MultiNMRFit

Release 2.1.3

Pierre Millard, Cyril Charlier

Mar 26, 2026

USAGE

1 Welcome to MultiNMRFit documentation! 1

1.1 Quick start 1
1.2 Tutorial 4
1.3 Models 11
1.4 How to cite 16
1.5 Frequently asked questions (FAQ) 17
1.6 Library documentation 18
1.7 License 27

Python Module Index 28

Index 29

CHAPTER

ONE

WELCOME TO MULTINMRFIT DOCUMENTATION!

MultiNMRFit is a scientific software dedicated to the analysis of NMR data. It is one of the routine tools that we use at the NMR team and MetaSys team of Toulouse Biotechnology Institute.

The code is open-source, and available on GitHub under a *GPLv3 license*.

This documentation is available on Read the Docs (https://multinmrfit.readthedocs.io) and can be downloaded as a PDF file.

Key features

- **fit series of 1D spectra** (acquired as individual 1D spectra, as a pseudo 2D spectrum, or provided as tabulated text files),
- can be used with **all nuclei** (^{1}H, ^{13}C, ^{15}N, ^{31}P, etc),
- estimation of several parameters for each signal of interest (**intensity**, **area**, **chemical shift**, **linewidth**, **coupling constant(s)**, etc),
- **semi-automated analysis** for **peak picking** and **definition of multiplicity** for each signal,
- account for **overlaps** between peaks and **zero-order baseline correction**,
- **visual inspection of the fitted curves**,
- estimation of **uncertainty** on estimated parameters (standard deviation),
- shipped as a **library** with a **graphical user interface**,
- open-source, free and easy to install everywhere where Python 3 and pip run,
- biologist-friendly.

See Also

We strongly encourage you to read the *Tutorial* before using MultiNMRFit.

1.1 Quick start

1.1.1 Installation

MultiNMRFit requires Python 3.8 or higher. If you do not have a Python environment configured on your computer, we recommend that you follow the instructions from Anaconda.

Then, open a terminal (e.g. run *Anaconda Prompt* if you have installed Anaconda) and type:

```
pip install multinmrfit
```

You are now ready to start MultiNMRFit.

If this method does not work, you should ask your local system administrator or the IT department "how to install a Python 3 package from PyPi" on your computer.

Alternatives & update

If you know that you do not have permission to install software systemwide, you can install MultiNMRFit into your user directory using the `--user` flag:

```
pip install --user multinmrfit
```

If you already have a previous version of MultiNMRFit installed, you can upgrade it to the latest version with:

```
pip install --upgrade multinmrfit
```

Alternatively, you can also download all sources in a tarball from GitHub, but it will be more difficult to update Multi-NMRFit later on.

1.1.2 Usage

Graphical User Interface

To start the Graphical User Interface, type in a terminal (Windows: *Anaconda Prompt*):

```
nmrfit
```

The MultiNMRFit window will open. If the window fails to open, have a look at our dedicated troubleshooting procedure to solve the problem.

Welcome to multiNMRFit (v2.0.0bDev1)

Data to process

Select data type

pseudo2D

Inputs

Enter data path

path/to/data

Enter data folder

data_folder

Enter Expno

1

Enter Procno

1

Outputs

Enter output path

path/to/results

Enter output folder

results_folder

Enter filename

filename

Load spectrum

The main processing steps can be performed via the menu on the left side bar:

- **Inputs & Outputs**: information required to load the data to process and export results (type of data, input and output directories, etc)
- **Process spectra**: process one or several signal(s) of specific spectra
- **Process from reference**: process a serie of spectra as done on a given spectrum (used as reference)
- **Results visualisation**: view and export processing results

Details on MultiNMRFit usage can be found in the tutorial section.

Note

The process is continuously and automatically saved as a pickle file in the output folder. To reopen the current processing state, just reopen this file by clicking on "Load a processing file - Browse files" on the side bar at the

left.

Warning

MultiNMRFit silently overwrites (results and processing) files if they already exist. So take care to copy your results elsewhere or to change the output path and/or filename if you want to protect them from overwriting.

Library

MultiNMRFit is also available as a library (a Python module) that you can import directly in your Python scripts:

```
import multinmrfit
```

See also

Have a look at our *API* if you are interested in this feature.

1.2 Tutorial

See also

If you have a question that is not covered in the tutorials, have a look at the faq or please contact us.

This tutorial will guide you through the different pages of MultiNMRFit.

1.2.1 1. Load spectra

Format of input data

MultiNMRFit requires that the main spectrum processing steps (baseline correction, phasing, . . .) have been performed beforehand. MultiNMRFit can load 1D NMR data provided in the following formats:

- **Pseudo2D**: pseudo2D experiment (Bruker format only),
- **list of 1Ds**: 1Ds spectra acquired independently (Bruker format only),
- **txt data**: data from a tabulated text file (`.txt` extension) with the following structure:

ppm	0	. . .	n
0	1.2e3	. . .	1.2e6
0.1	1.3e3	. . .	4e7
0.2	2e8	. . .	3.6e3
. . .	. . .	. . .	. . .
12	3e4	. . .	7.85e3

The column **ppm** is mandatory and contains the ppm scale, columns named **0** to **n** correspond to each individual spectra and contain the intensities.

> **Note**
>
> **list of 1Ds**: The list of experiments should be provided as e.g.
>
> - 1,8,109 : for non-consecutive spectra (here spectra 1, 8, and 109)
> - 1-5 : for sequential spectra (here spectra 1,2,3,4,5)
> - 1-5,109 : a mix of both formats for incomplete series (here spectra 1,2,3,4,5,109)

Inputs & outputs paths

data_path
: Path to the directory that contain the data

data_folder
: Folder containing your NMR data

expno
: List of Experiments which contain the spectra (i.e. expno in Topspin)

procno
: Process number (i.e. procno in Topspin)

> **Note**
>
> **Inputs**: The different fields will for inputs as described above will appear only for data type **Pseudo2D** & **list of 1Ds**. For **txt data**, the text file must be loaded using the drag-and-drop menu.

> **Note**
>
> **procno**: If a list of **expno** is provided the **procno** must be same for all **expnos**.

output_path
: Path to the folder use to export the outputs

output_folder
: Folder name

filename
: Name of the pickle file containing the process that will be automatically saved

Load a processing file

Current status of the process is continuously saved in a pickle file containing the entire process that has been perfomed. The pickle file can be reloaded using the drag-and-drop menu available in side bar of the Inputs & Outputs page.

1.2.2 2. Fit a spectrum

Once the data are correctly loaded the second page of the interface becomes available and allows users to fit one or several signals contained in a specific region of a given spectrum:

Process reference spectrum

Select spectrum to process: 1

Select region to (re)process: Add new region

Spectral limits (max): 3.35

Spectral limits (min): 3.20

Peak picking & Clustering

Peak picking threshold: 103712403.20

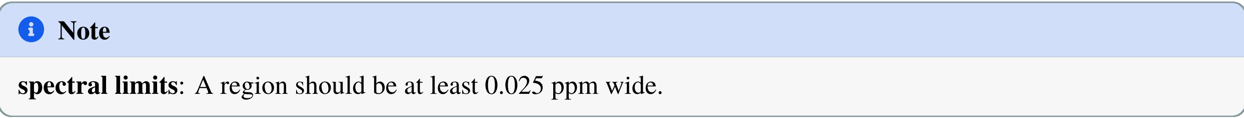
Peak list

peak position	peak intensity	cluster ID
3.242	208,597,760	1
3.2526	377,093,440	1
3.2632	234,149,216	1
3.277	282,441,184	2
3.2882	518,562,016	2
3.2989	309,316,160	2

Assign peaks

The top part of this page automatically performs peak picking on the reference spectrum within the region displayed in the plot:

- **Select spectrum**: Select one the spectrum of the list.
- **Select region to (re)process**: Multiple independent regions can be processed. Here, it will give you the possibility to display regions already processed or to define a new region to process.
- **Spectral limits (max)**: Maximal chemical shift of the spectral region (default is the maximum of the ppm scale)
- **Spectral limits (min)**: Minimal chemical shift of the spectral region (default is the min of the ppm scale)



Note

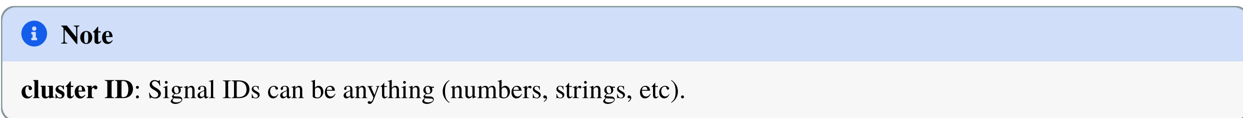
spectral limits: A region should be at least 0.025 ppm wide.

You can adjust the **Peak picking threshold** to detect the peaks of interest on the displayed spectrum.

While adjusting this threshold the software will automatically display a dataframe **Peak list** with the detected peaks in the region (marked with a yellow triangle on the spectrum). The peaks are displayed in the ascending order (e.g. from

right to left on the spectrum). You can also add peaks mannually by entering their chemical shifts and click on "Add peak".

You can now proceed with the clustering steps that consists in grouping peaks into a signal. For this purpose, fill the **cluster ID** column of the **Peak list**. Peaks that belongs to the same multiplets must have the same names.

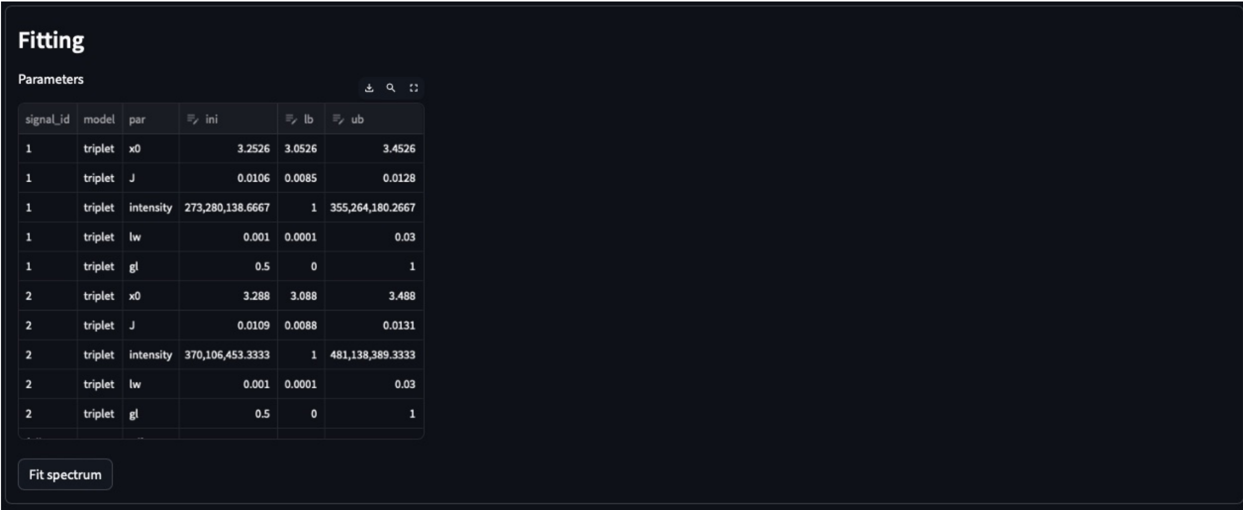


Note

cluster ID: Signal IDs can be anything (numbers, strings, etc).

Once clustering has been performed, click **Assign peaks** to move towards the model construction step:

For each signal, MultiNMRFit will provide a list of all signal models that correspond to can be used (i.e. all signal models containing the corresponding number of peaks). You can also choose to add an offset, which corresponds to a first-order phase correction on the selected window. Once this step is done, you can click on **Build model** to automatically create the spectrum model and display the table of initial parameters.



signal_id	model	par	ini	lb	ub
1	triplet	x0	3.2526	3.0526	3.4526
1	triplet	J	0.0106	0.0085	0.0128
1	triplet	intensity	273,280,138.6667	1	355,264,180.2667
1	triplet	lw	0.001	0.0001	0.03
1	triplet	gl	0.5	0	1
2	triplet	x0	3.288	3.088	3.488
2	triplet	J	0.0109	0.0088	0.0131
2	triplet	intensity	370,106,453.3333	1	481,138,389.3333
2	triplet	lw	0.001	0.0001	0.03
2	triplet	gl	0.5	0	1

Initial values are calculated based on [i] the results of the peak picking (intensities and peak position) [ii] the default parameters of the each model (look at models.rst for more details on the default parameters). If no changes are required press the **Fit spectrum** button to fit the spectrum.

Note

Parameters: All parameters are shown in **ppm** units.

Fitting

Parameters

signal_id	model	par	ini	lb	ub
1	triplet	x0	3.2526	3.0526	3.4526
1	triplet	J	0.0106	0.0085	0.0128
1	triplet	intensity	273,280,138.6667	1	355,264,180.2667
1	triplet	lw	0.001	0.0001	0.03
1	triplet	gl	0.5	0	1
2	triplet	x0	3.288	3.088	3.488
2	triplet	J	0.0109	0.0088	0.0131
2	triplet	intensity	370,106,453.3333	1	481,138,389.3333
2	triplet	lw	0.001	0.0001	0.03
2	triplet	gl	0.5	0	1

Fit spectrum

exp. spectrum initial values best fit residuum

intensity

residuum

chemical shift (ppm)

Spectrum has been fitted.

Add current region

The fitted spectrum will be automatically displayed on the resulting plot. This plot shows [i] the experimental data as dots [ii] the best fit as red a curve and [iii] the initial values in green. The residuals plot (i.e. difference between the fitted and the experimental spectra) is shown below.

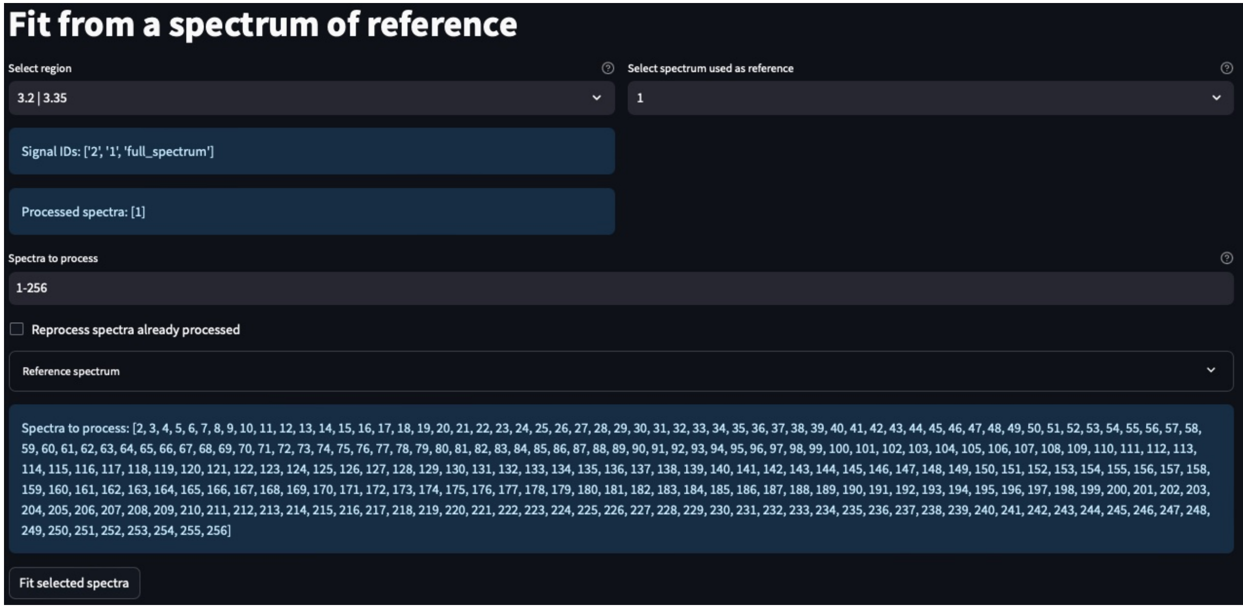
Note

Parameters: In the case of mismatch between the data and the best fit, you can adjust manually adjust the initial values in the former **parameters** table.

If the results are satisying, click on **Add current region** to save this region. To add another region, go to the top of page and select **add new region** in the field **Select region to (re)process**. Otherwise move to next page **Fit from reference**.

1.2.3 3. Batch analysis

This page contains the wrapper that enables fitting several spectra in batch based on an already processed spectrum (used as reference).



First select the region that you want to fit (**Select region**). MultiNMRFit will display the list of **Signal IDs** present in the selected region along with the **processed spectra** already analyzed.

Select the spectra you want to fit. By default it shows the complete dataset (here 1-256 as the pseudo2D contains 256 in the example). However if you want to analyze only the first ten spectra one can enter '1-10' and MultiNMRFit will update the list **spectra to process** automatically. Click the **Fit selected spectra** to run the fitting of the selected spectra. The progress of the fitting will be displayed by a progress bar and once complete a message **All spectra have been fitted** will appear.

Note

Fitting: This procedure can be repeated for the different regions defined in the previous pages upon selection in **Select region**. By default MultiNMRFit do not reprocess spectra that have been already been fitted so clicked the option if necessary. The reference spectrum associated with the selected region can be visualized on this page.

1.2.4 4. Results visualisation and export

This page enables visualizing the processing results in interactive plots. On top, you can inspect all fitted regions and spectra. If multiple signals were fitted on the same region, you can observe each one by clicking on the different signal IDs in the figure caption.

Spectra visualisation

Users can select the spectrum and the region to display.

Parameters visualisation

For the corresponding spectra shown above users can find the table of parameters. A particular attention must me given to the **opt** column that contains the values estimated from the best fit.

Parameters

signal_id	model	par	ini	lb	ub	opt	opt_sd	integral
1	triplet	x0	3.252585	3.052585	3.452585	3.252662	0.000021	4704194.511116
1	triplet	J	0.010629	0.008504	0.012755	0.010786	0.000040	4704194.511116
1	triplet	intensity	273280138.666667	1.000000	355264180.266667	195309442.400065	1061182.371098	4704194.511116
1	triplet	lw	0.001000	0.000100	0.030000	0.002171	0.000011	4704194.511116
1	triplet	gl	0.500000	0.000000	1.000000	0.421393	0.038608	4704194.511116
2	triplet	x0	3.288016	3.088016	3.488016	3.288015	0.000020	6480995.999202
2	triplet	J	0.010942	0.008754	0.013130	0.010727	0.000052	6480995.999202
2	triplet	intensity	370106453.333333	1.000000	481138389.333333	267387616.487518	2280498.653494	6480995.999202
2	triplet	lw	0.001000	0.000100	0.030000	0.002130	0.000010	6480995.999202
2	triplet	gl	0.500000	0.000000	1.000000	0.531790	0.016501	6480995.999202

Finally, users can observe the change of a given paramters as function of spectra IDs.

Export results

Users can export their results tabulated text file in two different manners: **all data** or **specific data** In the first case (**all data**) all the parameters of all the regions and spectra will be saved in the **output** location defined in the first page of the interface. If the second case (option **specific data** selected), you can select one region, one parameter that will exclusively saved in the file.

1.2.5 Warning and error messages

Error messages are explicit. You should examine carefully any warning/error message. After correcting the problem, perform the analysis again.

1.3 Models

1.3.1 Models shipped with multiNMRFit

Signal models for some typical multiplets (mixed Gaussian-Lorentzian models of singlet, doublet, triplet, quadruplet and doublet of doublet) are included in multiNMRFit.

The models used in MultiNMRFit can be found in the models folder, which is located in the multinmrfit package. To find the path to the multinmrfit package, you can use the following command in a Python console:

```
import multinmrfit
print(multinmrfit.__path__)
```

All models follow the same format. Have a look to model_singlet.py for a detailed example.

1.3.2 User-made models

Overview

multiNMRFit is designed to be easily extended with new signal models. Users can create their own signal models and add them to the list of available models.

This section explains how to write a model and how to implement it on your multiNMRFit instance. All models follow the same format, have a look to model_singlet.py as an example.

Build a template

To implement user-made models, multiNMRFit leverages Python's object model to create classes that inherit from an Abstract Base Class and that handles all the heavy-lifting for implementation. A simple set of rules enables users to use their model in multiNMRFit.

The model must be a class located in a dedicated module. Start by opening a text file using your IDE (Integrated Development Environment), and enter the following structure in the file:

```
import numpy as np
from multinmrfit.models.base_model import Model

class SignalModel(Model):

    def __init__(self, data):
        self.name = "model name"
        self.description = "model description"
        self.peak_number = int
        self.default_params = {}

    def pplist2signal(self, peak_list):
        pass

    @staticmethod
    def simulate():
        pass

if __name__ == "__main__":
    pass
```

This is the base template to build your model, which includes the following methods:

- **__init__**: initialize the signal model object.
- **pplist2signal**: the function that will be used to built the signal from the peak list. It should return a dictionary containing the name of the signal, and optionaly some parameter values to be updated based on the peak list (if different from the default values).
- **simulate**: the function that will be used to simulate the signal. It should return the simulated signal given the parameters and chemical shifts.

Additional methods are allowed if needed (e.g. to carry out intermediary steps for the simulation).

Populate the template

The first attribute to add in your model's `__init__` method is the model name.

This method should include the following attributes:

- **name**: the name of the model. It should be unique and descriptive.
- **description**: a short description of the model.
- **peak_number**: the number of peaks in the model. This is used to determine how many peaks are contained in the signal.
- **default_params**: a dictionary containing the default parameters for the model. The keys should be the names of the parameters, and the values should be their default values. The parameters are defined as follows:
  - **model**: name of the model

- **par**: name of the parameters
- **ini**: default value of the parameter
- **lb**: lower bound of the parameter
- **ub**: upper bound of the parameter
- **shift_allowed**: window of allowed shift for the parameter in comparison to the previous spectrum used as reference. This is used to dynamically adapt the bounds from a spectrum to the next one during the fitting process in batch.
- **relative**: boolean indicating if the shift allowed is expressed as a relative or absolute value. If True, the shift is relative to the parameter value. If False, the shift is defined as absolute value.

For instance, in the case of a mixed Gaussian-Lorentzian doublet model, the default parameters are:

```
import numpy as np
from multinmrfit.models.base_model import Model

class SignalModel(Model):

    def __init__(self):
        self.name = "doublet"
        self.description = "mixed gaussian-lorentzian doublet"
        self.peak_number = 2
        self.default_params = {'model': [self.name]*5,
                               'par': ['x0', 'J', 'intensity', 'lw', 'gl'],
                               'ini': [1.0, 0.05, 1e6, 0.001, 0.5],
                               'lb': [0.0, 0.01, 1, 0.0001, 0.0],
                               'ub': [10.0, 1.0, 1e15, 0.03, 1.0],
                               'shift_allowed': [0.01, 0.10, 10, 0.3, 10],
                               'relative': [False, True, True, True, False]}

    def pplist2signal(self, peak_list):
        pass

    @staticmethod
    def simulate(params, ppm):
        pass

if __name__ == "__main__":
    pass
```

The second method to implement is the `pplist2signal` method. This method is used to convert a peak list into a signal. It should return a dictionary containing the name of the signal, and optionaly some parameter values to be updated based on the peak list (if different from the default values). The dictionary should contain the following keys:

- **model**: name of the signal model
- **par**: dictionary containing the parameters of the signal. The keys should be the names of the parameters, and the values should be a dictionary containing their values and lower and upper bounds.

For instance, in the case of a mixed Gaussian-Lorentzian doublet model, the signal is built as follows:

```
import numpy as np
from multinmrfit.models.base_model import Model
```

(continues on next page)

(continued from previous page)

```
class SignalModel(Model):

    def __init__(self):
        self.name = "doublet"
        self.description = "mixed gaussian-lorentzian doublet"
        self.peak_number = 2
        self.default_params = {'model': [self.name]*5,
                               'par': ['x0', 'J', 'intensity', 'lw', 'gl'],
                               'ini': [1.0, 0.05, 1e6, 0.001, 0.5],
                               'lb': [0.0, 0.01, 1, 0.0001, 0.0],
                               'ub': [10.0, 1.0, 1e15, 0.03, 1.0],
                               'shift_allowed': [0.01, 0.10, 10, 0.3, 10],
                               'relative': [False, True, True, True, False]}

    def pplist2signal(self, peak_list):
        # Estimate the chemical shift of the center of the doublet as the mean of␣
↪chemical shift between the two peaks
        detected_peak_position = np.mean(peak_list.ppm.values)
        # Estimate the intensity of the doublet as the intensity of the first peak
        detected_peak_intensity = peak_list.intensity.values[0]
        # Estimate the coupling constant as the absolute difference between the two peaks
        detected_coupling_constant = np.abs(max(peak_list.ppm.values)-min(peak_list.ppm.
↪values))
        # Build the signal dictionary with the parameters and bounds
        signal = {
            "model": self.name,
            'par': {'x0': {'ini': detected_peak_position, 'lb': detected_peak_position-1,
↪ 'ub': detected_peak_position+1},
                    'intensity': {'ini': detected_peak_intensity, 'ub': 1.1*detected_
↪peak_intensity},
                    'J': {'ini': detected_coupling_constant, 'lb': 0.8*detected_coupling_
↪constant, 'ub': 1.2*detected_coupling_constant},
                    }
        }

    return signal

    @staticmethod
    def simulate(params, ppm):
        pass

if __name__ == "__main__":
    pass
```

Finally, the last method to implement is the `simulate` method. This method is used to simulate the signal. It should return the simulated signal given the parameters and chemical shifts. The method should take as input the parameters and chemical shifts, and return the simulated signal.

```
import numpy as np
from multinmrfit.models.base_model import Model

class SignalModel(Model):
```

(continues on next page)

(continued from previous page)

```
    def __init__(self):
        self.name = "doublet"
        self.description = "mixed gaussian-lorentzian doublet"
        self.peak_number = 2
        self.default_params = {'model': [self.name]*5,
                               'par': ['x0', 'J', 'intensity', 'lw', 'gl'],
                               'ini': [1.0, 0.05, 1e6, 0.001, 0.5],
                               'lb': [0.0, 0.01, 1, 0.0001, 0.0],
                               'ub': [10.0, 1.0, 1e15, 0.03, 1.0],
                               'shift_allowed': [0.01, 0.10, 10, 0.3, 10],
                               'relative': [False, True, True, True, False]}

    def pplist2signal(self, peak_list):
        detected_peak_position = np.mean(peak_list.ppm.values)
        detected_peak_intensity = peak_list.intensity.values[0]
        detected_coupling_constant = np.abs(max(peak_list.ppm.values)-min(peak_list.ppm.
↪values))

        signal = {
            "model": self.name,
            'par': {'x0': {'ini': detected_peak_position, 'lb': detected_peak_position-1,
↪ 'ub': detected_peak_position+1},
                    'intensity': {'ini': detected_peak_intensity, 'ub': 1.1*detected_
↪peak_intensity},
                    'J': {'ini': detected_coupling_constant, 'lb': 0.8*detected_coupling_
↪constant, 'ub': 1.2*detected_coupling_constant},
                    }
        }

    return signal

    @staticmethod
    def simulate(params, ppm):
        # Peak #1, at chemical shift x0 - J/2
        peak_1 = params[4] * params[2] / (1 + ((ppm - params[0] - (params[1]/2))/
↪params[3])**2) + (1-params[4]) * \
            params[2]*np.exp(-(ppm - params[0] - (params[1]/2))**2/(2*params[3]**2))
        # Peak #2, at chemical shift x0 + J/2
        peak_2 = params[4] * params[2] / (1 + ((ppm - params[0] + (params[1]/2))/
↪params[3])**2) + (1-params[4]) * \
            params[2]*np.exp(-(ppm - params[0] + (params[1]/2))**2/(2*params[3]**2))

    return peak_1 + peak_2

if __name__ == "__main__":
    pass
```

Test the model

We can now check that the model can be initialized properly. Use the block at the end of the file for testing purposes. Here is an example of how you can test the model:

```
if __name__ == "__main__":

    model = SignalModel()
    print(model.name)
    print(model.description)
    print(model.peak_number)
    print(model.default_params)
```

If you now run the file, you should have a standard output in your console that contains the name of the model, its description, the number of peaks and the default parameters. If you have an error message, check the code and correct it.

The last step is to simulate a spectra with your model. This can be done using the following code:

```
if __name__ == "__main__":

    model = SignalModel()
    x_ppm = np.arange(0.0, 10.0, 0.01)
    spectrum = model.simulate(model.default_params, x_ppm)
    print(spectrum)
```

If you now run the file, you should have a standard output in your console that contains the simulated intensities. If you have an error message, check the code and correct it.

Include the model in multiNMRFit

To test the integration of the model into the GUI, copy the `.py` file in the folder `models` of multiNMRFit directory.

To find the path to the multinmrfit package, you can use the following command in a Python console:

```
import multinmrfit
print(multinmrfit.__path__)
```

Once you have included your model, you can start multiNMRFit's GUI and use your model to fit a spectra. In case of errors, have a look to the error message and correct the code.

> **Note**
>
> We would be happy to broaden the types of models shipped with multiNMRFit. If you have developed a new model, it might be usefull and valuable to the NMR community! Please, keep in touch with us to discuss on the model and see if we can include your model in the built-in models shipped with multiNMRFit! :)

1.4 How to cite

Thank you for using MultiNMRFit and citing us in your work! It means a lot to us and encourage us to continue its development.

> MultiNMRFit: A software to fit 1D and pseudo-2D NMR spectra.
>
> Pierre Millard, Loïc Le Grégam, Svetlana Dubiley, Thomas Gosselin-Monplaisir, Guy Lippens, Cyril Charlier.

BioRxiv preprint, 2024, doi: 10.1101/2024.12.19.629408.

1.5 Frequently asked questions (FAQ)

1.5.1 I cannot start MultiNMRFit graphical user interface, can you help me?

If you installed MultiNMRFit following our standard procedure and that you are unable to start MultiNMRFit by opening a terminal and typing `nmrfit`, then there is indeed something wrong. Do not panic, we are here to help! Please follow this simple procedure:

1. The first step of the debugging process will be to get a *traceback*, i.e. a message telling us what is actually going wrong. You should see this message in the terminal you opened.
2. Read the traceback and try to understand what is going wrong:
   - If it is related to your system or your Python installation, you will need to ask some help from your local system administrator or your IT department so they could guide you toward a clean installation. Tell them that you wanted "to use the graphical user interface of MultiNMRFit, a Python 3.8+ software" and what you did so far (installation), give them the traceback and a link toward the documentation. They should know what to do.
   - If you believe the problem is in MultiNMRFit or that your local system administrator told you so, then you probably have found a bug! We would greatly appreciate if you could open a new issue on our issue tracker.

1.5.2 An error has been raised. What should I do?

The first thing to do is to read the error message which might contain information on how to resolve it. If not, check the FAQ section (yes, this one) to see if the error has been explained in more depth. If the error persists or if you do not understand the error, please post it in the "issues" section on GitHub. We will try to respond as quickly as possible to solve your problem.

1.5.3 Which model should I use?

The choice of the model depends on the signal you want to process. We provide a set of models in MultiNMRFit. If you have a specific signal that is not included in the models, you can create your own model. We would be happy to include it in the distribution of MultiNMRFit. Please post it in the "issues" section on GitHub.

1.5.4 How can I check if my data has been fitted correctly?

The quality of the fit can be evaluated based on the plots of experimental vs simulated data for the best fit, which should be as close as possible.

1.5.5 My spectrum hasn't been correctly fitted. Why?

A possible reason to explain a bad fit is that you did not select the right model(s). For instance, if you use a quartet model (i.e. with intensities 1:3:3:1) to fit a doublet of doublet (i.e. with intensities 1:1:1:1), the spectrum will not be fitted correctly.

In some situations, it may also be because some parameters have to be tweaked to help MultiNMRFit fit the spectrum, which results in obviously aberrant fits (e.g. with flat spectrum). If this situation happens, we suggest modifying the initial values of some parameters to obtain a simulated spectrum as close as possible to the experimental one, and re-run the fitting. For more info on the parameters and how they may affect the fitting process, please refer to section parameters.

If you think the problem is in MultiNMRFit, we would greatly appreciate if you could open a new issue on our issue tracker.

1.5.6 I have develop a new signal model, can you include it in MultiNMRFit distribution?

If you have developed a new signal model, we would be happy to include it in MultiNMRFit! Open a new issue on our issue tracker, and let's discuss about your model and how we could include it! :)

Examples of how to use MultiNMRFit programmatically can be found in the section testing_the_model, which offers demonstrations on running simulations and flux calculations.

1.5.7 I would like a new feature.

We would be glad to improve MultiNMRFit. Please get in touch with us here so we could discuss your problem.

1.6 Library documentation

1.6.1 Usage of multinmrfit package

1. General information

This notebook showcases usage of the Spectrum object of multinmrfit.

Content

- *Prepare environment*
- *Load NMR data*
- *Peak picking*
- *Spectrum simulation and fitting*
  - *Load models of NMR signals*
  - *Model construction*
  - *Simulation*
  - *Fitting*

2. Prepare environment

- Download and install Anaconda (>= 3.7) on your computer: https://www.anaconda.com/distribution/
- Install `multinmrfit`:
  - run `Anaconda prompt` from the start menu
  - install multinmrfit with: `python -m pip install git+https://github.com/NMRTeamTBI/MultiNMRFit.git@branch_name` (where `branch_name` is the git branch you want to install)
- Start Jupyter from the start menu
- Open the jupyter notbook

Import required packages.

[1]:
```
import logging
import sys
import pandas as pd
```

Initialize logger.

```
[2]: logging.basicConfig(format='%(asctime)s | %(levelname)s : %(message)s', level=logging.
     ↪DEBUG, stream=sys.stdout)
```

Load multinmrfit.

```
[3]: import multinmrfit.base.spectrum as spectrum
     import multinmrfit.base.io as io
```

3. Load NMR data

Data can be loaded from a TSV file containing columns 'ppm' and 'intensity'.

```
[4]: test_synthetic_dataset = pd.read_table("./data/data_sim_nmrfit.csv", sep="\t")
```

```
[5]: test_synthetic_dataset.columns
```

```
[5]: Index(['ppm', 'intensity'], dtype='object')
```

Data can also be loaded from TopSpin files by providing all required information in a dictionnary, assuming one-dimensional spectrum if rowno is None or two-dimensional spectrum if rowno is provided.

```
[6]: test_topspin_dataset = {"data_path":"C:/Bruker/TopSpin4.0.7/data",
                             "dataset":"CFE_test",
                             "expno":"991",
                             "procno":"1",
                             "rowno":"3"}
```

The window of interest can be provided as a tuple containing lower and upper boundaries.

```
[7]: window = (-0.2, 0.2)
```

We can then load the data in a `Spectrum` object.

```
[8]: sp = spectrum.Spectrum(data=test_synthetic_dataset, window=window)
```

```
2024-12-20 09:44:20,814 | DEBUG : create Spectrum object
```

We can then work directly with this object. For instance, to view the experimental spectrum, use the `plot()` method with `exp=True`.

```
[9]: fig = sp.plot(exp=True)
```

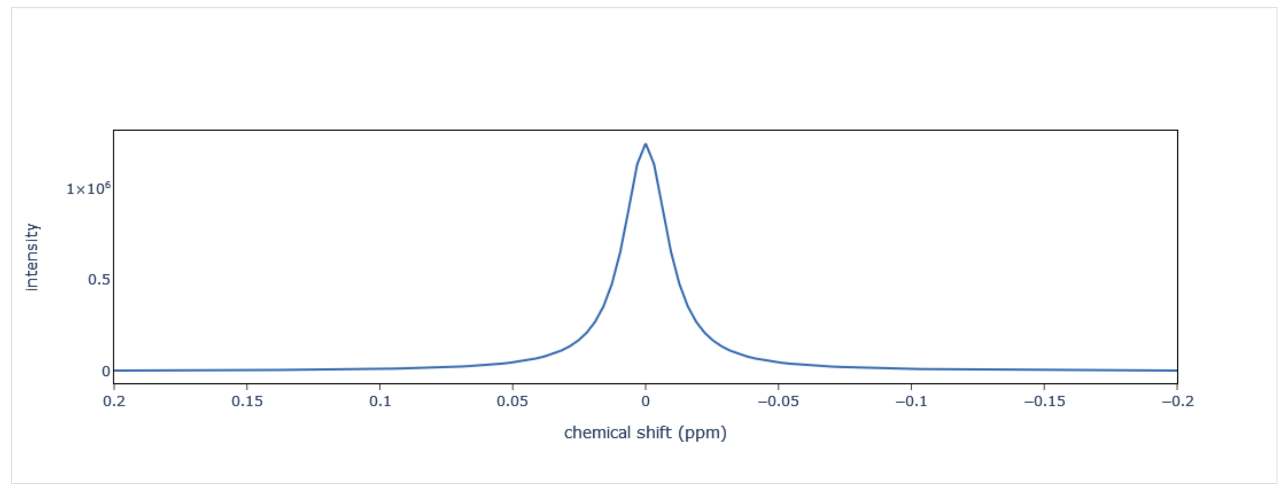
```
2024-12-20 09:44:25,632 | DEBUG : create plot
```

This method returns a `plotly.graph_objects.Figure` object that can be updated (e.g. to change the layout as shown below fot the size) and plotted.

```
[10]: type(fig)
```

```
[10]: plotly.graph_objs._figure.Figure
```

```
[11]: fig.update_layout(autosize=False, width=900, height=400)
      fig.show()
```



4. Peak picking

Use the `peak_picking()` method with argument `threshold`.

[12]:
```
peak_table = sp.peak_picking(1e6)
```

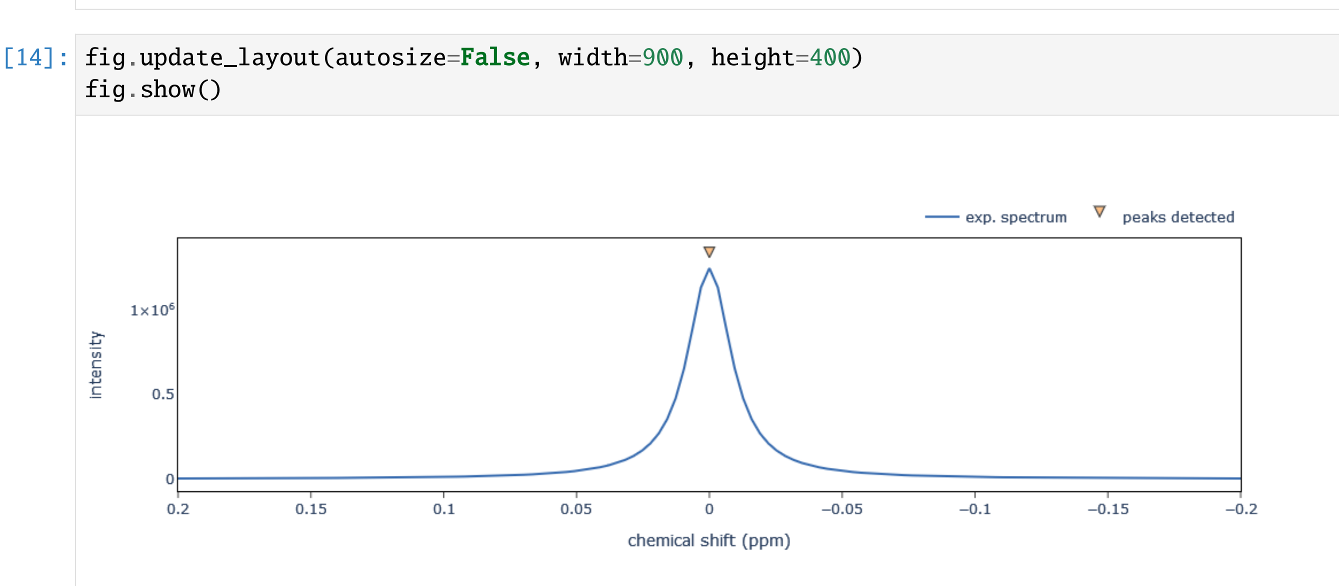
```
2024-12-20 09:44:33,517 | DEBUG : peak peaking
2024-12-20 09:44:33,543 | DEBUG : peak table
   ppm     intensity cID
0  0.0  1.250091e+06
```

To visualize the spectrum and the identified peaks, use the `plot()` method with the `peak_table` provided as argument `pp`.

[13]:
```
fig = sp.plot(pp=peak_table)
```

```
2024-12-20 09:44:36,253 | DEBUG : create plot
```

[14]:
```
fig.update_layout(autosize=False, width=900, height=400)
fig.show()
```



5. Spectrum simulation and fitting

5.1. Load models of NMR signals

Load models of signals implemented in multinmrfit.

[15]:
```
available_models = io.IoHandler.get_models()
```

```
2024-12-20 09:44:43,764 | DEBUG : add model from file 'model_acetate.py'
2024-12-20 09:44:43,785 | DEBUG : model name: 13C-acetate (CH3)
2024-12-20 09:44:43,787 | DEBUG : add model from file 'model_doublet.py'
2024-12-20 09:44:43,791 | DEBUG : model name: doublet
2024-12-20 09:44:43,792 | DEBUG : add model from file 'model_doublet_of_doublet.py'
2024-12-20 09:44:43,797 | DEBUG : model name: doublet of doublet
2024-12-20 09:44:43,798 | DEBUG : add model from file 'model_quartet.py'
2024-12-20 09:44:43,802 | DEBUG : model name: quartet
2024-12-20 09:44:43,803 | DEBUG : add model from file 'model_singlet.py'
2024-12-20 09:44:43,807 | DEBUG : model name: singlet
2024-12-20 09:44:43,808 | DEBUG : add model from file 'model_triplet.py'
2024-12-20 09:44:43,813 | DEBUG : model name: triplet
```

`io.IoHandler.get_models()` returns a dict with model_name-model_object as key-value pairs.

[16]:
```
available_models.keys()
```

[16]:
```
dict_keys(['13C-acetate (CH3)', 'doublet', 'doublet of doublet', 'quartet', 'singlet',
↪'triplet'])
```

[17]:
```
available_models["doublet"]
```

[17]:
```
multinmrfit.models.model_doublet.SignalModel
```

5.2. Model construction

To simulate or fit a spectrum, we need to provide a list of signals containing the type of signal (e.g. singlet or doublet) and the corresponding parameters (chemical shift, coupling constant, linewidth, intensity, etc). Signals must be provided as a dictionary.

[18]:
```
signals = {"singlet_TSP": {"model":"singlet", "par": {"x0": {"ini":0.0, "lb":-0.05, "ub":
↪0.05}}}}

#signals = {"singlet_TSP": {"model":"singlet", "par": {"x0": {"ini":0.0, "lb":-0.05, "ub
↪":0.05}}},
#           "doublet_TSP": {"model":"doublet", "par": {"x0": {"ini":-0.01, "lb":-0.01,
↪"ub":0.01}, "J": {"ini":0.147, "lb":0.14, "ub":0.15}, "lw": {"ini":0.001}}}}
```

Then we can build a model of the spectrum with the `build_model()` method.

[19]:
```
sp.build_model(signals=signals, available_models=available_models)
```

```
2024-12-20 09:45:01,755 | DEBUG : build Model for signal 'singlet_TSP'
2024-12-20 09:45:01,762 | DEBUG : parameters
     signal_id    model        par          ini       lb             ub
```

(continues on next page)

(continued from previous page)

```
0  singlet_TSP  singlet         x0         1.000  0.0000  1.000000e+01
1  singlet_TSP  singlet  intensity  1000000.000  1.0000  1.000000e+15
2  singlet_TSP  singlet         lw         0.001  0.0001  3.000000e-02
3  singlet_TSP  singlet         gl         0.500  0.0000  1.000000e+00
```

Parameters can be accessed via the `params` attibute.

[20]:
```
sp.params
```

[20]:
```
     signal_id    model        par          ini      lb            ub
0  singlet_TSP  singlet         x0        0.000 -0.0500  5.000000e-02
1  singlet_TSP  singlet  intensity  1000000.000  1.0000  1.000000e+15
2  singlet_TSP  singlet         lw        0.001  0.0001  3.000000e-02
3  singlet_TSP  singlet         gl        0.500  0.0000  1.000000e+00
```

We can update parameters and offser using the `update_params()` method.

[21]:
```
sp.update_params({"singlet_TSP": {"par": {"intensity": {"ini":1e6, "ub":1e12}}}})
```

Similarly, we can update the offset with the `update_offset()` method. If `offset=None`, the offset is removed. To set an offset, provide a dictionary (if empty, offset is initialized to default values).

[22]:
```
sp.update_offset(offset={})
print(sp.params)
```

```
       signal_id    model        par          ini             lb            ub
0    singlet_TSP  singlet         x0        0.000      -0.050000  5.000000e-02
1    singlet_TSP  singlet  intensity  1000000.000       1.000000  1.000000e+12
2    singlet_TSP  singlet         lw        0.001       0.000100  3.000000e-02
3    singlet_TSP  singlet         gl        0.500       0.000000  1.000000e+00
4  full_spectrum     None     offset        0.000 -250018.225728  2.500182e+05
```

[23]:
```
sp.update_offset(offset=None)
print(sp.params)
```

```
     signal_id    model        par          ini      lb            ub
0  singlet_TSP  singlet         x0        0.000 -0.0500  5.000000e-02
1  singlet_TSP  singlet  intensity  1000000.000  1.0000  1.000000e+12
2  singlet_TSP  singlet         lw        0.001  0.0001  3.000000e-02
3  singlet_TSP  singlet         gl        0.500  0.0000  1.000000e+00
```

5.3. Simulation

Simulate spectrum using the `simulate()` method which returns a list of simulated intensities. If `params` is not given as argument, initial parameters values are used.

[24]:
```
sim_spectrum = sp.simulate()
```

[25]:
```
display(sim_spectrum)
```

```
0       12.499688
1       12.906144
2       13.332754
3       13.780870
```

(continues on next page)

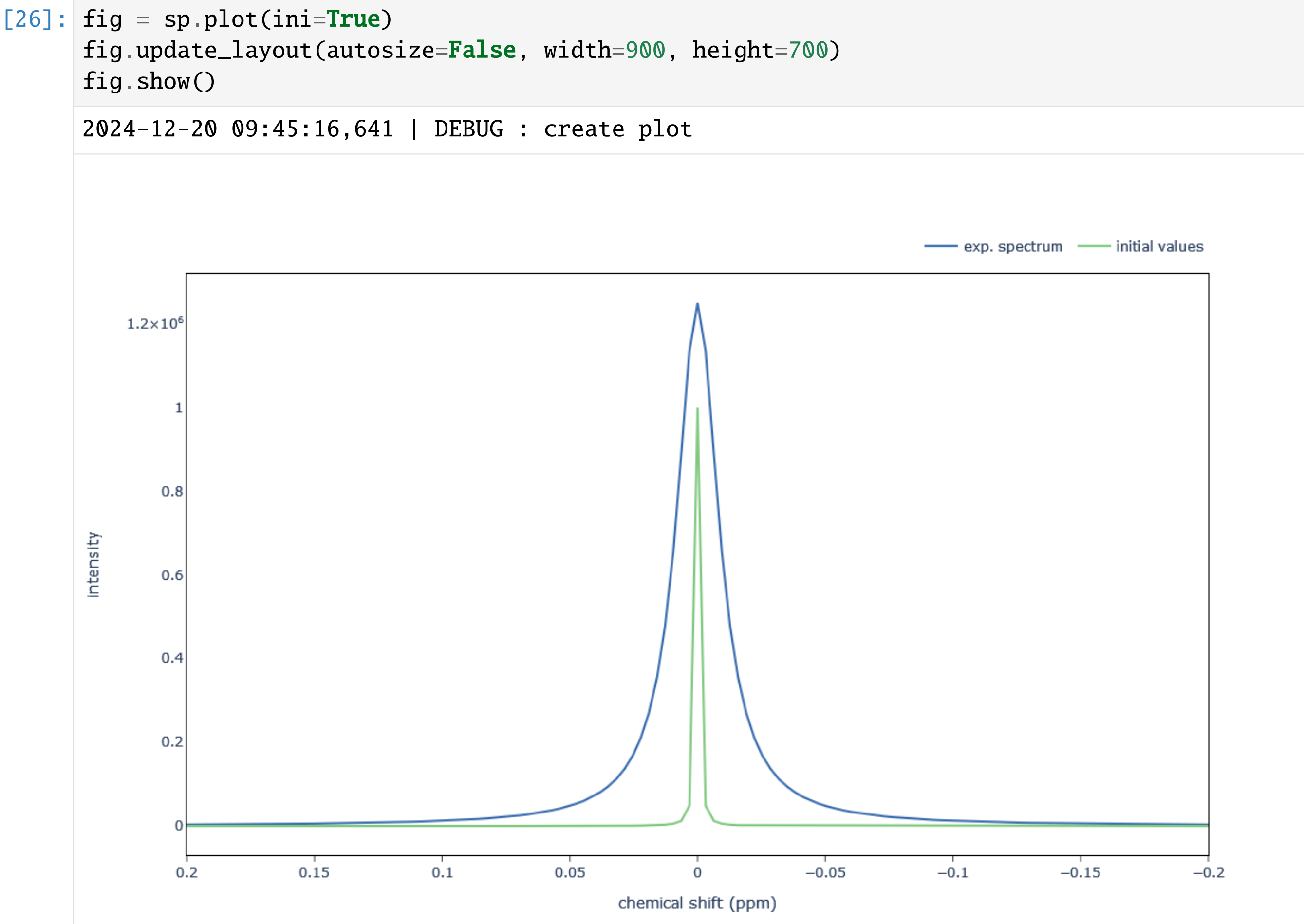
(continued from previous page)

```
4       14.251964
          ...
122     14.251964
123     13.780870
124     13.332754
125     12.906144
126     12.499688
Name: ppm, Length: 127, dtype: float64
```

To view the spectrum simulated from initial values, just use the `plot()` method.

[26]:
```
fig = sp.plot(ini=True)
fig.update_layout(autosize=False, width=900, height=700)
fig.show()
```

```
2024-12-20 09:45:16,641 | DEBUG : create plot
```



5.4. Fitting

To fit experimental spectrum, use the `fit()` method.

[27]:
```
sp.fit()
```

```
2024-12-20 09:45:20,402 | DEBUG : fit spectrum
2024-12-20 09:45:21,197 | DEBUG : parameters
     signal_id     model         par          ini       lb             ub  \
0  singlet_TSP  singlet          x0        0.000 -0.0500  5.000000e-02
```

(continues on next page)

(continued from previous page)

```
1  singlet_TSP  singlet  intensity  1000000.000  1.0000  1.000000e+12
2  singlet_TSP  singlet         lw        0.001  0.0001  3.000000e-02
3  singlet_TSP  singlet         gl        0.500  0.0000  1.000000e+00

            opt     opt_sd      integral
0  8.048877e-10   0.000004  38655.427821
1  1.250012e+06  62.154093  38655.427821
2  1.000263e-02   0.000002  38655.427821
3  1.000000e+00   0.000177  38655.427821
```

Estimated parameters, standard deviations and integrals are now in the params attributes (columns `opt`, `opt_sd` and `integral`, respectively).

[28]:
```
sp.params
```

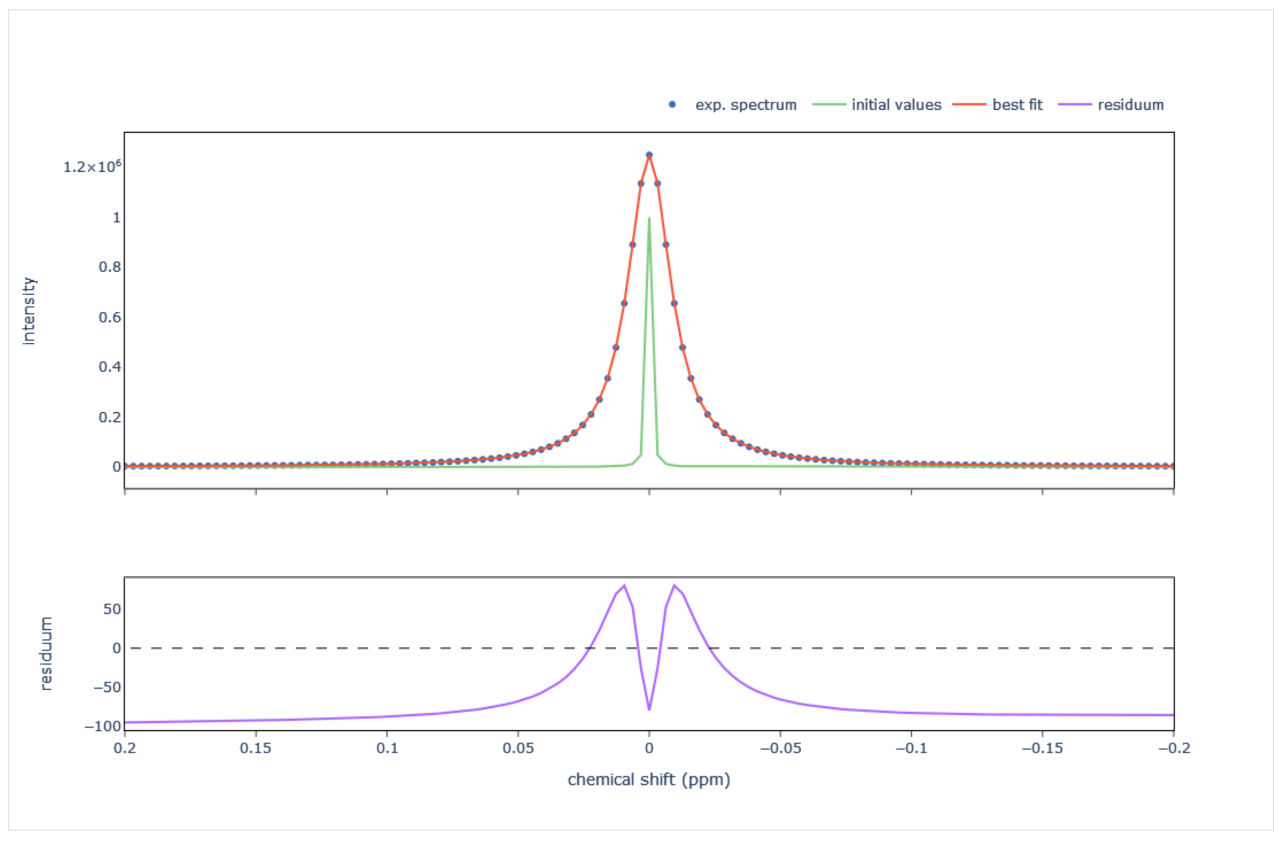
[28]:
```
     signal_id    model        par          ini      lb            ub  \
0  singlet_TSP  singlet         x0        0.000 -0.0500  5.000000e-02
1  singlet_TSP  singlet  intensity  1000000.000  1.0000  1.000000e+12
2  singlet_TSP  singlet         lw        0.001  0.0001  3.000000e-02
3  singlet_TSP  singlet         gl        0.500  0.0000  1.000000e+00

            opt     opt_sd      integral
0  8.048877e-10   0.000004  38655.427821
1  1.250012e+06  62.154093  38655.427821
2  1.000263e-02   0.000002  38655.427821
3  1.000000e+00   0.000177  38655.427821
```

Fitting results can be viewed using the plot method with fit=True.

[29]:
```
fig = sp.plot(ini=True, fit=True)
fig.update_layout(autosize=False, width=900, height=700)
fig.show()
```

```
2024-12-20 09:45:24,827 | DEBUG : create plot
```



The fit will be better with an offset, we thus add it and fit again the data.

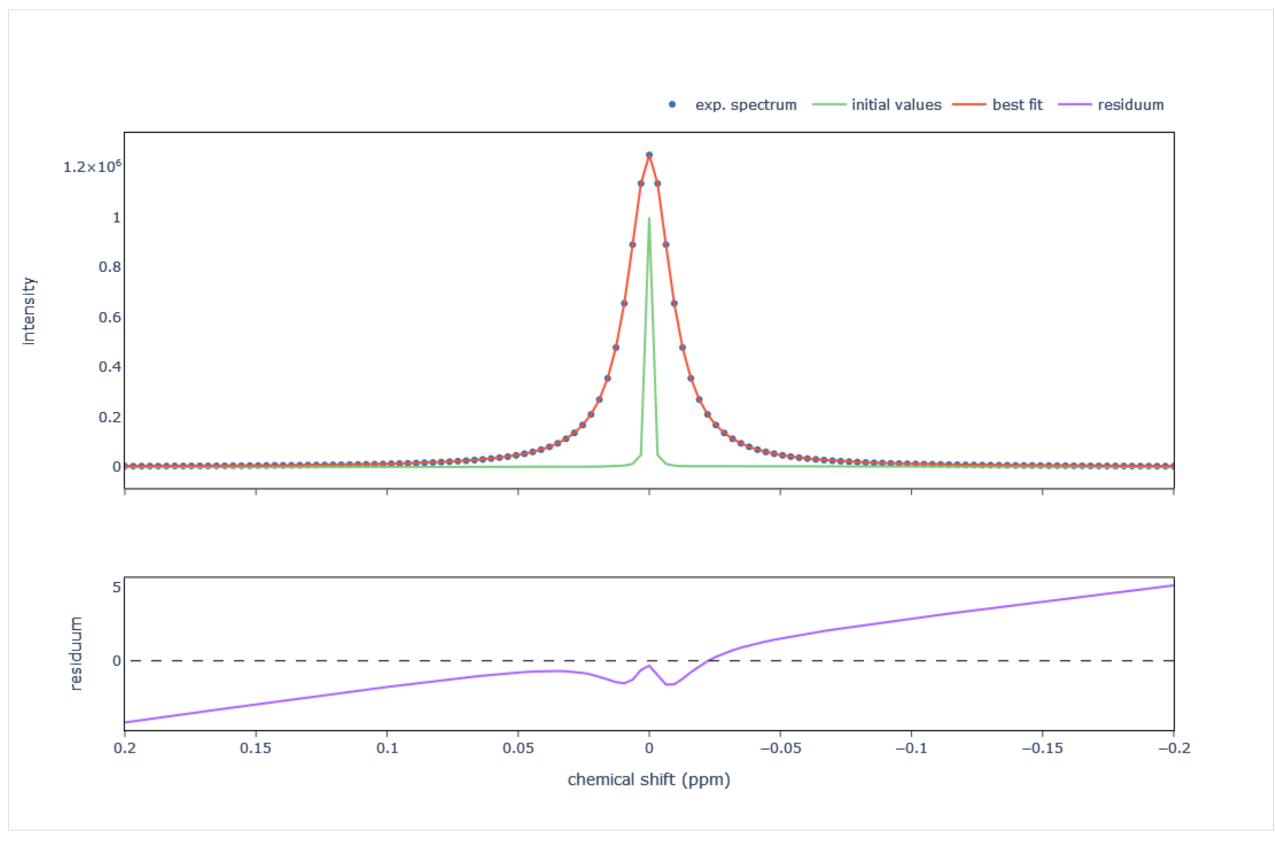
[30]:
```
sp.update_offset(offset={})
sp.fit()
```

```
2024-12-20 09:45:29,255 | DEBUG : fit spectrum
2024-12-20 09:45:30,136 | DEBUG : parameters
       signal_id    model         par          ini            lb  \
0    singlet_TSP  singlet          x0        0.000     -0.050000
1    singlet_TSP  singlet   intensity  1000000.000      1.000000
2    singlet_TSP  singlet          lw        0.001      0.000100
3    singlet_TSP  singlet          gl        0.500      0.000000
4   full_spectrum     None      offset        0.000 -250018.225728

             ub           opt        opt_sd      integral
0  5.000000e-02  3.940879e-09  7.231758e-07  38644.900682
1  1.000000e+12  1.249999e+06  3.013674e+02  38644.900682
2  3.000000e-02  9.999972e-03  1.117001e-06  38644.900682
3  1.000000e+00  9.999999e-01  9.555780e-05  38644.900682
4  2.500182e+05  9.180373e+01  8.011525e+01           NaN
```

[31]:
```
fig = sp.plot(ini=True, fit=True)
fig.update_layout(autosize=False, width=900, height=700)
fig.show()
```

```
2024-12-20 09:45:33,141 | DEBUG : create plot
```



1.6.2 API reference

This module serves as reference for the different classes and associated methods of the MultiNMRFit package.

`spectrum.py`

class `multinmrfit.base.io.`**`IoHandler`**

Bases: `object`

`check_dataset`()

static **`filter_window`**(*ppm: list*, *intensity: list*, *window: tuple = None*)

static **`get_models`**()

Load signal models. :return: dict containing the different model objects, with model.name as keys

static **`load_1D_spectrum`**(*data_path*, *dataset*, *procno*, *expno_list*)

static **`load_2D_spectrum`**(*data_path*, *dataset*, *expno*, *procno*)

Load 2D NMR spectra.

Returns:

list: chemical shift list: intensity

`load_data`(*data*, *window: tuple = None*, *rowno: int = None*)

static **`load_txt_spectrum`**(*txt_data*)

static **`read_topspin_data`**(*data_path*, *dataset*, *expno*, *procno*, *rowno=None*, *window=None*)

1.7 License

MultiNMRFit is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

MultiNMRFit is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with MultiNMRFit. If not, see https://www.gnu.org/licenses/.

PYTHON MODULE INDEX

m

multinmrfit.base.io, 26

INDEX

C

`check_dataset()` (*multinmrfit.base.io.IoHandler method*), 26

F

`filter_window()` (*multinmrfit.base.io.IoHandler static method*), 26

G

`get_models()` (*multinmrfit.base.io.IoHandler static method*), 26

I

`IoHandler` (*class in multinmrfit.base.io*), 26

L

`load_1D_spectrum()` (*multinmrfit.base.io.IoHandler static method*), 26

`load_2D_spectrum()` (*multinmrfit.base.io.IoHandler static method*), 26

`load_data()` (*multinmrfit.base.io.IoHandler method*), 26

`load_txt_spectrum()` (*multinmrfit.base.io.IoHandler static method*), 26

M

`module`
- `multinmrfit.base.io`, 26

`multinmrfit.base.io`
- `module`, 26

R

`read_topspin_data()` (*multinmrfit.base.io.IoHandler static method*), 26