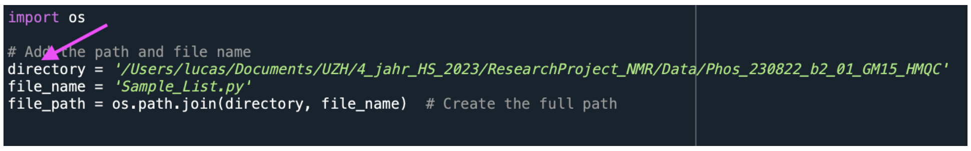
Semi-Automated Data-Handling of a Phosphorylation Assay

Lucas Kirchen

Phosphorylation assays by means of NMR can require a very time-consuming processing due to the large amount of data. With this workflow and the associated scripts "Data_Sorting" and "Phosphorylation_Processing", a first approach was created to make this process more time-efficient.

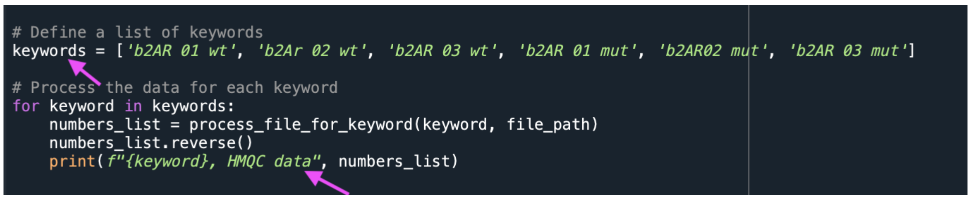
Data Sorting

In order to process the data easily, especially if one has measured several phosphrylations in parallel, it is advantageous to break them down. For this purpose, the ’Sample_List.py’ file which was generated when creating the measurement series and the script ’Data_Sorting.py’ can be used. For this purpose, the path to the ’Sample_List.py’ file must be given for the variable ’Directory’ Figure1, and the keywords used for sorting and the type of measurement (e.g. HMQC, HSQC, ...) must be specified.
This gives as output a list Figuren3 which corresponds to the numbers of the relevant measurement.



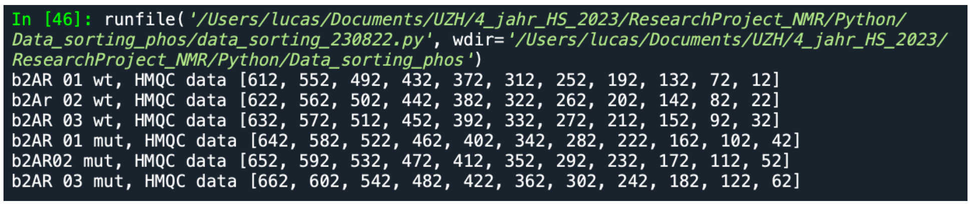
```
import os

# Add the path and file name
directory = '/Users/lucas/Documents/UZH/4_jahr_HS_2023/ResearchProject_NMR/Data/Phos_230822_b2_01_GM15_HMQC'
file_name = 'Sample_List.py'
file_path = os.path.join(directory, file_name)  # Create the full path
```

Figure 1: The path to the ’Sample_List.py’ must be given here.



```
# Define a list of keywords
keywords = ['b2AR 01 wt', 'b2Ar 02 wt', 'b2AR 03 wt', 'b2AR 01 mut', 'b2AR02 mut', 'b2AR 03 mut']

# Process the data for each keyword
for keyword in keywords:
    numbers_list = process_file_for_keyword(keyword, file_path)
    numbers_list.reverse()
    print(f"{keyword}, HMQC data", numbers_list)
```

Figure 2: The keywords, which are the different phosphorylation assays, need to be defined here.



```
In [46]: runfile('/Users/lucas/Documents/UZH/4_jahr_HS_2023/ResearchProject_NMR/Python/
Data_sorting_phos/data_sorting_230822.py', wdir='/Users/lucas/Documents/UZH/4_jahr_HS_2023/
ResearchProject_NMR/Python/Data_sorting_phos')
b2AR 01 wt, HMQC data [612, 552, 492, 432, 372, 312, 252, 192, 132, 72, 12]
b2Ar 02 wt, HMQC data [622, 562, 502, 442, 382, 322, 262, 202, 142, 82, 22]
b2AR 03 wt, HMQC data [632, 572, 512, 452, 392, 332, 272, 212, 152, 92, 32]
b2AR 01 mut, HMQC data [642, 582, 522, 462, 402, 342, 282, 222, 162, 102, 42]
b2AR02 mut, HMQC data [652, 592, 532, 472, 412, 352, 292, 232, 172, 112, 52]
b2AR 03 mut, HMQC data [662, 602, 542, 482, 422, 362, 302, 242, 182, 122, 62]
```

Figure 3: The output is given as a list.

Topspin Processing

In Topspin the desired peaks have to be picked, this can be done with each measurement manually or automatically. This can be done by selecting [Process Dataset List (serial)] under [Advanced] Figure 4.

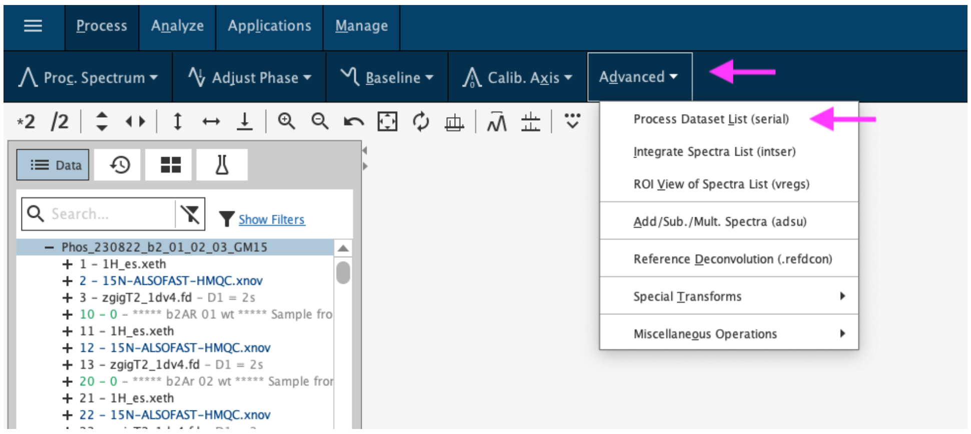

Process
Analyze
Applications
Manage
Proc. Spectrum
Adjust Phase
Baseline
Calib. Axis
Advanced
Process Dataset List (serial)
Integrate Spectra List (intser)
ROI View of Spectra List (vregs)
Add/Sub./Mult. Spectra (adsu)
Reference Deconvolution (.refdcon)
Special Transforms
Miscellaneous Operations
Data
Search...
Show Filters
Phos_230822_b2_01_02_03_GM15
1 - 1H_es.xeth
2 - 15N-ALSOFAST-HMQC.xnov
3 - zgigT2_1dv4.fd - D1 = 2s
10 - 0 - ***** b2AR 01 wt ***** Sample fro
11 - 1H_es.xeth
12 - 15N-ALSOFAST-HMQC.xnov
13 - zgigT2_1dv4.fd - D1 = 2s
20 - 0 - ***** b2Ar 02 wt ***** Sample fror
21 - 1H_es.xeth
22 - 15N-ALSOFAST-HMQC.xnov

Figure 4: The output is given as a list.

Now the data sets that are to be processed must be selected, these should be those that were sorted in the previous step. To simplify this selection, it is easiest to create a new folder containing only these measurements. The list can be generated under [Select Dataset List] and then [Build dataset list using "find"] Figure 5. This list can be edited and saved under [Edit Dataset List]. If a list already exists, it can be selected under [Select Dataset List].

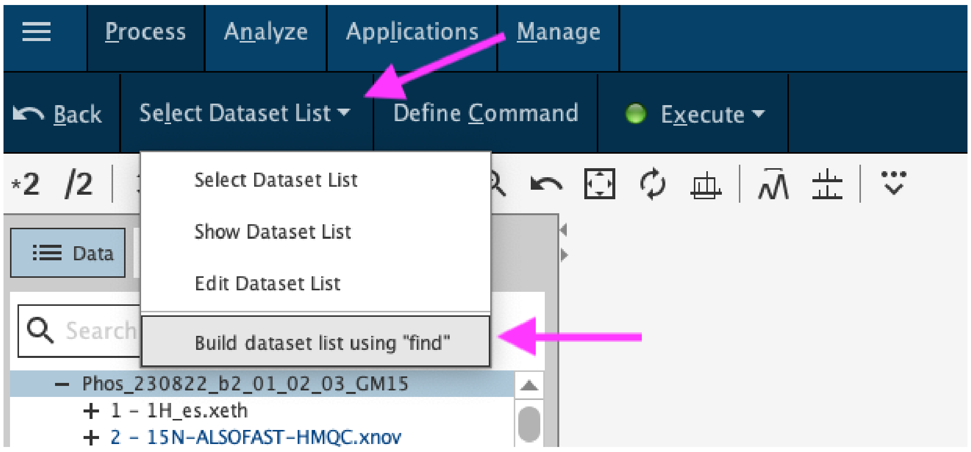


Figure 5: The measurements for which pick picking is to take place must be selected here.

Now the respective commands can be defined which topspin should execute Figure 6. In this case, this corresponds to the region of F1 and F2 in which the peak picking is to be performed Figure 7. After the command has been defined, [Execute] must be clicked to execute the command. To pick the peaks, the command 'pp' can be defined. However, this has the disadvantage that the options window is opened for each spectrum and must be confirmed manually. To avoid this, 'pp nodia' can be used, but important options such as minimum intensity can no longer be defined.

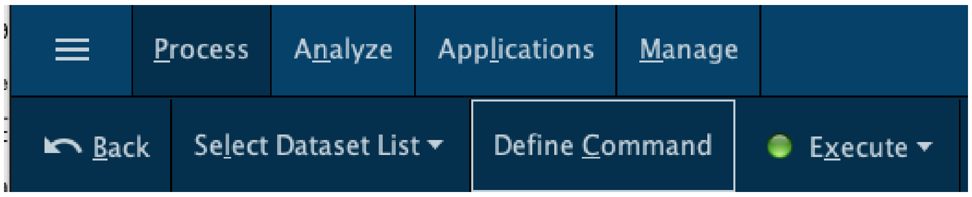
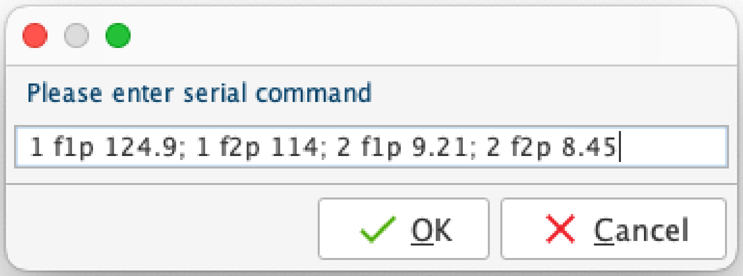


Figure 6: The commands need to be defined here and can then be executed.

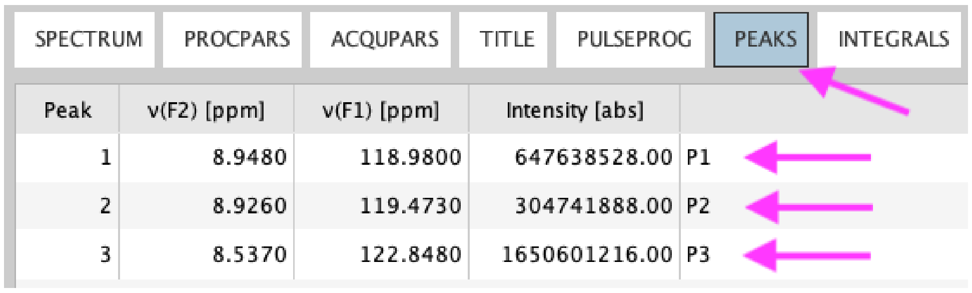


Please enter serial command

```
1 f1p 124.9; 1 f2p 114; 2 f1p 9.21; 2 f2p 8.45
```

Figure 7: The limit must be specified if it is not desired to peak over the entire spectrum.

After the peaks have been picked, however, the annotation must now be entered manually for each peak. This is important because the code later depends on the presence of the annotation column in the xml file. Which name is chosen here is only relevant for the last spectrum, as all others are adjusted on this one. For the others, there must simply be something there.

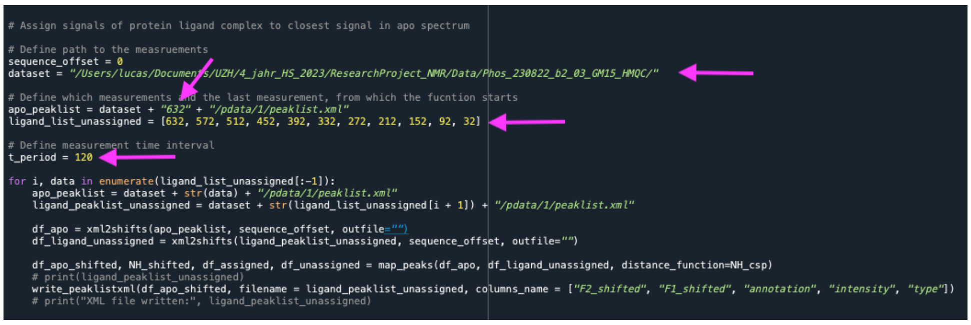


Peak	ν(F2) [ppm]	ν(F1) [ppm]	Intensity [abs]	
1	8.9480	118.9800	647638528.00	P1
2	8.9260	119.4730	304741888.00	P2
3	8.5370	122.8480	1650601216.00	P3

Figure 8: It is crucial that an annotation is given for every peak.

Data Processing in Python

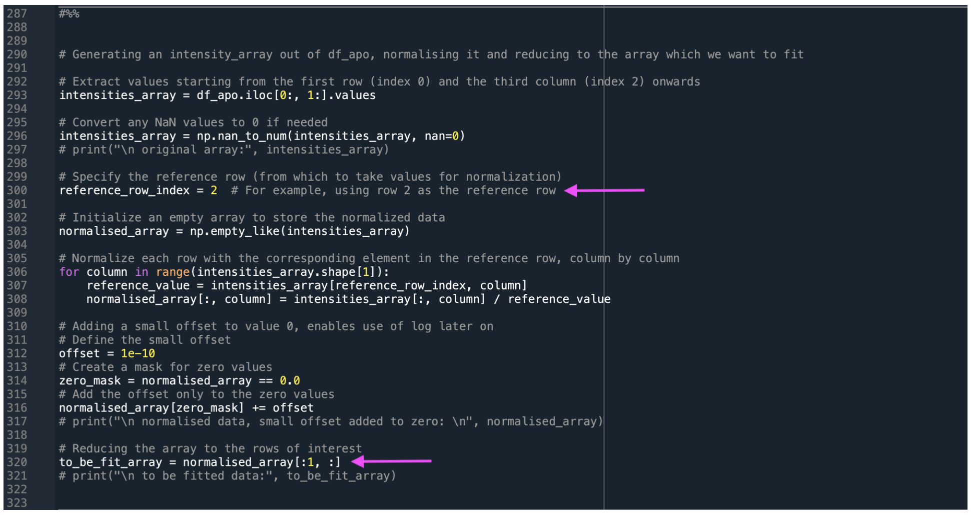
To process the data further, the script "Phosphorylation_Processing" can be used. Only a few changes need to be made in the code. First, the path to the spectra must be given, then the number of the number from which to start (usually the last measurement) must be changed and then the list with the spectra must be updated. This corresponds to the lists which can be generated using the 'Data_Sorting.py' script. In addition, the time interval in which the measurements were taken must be defined in the variable t_period. Figure 9



```python
# Assign signals of protein ligand complex to closest signal in apo spectrum

# Define path to the measruements
sequence_offset = 0
dataset = "/Users/lucas/Documents/UZH/4_jahr_HS_2023/ResearchProject_NMR/Data/Phos_230822_b2_03_GM15_HMQC/"

# Define which measurements and the last measurement, from which the fucntion starts
apo_peaklist = dataset + "632" + "/pdata/1/peaklist.xml"
ligand_list_unassigned = [632, 572, 512, 452, 392, 332, 272, 212, 152, 92, 32]

# Define measurement time interval
t_period = 120

for i, data in enumerate(ligand_list_unassigned[:-1]):
    apo_peaklist = dataset + str(data) + "/pdata/1/peaklist.xml"
    ligand_peaklist_unassigned = dataset + str(ligand_list_unassigned[i + 1]) + "/pdata/1/peaklist.xml"

    df_apo = xml2shifts(apo_peaklist, sequence_offset, outfile="")
    df_ligand_unassigned = xml2shifts(ligand_peaklist_unassigned, sequence_offset, outfile="")

    df_apo_shifted, NH_shifted, df_assigned, df_unassigned = map_peaks(df_apo, df_ligand_unassigned, distance_function=NH_csp)
    # print(ligand_peaklist_unassigned)
    write_peaklistxml(df_apo_shifted, filename = ligand_peaklist_unassigned, columns_name = ["F2_shifted", "F1_shifted", "annotation", "intensity", "type"])
    # print("XML file written:", ligand_peaklist_unassigned)
```

Figure 9: The path to the file as well as the names of the spectra need to be defined here. Furthermore, the time interval of the measurements needs to be specified, in this example an interval of 120 minutes was used.

Since the data is to be normalised, the row containing the normalised values must be selected. As the normalised values are not to be plotted, the array must be reduced to the relevant values, this must also be defined here (second arrow), Figure 10.



```python
#%%


# Generating an intensity_array out of df_apo, normalising it and reducing to the array which we want to fit

# Extract values starting from the first row (index 0) and the third column (index 2) onwards
intensities_array = df_apo.iloc[0:, 1:].values

# Convert any NaN values to 0 if needed
intensities_array = np.nan_to_num(intensities_array, nan=0)
# print("\n original array:", intensities_array)

# Specify the reference row (from which to take values for normalization)
reference_row_index = 2  # For example, using row 2 as the reference row

# Initialize an empty array to store the normalized data
normalised_array = np.empty_like(intensities_array)

# Normalize each row with the corresponding element in the reference row, column by column
for column in range(intensities_array.shape[1]):
    reference_value = intensities_array[reference_row_index, column]
    normalised_array[:, column] = intensities_array[:, column] / reference_value

# Adding a small offset to value 0, enables use of log later on
# Define the small offset
offset = 1e-10
# Create a mask for zero values
zero_mask = normalised_array == 0.0
# Add the offset only to the zero values
normalised_array[zero_mask] += offset
# print("\n normalised data, small offset added to zero: \n", normalised_array)

# Reducing the array to the rows of interest
to_be_fit_array = normalised_array[:1, :]
# print("\n to be fitted data:", to_be_fit_array)
```

Figure 10: The row which is going to be used for the normalisation needs to be defined here and the array reduced to the data which is going to be plotted and fitted.

As an output one should now obtain the plot with the fit as well as the parameter *k* and the R^2.