs(x[1] - event.xdata): This is a lambda function that defines a custom key for the min function. It takes each pair (index, element) generated by enumerate(data_x) as x and calculates the absolute difference between x[1] and event.xdata (x coordinate of the selected point). This lambda function is used to determine the comparison value for each element in the iterable when finding the minimum
- min(...): The min function is used to find the minimum element from the iterable generated by enumerate(data_x) based on the custom key defined by the lambda function. In other words, it finds the element in data_x that has the smallest absolute difference from event.xdata

In the next step, the point defined through *closest_val* is displayed on the spectra and its coordinates are saved in *x_man* and *y_man* lists. Additionally, coordinates of the point are displayed.

b) Deleting the points

```
elif event.button == 3: #removing last point
    if not no_peaks: #2 peaks were found in automatic mode
        a_peaks_x = [peak_x[0],peak_x[-1]] #automatically found peaks
        a_peaks_y = [peak_y[0],peak_y[-1]]

    ax_detect.clear()
    plotter(actual_nb,ax_detect,'sub and normalized', actual_w, actual_iter_nb) #plotting spectra
    if not no_peaks:
        ax_detect.plot(a_peaks_x, a_peaks_y, '*', color = 'orange', label ='Automatic-searched peaks')
    ax_detect.set_xlabel('Energy (eV)')
    ax_detect.set_ylabel('Intensity (a.u.)')
    ax_detect.set_title('Meusurement number {}'.format(actual_nb))
    ax_detect.legend(handles=[legend_auto, legend_man, legend_spectra])
    #remove one point from choosen peaks tab
    x_man.pop()
    y_man.pop()

    ax_detect.plot(x_man, y_man,'*', color = 'green', label = 'Choosen peaks') #plotting the remaining points
    canvas_detect.draw()
```

Figure 16. Deleting the last point in the *clicking* function.

If the right mouse button is clicked, a spectrum with marked automatically found peaks (if automatic detection was successful) is plotted. Besides, the last chosen point is removed from the *x_man* and *y_man* lists. Finally, peaks from the *x_man* and *y_man* lists are displayed (without the deleted point).

2.4.4. Function *pick_manual*

The function is used in the process of accepting manually chosen peaks for the purpose of saving calculations based on the choices. The function has one parameter:

- *prev_nb* –measurement number for which peaks have been accepted

```
#accepting automatically searched peaks or adding choosen peaks to results
def pick_manual(prev_nb):
    global man_nb
    if len(y_man) == 2: #auto-searched peaks denied
        RL3_tab.append(RL3_man(y_man[0],y_man[1])) #adding manual choosen peaks to results
        man_peaks[prev_nb] = ((x_man[0],y_man[0]), (x_man[1],y_man[1]))
    elif len(y_man) == 0: #auto-searched peaks accepted and 2 peaks found
        try:
            RL3_tab.append(RL3_auto(peak_y)) #adding auto-searched peaks to results
            man_peaks[prev_nb] = ((peak_x[0],peak_y[0]), (peak_x[-1],peak_y[-1]))
        except: ##less than 2 peaks found in automatic mode
            man_nb -= 1
            popup_nopeaks()

    else: #repeat detection for last measurement
        man_nb -= 1
```

Figure 17. Function *pick_manual*.

The global variable *man_nb* is created to provide information on the number of analyzed spectra. It allows the detection mode to end at the appropriate moment and repetition of analysis for the spectra in case of some errors in detection.

a) Rejection of automatically found peaks by the user

If the user chooses two peaks their heights are used for calculations and the results are saved in the *RL3_tab* list. Moreover, the positions of the peaks are saved in the dictionary *man_peaks*.

If the user chooses not exactly two peaks, the pop-up window is displayed and the *man_nb* variable is decremented by one which allows for repetition of the analysis.

b) Acceptance of automatically found peaks by the user

To accept automatically found peaks user should not select any peak. In that case, heights of automatically searched peaks are used for calculations and the results are saved in the *RL3_tab* list. The positions of the peaks are saved in the dictionary *man_peaks*. If the automatic detection was failed, the pop-up window is displayed and *man_nb* variable is decremented by one what allows for repetition of the analysis, instead of this.

3. Crucial variables

3.1. Uploading a file

- *file_path*: Path of file with measurements
- *Lines*: List of all lines from the imported file
- *rscans*: List with numbers of measurements as a function of energy
- *total*: Number of measurements in file
- *data_x*: List of x values for single measurement
- *data_y*: List of x values for single measurement
- *measurements0*: List of chosen measurements (SROT 0) for analysis
- *measurements60*: List of chosen measurements (SROT 60) for analysis

3.2. Background removal

- *accepted*: List of measurements for which background removal parameters are already accepted
- *accepted_p*: Dictionary with measurements after accepting background removal settings. The measurement numbers are the keys and tuples in the form (*width*, *Iterations*, SROT value) are the elements

3.3. Detection and calculations

- *plot_obj*: Axes object, it is used for getting colors of the plots in the program
- *RL3_tab*: List of calculated RL coefficients
- *dRL3_tab*: List of calculated ΔRL coefficients
- *peak_x*: List of x coordinates of automatically found peaks (for single measurement)
- *peak_y*: List of y coordinates of automatically found peaks (for single measurement)
- *x_man*: List of x coordinates of manually chosen peaks (for single measurement)
- *y_man*: List of y coordinates of manually chosen peaks (for single measurement)
- *man_peaks*: Dictionary with positions of manually chosen peaks. The measurement numbers are the keys and the tuples with coordinates of peaks are the elements
- *man_nb*: Integer that provides information on the number of analyzed spectra
- *no_peaks*: Boolean variable that takes the value *True* if automatic detection of peaks has failed

3.4. Results

- *srot0el*: List of tuples. Each tuple contains the measurement number (SROT 0) and value of the RL coefficient)
- *srot60el*: List of tuples. Each tuple contains the measurement number (SROT 60) and value of the RL coefficient)

4. Widgets

4.1. Upload tab

4.1.1. Listboxes

- *ms_list*: List of uploaded measurements
- *selected0*: List of measurements (SROT 0) selected for analysis
- *selected60*: List of measurements (SROT 0) selected for analysis

Each of the lists has its own scrollbar.

4.1.2. Buttons

- *upload_btn*: Opening File Explorer to open file
- *move0* and *move60*: Choice of measurements for analysis, respectively for SROT 0 and SROT 60
- *clear_plot:* Deleting all displayed spectra from the plot
- *new_session_btn*: Starting new session – clearing all containers with results, restoring the program to its initial state
- *acc*: Accepting choices of measurements for analysis and switching to the background tab

4.1.3. Check buttons

- *check_rscan*: If selected, only measurements as a function of energy are displayed in *ms_list*. The value of the check button is stored in a variable *if_only*

4.2. Background removal tab

4.2.1. Listboxes

- *back_list*: List of all measurements chosen for analysis. The scrollbar is assigned to the list.

4.2.2. Buttons

- *sub_btn*: Subtraction of background from chosen spectra and displaying it
- *undo_btn*: Undoing the subtraction and displaying spectra
- *acc_btn*: Saving parameters of background removal for chosen spectra and blocking the possibility of changing them
- *accall_btn*: Accepting background setting for all measurements and switching to the detection tab

4.2.3. Checkbuttons

- *check_noback*: If selected, an option of changing background parameters is disabled and the background will not be subtracted before peak detection. The value of the check button is stored in a variable *if_noback*
- *check_oneback*: If selected, the option of changing background parameters is disabled and the same background will be subtracted from all the spectra before

peak detection. The last selected parameters will be used for that purpose. The value of the check button is stored in a variable *if_oneback*

4.2.4. Spinboxes

- *w_widget*: Changing the value of the strip background width
- *iter_widget*: Changing the value of the strip background number of iterations

4.3. Calculation tab

4.3.1. Buttons

- *detect_btn*: Approval of the calculations settings and switching to the detection tab

4.3.2. Radio buttons

- *mode_man* and *mode_aut*: Selection of the detection mode. The choice is stored in the *mode* variable
- *mode_RL2* and *mode_RL3*: Selection of the RL calculations mode. The choice is stored in the *RL_mode* variable

4.4. Detection window

4.4.1. Buttons

- *acc_man_btn*: Saving choices of peak position in manual mode and displaying the next spectrum
- *go_results_btn*: Approval of the peak positions for all spectra and switching to the results tab
- *repeat_btn*: Clearing containers with detection results and run analysis in manual mode

4.4.2. Entries

- *ent_xposition*: Displaying x coordinate of last selected point in manual mode. The value is stored in *act_xposition*
- *ent_yposition*: Displaying y coordinate of last selected point in manual mode. The value is stored in *act_yposition*

4.5. Results tab

4.5.1. Trees

- *res0_tree*: List of all analyzed measurements for SROT 0 with calculated values of RL coefficient
- *res60_tree*: List of all analyzed measurements for SROT 60 with calculated values of RL coefficient
- *resdRL3_tree*: List of all analyzed corresponding measurements for both SROT with calculated values of ΔRL coefficient for each pair

Each of the lists has its own scrollbar.

4.5.2. Buttons

- *plt_res_btn*: Open a new window with four plots with the results
- *save_file_btn*: Open File Explorer to save a file

4.5.3. Check buttons

- *check_x_value*: If selected, entries allowing providing of additional data for the plots (x-axis title and ticks) are displayed. The value of the check button is stored in a variable *if_specify*

4.5.4. Entries

- *x_title_ent*: Providing title of x-axis for the results plots
- *x_val_ent*: Providing ticks for x-axis for the results plots