PyMoDAQ-Femto Documentation

Release 0.2.2

Weber Sebastien

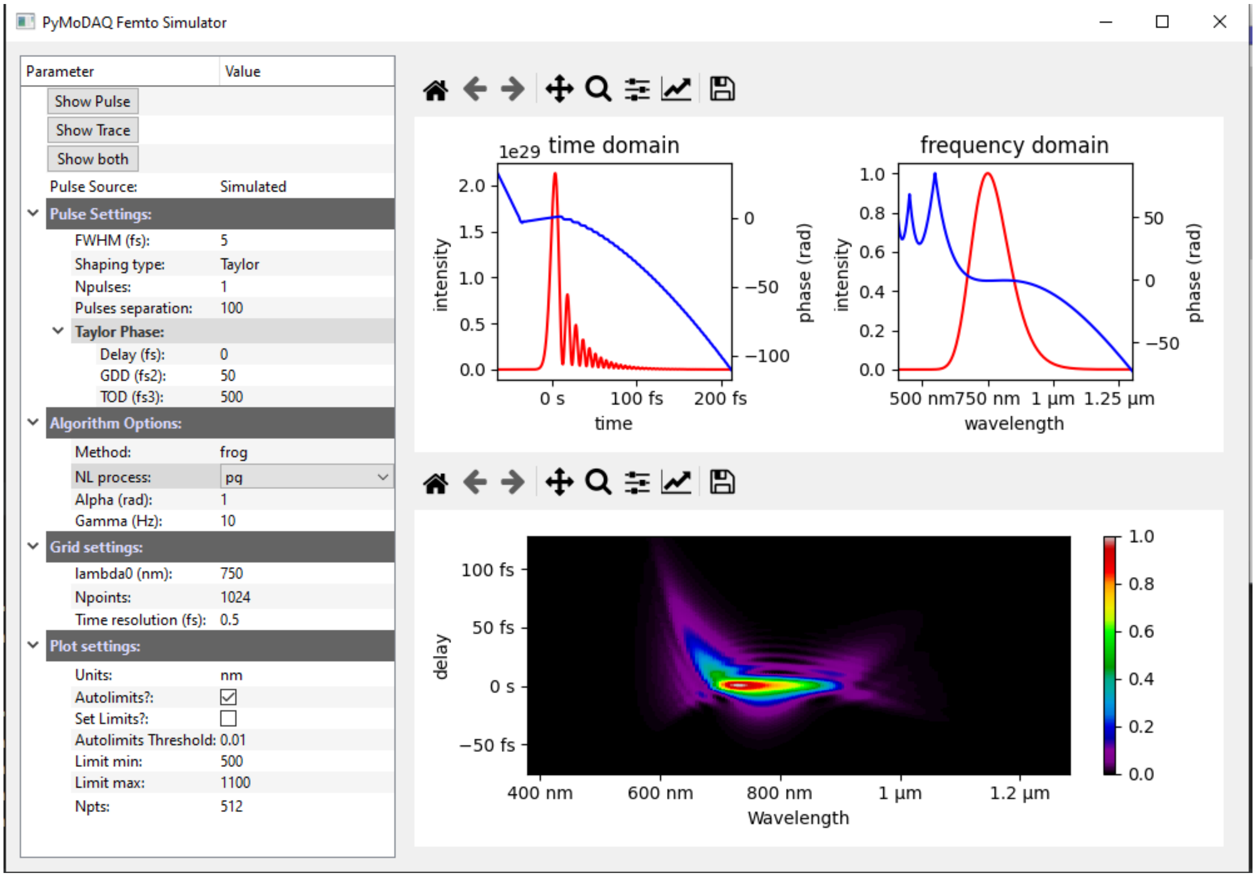
Sep 11, 2023

CONTENTS:

PyMoDAQ-Femto is an application that allows the temporal characterization of laser pulses (typically of femtosecond or picosecond duration). The module was initially developed for educational purposes in the framework of the international Femto-UP 2020 School. It has now turned into a tool used in several research laboratories.

PyMoDAQ-Femto is a 2-in-1 python application dealing with femtosecond laser pulse characterization. It features:

- A user interface called **Simulator** to define short pulses and simulate the non-linear traces they yield. Most known characterization techniques are available (FROG, D-Scan, . . . see *Available methods*), and several non-linear processes are implemented for each of them.
- A user interface called **Retriever** to run various retrieval algorithms on simulated or experimental traces (acquired using PyMoDAQ or other means)



PyMoDAQ Femto Simulator

Parameter	Value
Show Pulse	
Show Trace	
Show both	
Pulse Source:	Simulated
Pulse Settings:	
FWHM (fs):	5
Shaping type:	Taylor
Npulses:	1
Pulses separation:	100
Taylor Phase:	
Delay (fs):	0
GDD (fs2):	50
TOD (fs3):	500
Algorithm Options:	
Method:	frog
NL process:	pg
Alpha (rad):	1
Gamma (Hz):	10
Grid settings:	
lambda0 (nm):	750
Npoints:	1024
Time resolution (fs):	0.5
Plot settings:	
Units:	nm
Autolimits?:	☑
Set Limits?:	☐
Autolimits Threshold:	0.01
Limit min:	500
Limit max:	1100
Npts:	512

Fig. 1: PyMoDAQ-Femto's Simulator.

Both modules can be ran as stand-alone applications or plugged as an extension to PyMoDAQ. All together it produces a framework for complete temporal characterization of shaped ultrashort femtosecond pulses.

CHAPTER ONE

INFORMATION

GitHub repo: https://github.com/PyMoDAQ/pymodaq_femto

Documentation: http://pymodaq_femto.cnrs.fr/

Based on PyMoDAQ, the pypret library and the `pyqtgraph` library.

PyMoDAQ-Femto is written by Sébastien Weber: sebastien.weber@cemes.fr and Romain Géneaux: romain.geneaux@cea.fr under a MIT license.

CHAPTER

TWO

CONTRIBUTION

If you want to contribute see this page: *Contributors*

CHAPTER THREE

CITATION

The module is described at length in the following open-access paper:

> Romain Géneaux and Sébastien Weber, "Femtosecond Pulse Shaping and Characterization: From Simulation to Experimental Pulse Retrieval Using a Python-Based User Friendly Interface". In J. Léonard and C. Hirlimann (Eds.), *Ultrafast Laser Technologies and Applications* (pp. 111-128). EDP Sciences. https://doi.org/10.1051/978-2-7598-2719-0

If you publish results obtained with the help of the PyMoDAQ-Femto interface, we would appreciate your using this reference. In that way, you're also helping in its promotion and amelioration.

CHAPTER

FOUR

INDEX

4.1 Features

4.1.1 Overview of PyMoDAQ-Femto

The package implements several retrieval algorithms for ultrashort laser pulse measurement methods, such as as frequency-resolved optical gating (FROG), dispersion scan (d-scan), and more. The application can simulate measurement traces from various pulse shapes, and apply retrieval algorithms to them. It also works on real experimental measured traces.

PyMoDAQ-Femto is written in Python and uses Python 3.7+. It is an extension of the PyMoDAQ package, which is a Modular Data Acquisition module that can interface any kind of experiment. As such, PyMoDAQ-Femto can natively work on data measured using PyMoDAQ, although it can also work on any data provided that it is converted to the proper format (see *Converting raw data to be used in the retriever* for conversion guidelines).

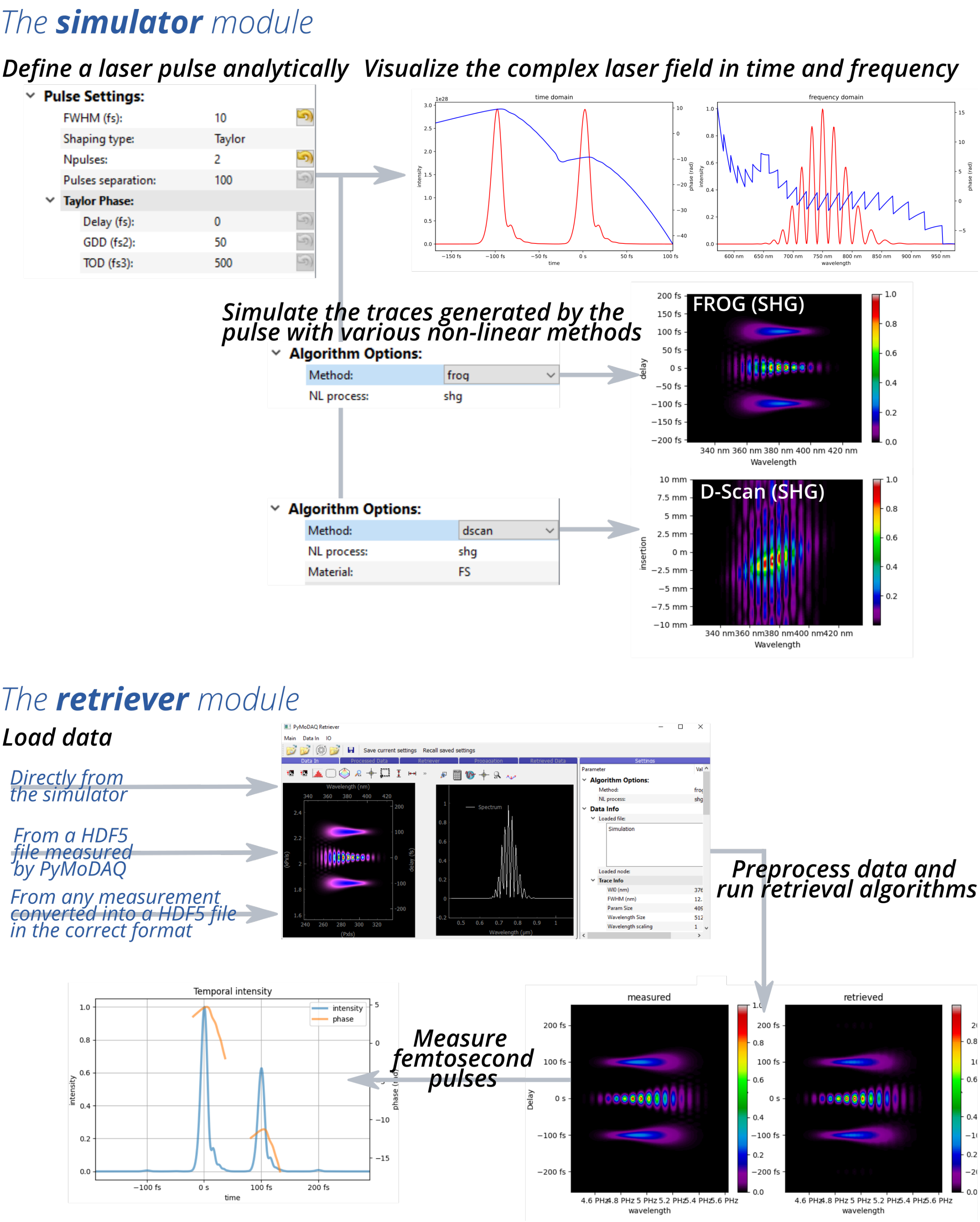
The algorithms implemented in PyMoDAQ-Femto are based on the excellent pypret package, which provides a common pulse retrieval algorithm to several pulse measurement methods (see [Geib2019] for a full description).

4.1.2 Available methods

Table 4.1: Available measurement types that can be simulated or retrieved, with their supported non-linear processes.

Method	Full Name	Supported non-linear processes[0]
*frog	Frequency-resolved optical gating	shg, pg, tg
dscan	Dispersion scan	shg, thg, sd
ifrog	Interferometric frequency-resolved optical gating	shg, thg, sd
miips	Multiphoton intrapulse interference phase scan	shg, thg, sd
tdp	Time-domain ptychography	shg, thg, sd

[0] shg: Second Harmonic Generation, thg: Third Harmonic Generation, sd: Self Diffraction, pg: Polarization Gating, tg: Transient Grating



The ***simulator*** *module*

Define a laser pulse analytically

Pulse Settings:
- FWHM (fs): 10
- Shaping type: Taylor
- Npulses: 2
- Pulses separation: 100

Taylor Phase:
- Delay (fs): 0
- GDD (fs2): 50
- TOD (fs3): 500

Visualize the complex laser field in time and frequency

Simulate the traces generated by the pulse with various non-linear methods

Algorithm Options:
- Method: frog
- NL process: shg

Algorithm Options:
- Method: dscan
- NL process: shg
- Material: FS

The ***retriever*** *module*

Load data

Directly from the simulator

From a HDF5 file measured by PyMoDAQ

From any measurement converted into a HDF5 file in the correct format

Preprocess data and run retrieval algorithms

Measure femtosecond pulses

Fig. 4.1: Overview of the two modules of PyMoDAQ-Femto

4.2 Installation

For PyMoDAQ-Femto to run smoothly, you need a Python distribution to be installed. Here are some advices. On all platforms **Windows**, **MacOS** or **Linux**, Anaconda or Miniconda is the advised distribution/package manager. Environments can be created to deal with different version of packages and isolate the code from other programs. Anaconda comes with a full set of installed scientific python packages while *Miniconda* is a very light package manager.

4.2.1 Setting up a new environment

- Download and install Miniconda3.
- Open a console, and cd to the location of the *condabin* folder, for instance: `C:\Miniconda3\condabin`
- Create a new environment: `conda create -n my_env python=3.8`, where my_env is your new environment name. This will create the environment with python version 3.8 that is currently the recommended one.
- Activate your environment so that only packages installed within this environment will be *seen* by Python: `conda activate my_env`

4.2.2 Installing PyMoDAQ-Femto

Easiest part: in your newly created and activated environment enter: `pip install pymodaq_femto`. This will install the latest PyMoDAQ-Femto available version and all its dependencies. For a specific version enter: `pip install pymodaq_femto==x.y.z`.

4.2.3 Launching PyMoDAQ-Femto

During its installation, two scripts have been installed within you environment directory, this means you can start PyMoDAQ-Femto's two main functionalities directly writing in your console either:

- `simulator`
- `retriever`

Alternatively, you can specify the full commands (The *-m* option tells python to look within its *site-packages* folder, where you've just installed pymodaq_femto):

- `python -m pymodaq_femto.simulator`
- `python -m pymodaq_femto.retriever`

4.2.4 Creating shortcuts on Windows

Python packages can easily be started from the command line (see *Launching PyMoDAQ-Femto*). However, Windows users will probably prefer using shortcuts on the desktop. Here is how to do it (Thanks to Christophe Halgand for the procedure):

- First create a shortcut (see Fig. 4.2) on your desktop (pointing to any file or program, it doesn't matter)
- Right click on it and open its properties (see Fig. 4.3)
- On the *Start in* field ("Démarrer dans" in french and in the figure), enter the path to the condabin folder of your miniconda or anaconda distribution, for instance: `C:\Miniconda3\condabin`
- On the *Target* field, ("Cible" in french and in the figure), enter this string: `C:\Windows\System32\cmd.exe /k conda activate my_env & python -m pymodaq_femto.retriever`. This means that your shortcut will open the windows's command line, then execute your environment activation (*conda activate my_env* bit), then finally execute and start **Python**, opening the correct pymodaq_femto file (here *retriever.py*, starting the Retriever module, *python -m pymodaq_femto.retriever* bit)
- You're done!
- Do it again for each PyMoDAQ-Femto's module you want (to get the correct python file and it's path, see *Launching PyMoDAQ-Femto*).

4.3 Simulator Module

4.3.1 Launching the simulator

- Follow the steps to install PyMoDAQ Femto (see *Installation section*)
- Open a shell (for instance, Anaconda Prompt) with the correct conda environment activated
- Type `simulator`

4.3.2 Interface

The interface with default parameters is shown below. It is comprised of 3 panels:

1. The parameter tree
2. The pulse panel (with time and frequency plots)
3. The trace panel

Note that at first, panels 2 and 3 will be blank, you need to press the buttons *Show Pulse*, *Show Trace*, *Show Both* to update panel 2, 3, or both.

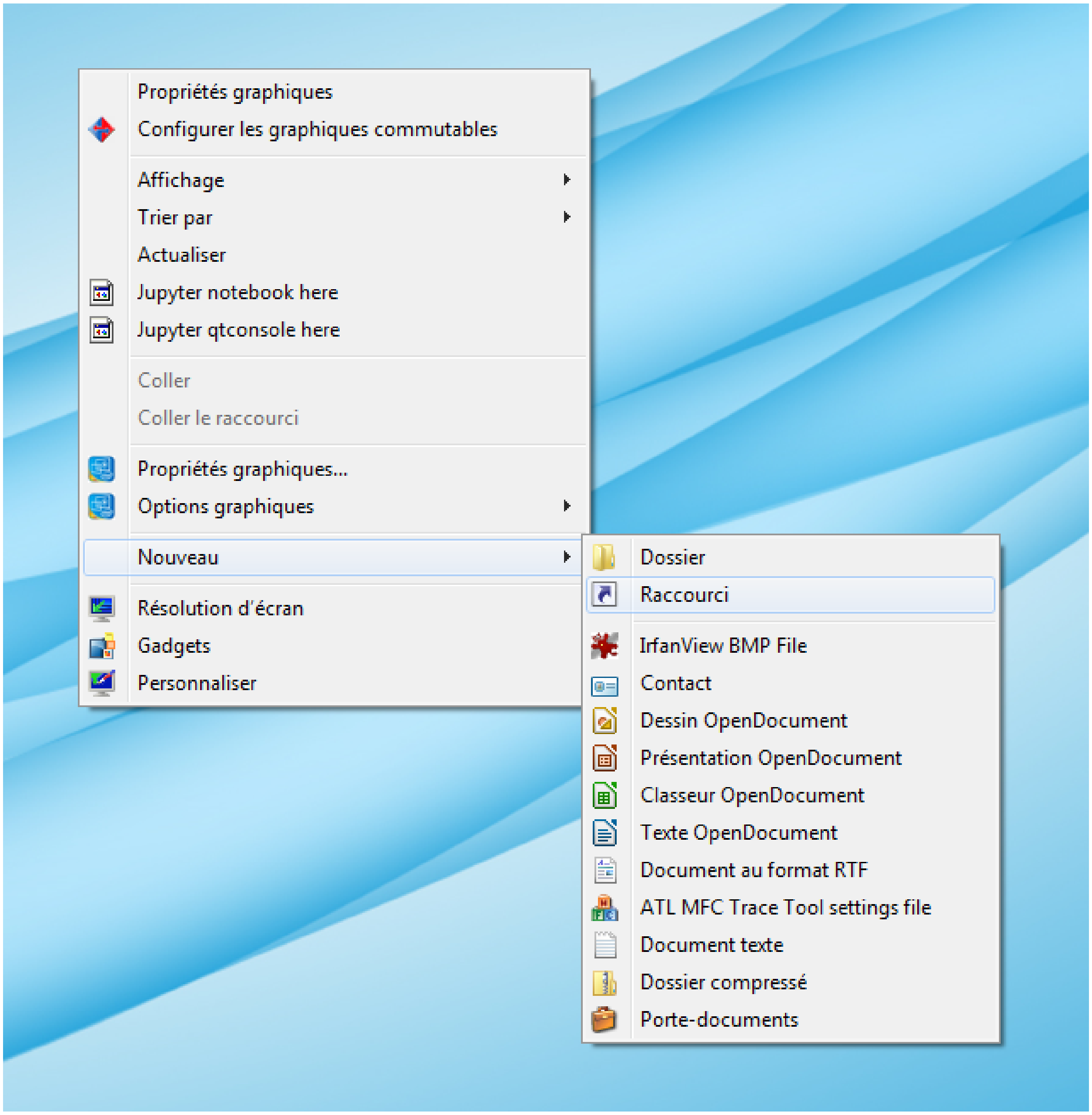
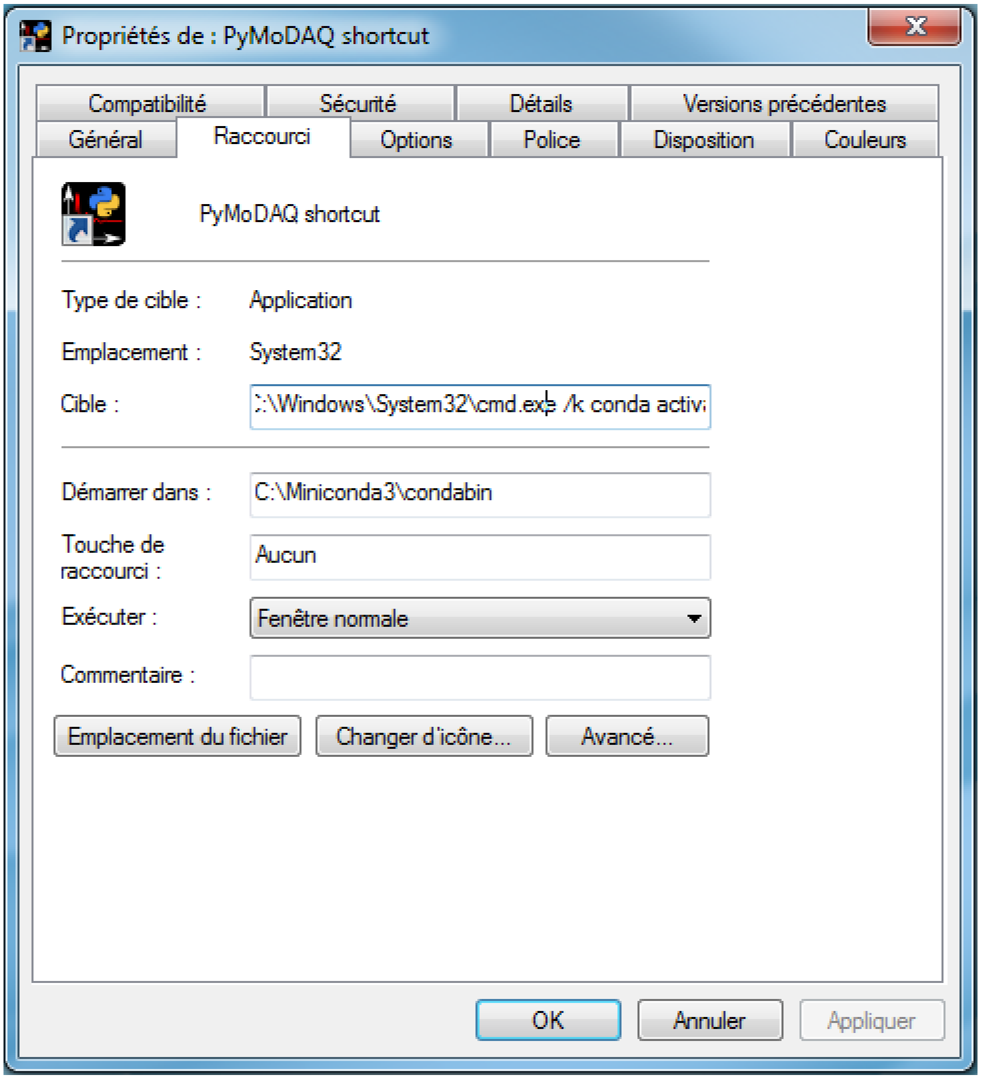


Fig. 4.2: Create a shortcut on your desktop



PyMoDAQ shortcut

Type de cible : Application

Emplacement : System32

Cible : :\Windows\System32\cmd.exe /k conda activ

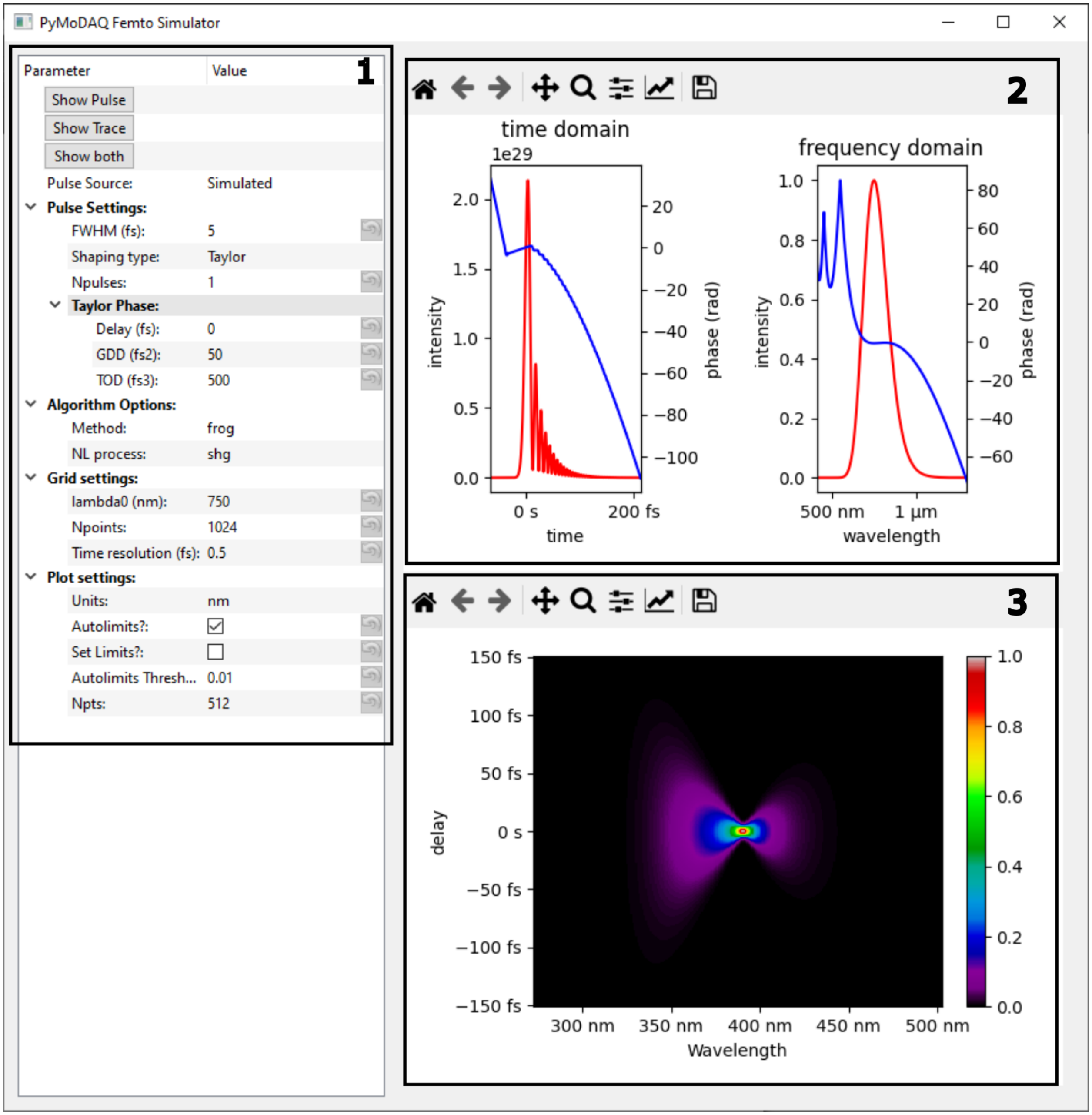
Démarrer dans : C:\Miniconda3\condabin

Touche de raccourci : Aucun

Exécuter : Fenêtre normale

Commentaire :

Fig. 4.3: Shortcut properties



PyMoDAQ Femto Simulator

Parameter	Value
Show Pulse	
Show Trace	
Show both	
Pulse Source:	Simulated
Pulse Settings:	
FWHM (fs):	5
Shaping type:	Taylor
Npulses:	1
Taylor Phase:	
Delay (fs):	0
GDD (fs2):	50
TOD (fs3):	500
Algorithm Options:	
Method:	frog
NL process:	shg
Grid settings:	
lambda0 (nm):	750
Npoints:	1024
Time resolution (fs):	0.5
Plot settings:	
Units:	nm
Autolimits?:	☑
Set Limits?:	□
Autolimits Thresh...	0.01
Npts:	512

Fig. 4.4: Interface of the simulator module of PyMoDAQ-Femto with default parameters.

4.3.3 Description of parameters

Show Pulse (type: `action`)
Updates the display of the input pulse according to the current *Pulse Settings*. The input pulse is displayed in a *matplotlib* figure, in the temporal and spectral domains. Both intensity and phase are shown for each plot.

Show Trace (type: `action`)
Updates the display of the non-linear trace according to the current *Algorithm Settings*. The trace corresponds to the spectra generated by a non-linear process as a function of a parameter, see the pypret documentation).

Show both (type: `action`)
Updates the display of both the input pulse and the non-linear trace.

Pulse Source: (type: `list`, values: `Simulated` or `From File`)
If set to `Simulated`, the input pulse is analytically defined according to the *Pulse Settings*.

If set to `From File`, an external CSV file will be used. The CSV file must contain in columns: Wavelength (nm), Normalized Spectral Intensity and Phase in radians. A default file (*spectral_data.csv*) is supplied in the plugin.

Pulse Settings:

- FWHM (fs): `(type: float)` Temporal full width at half maximum in femtoseconds.
- Shaping type: `(type: list, values: Taylor or Gaussian)` The phase is either defined as a Taylor series in the spectral domain, or as a Gaussian in the temporal domain.
- Npulses: `(type: int)` Number of pulses.
- Pulses separation: `(type: float)` Delay between the pulses, if there are more than one.
- Taylor Phase: `(type: group)` Group delay in fs, group delay dispersion in fs^2, and third order dispersion in fs^3. Higher phase orders could be easily implemented if required.
- Gaussian Phase: `(type: group)` Amplitude and full-width at half maximum of the Gaussian temporal phase.
- Data File: `(type: browsepath)` Path to the user-supplied file if Pulse Source is set to `From File`.

Algorithm Options:

- Method: `(type: list)` The type of measurement (FROG, DSCAN, etc.). See *Available methods* for a full list of available methods.
- NL process: `(type: list)` The non-linear process to use (second harmonic generation, third harmonic generation, etc.). Note that not all processes are compatible with all methods. See *Available methods* for a full list.

The next available option is the definition of the parameter that is scanned during the measurement. It depends on the chosen method.
FROG, IFROG, TDP: The parameter is the delay in fs. It is taken to be the same as the temporal axis on which the pulse is defined.

MIIPS: The parameter is the delta of the phase pattern.

- Alpha (rad): `(type: float)` Amplitude
- Gamma (Hz): `(type: float)` Frequency
- MIIPS Parameter Scan: `(type: group)` Phase minimum, maximum, and step size, in rad.

DSCAN: The parameter is the amount of material inserted in mm.

- Material: `(type: list)` Material inserted. Currently, only Fused Silica (FS) and BK7 are implemented.
- Dscan Parameter Scan: `(type: group)` Insertion minimum, maximum, and step size, in mm.

Grid settings:

- lambda0 (nm): `(type: float)` Central wavelength of the spectral axis.
- Npoints: `(type: list)` Number of points of the spectral axis.
- Time resolution (fs): `(type: float)` Time between two points in the temporal axis. Determines the temporal resolution, and also sets the span of the spectral axis.

Plot settings:

- Units: `(type: list, values: nm or Hz)` Unit for spectral axis: wavelength in nanometers, or angular frequency in Hz.
- Autolimits?: `(type: bool)` When toggled, automatically restricts the limits of the trace plot to match the data.
- Autolimits Threshold: `(type: float)` Threshold for the autolimits: 0.01 means that we show the region where the trace is above 1% of its maximum value.
- Set Limits?: `(type: bool)` When toggled, the user can choose limits in the spectral domain.
- Npts: `(type: list)` Number of points in the spectral domain for the trace plot.

4.4 Retriever Module

4.4.1 Launching the retriever

Launching the retriever as a stand-alone program:

- Follow the steps to install PyMoDAQ Femto (see *Installation section*)
- Open a shell (for instance, Anaconda Prompt) with the correct conda environment activated
- Type `retriever`

Launching the retriever from a PyMoDAQ Dashboard:

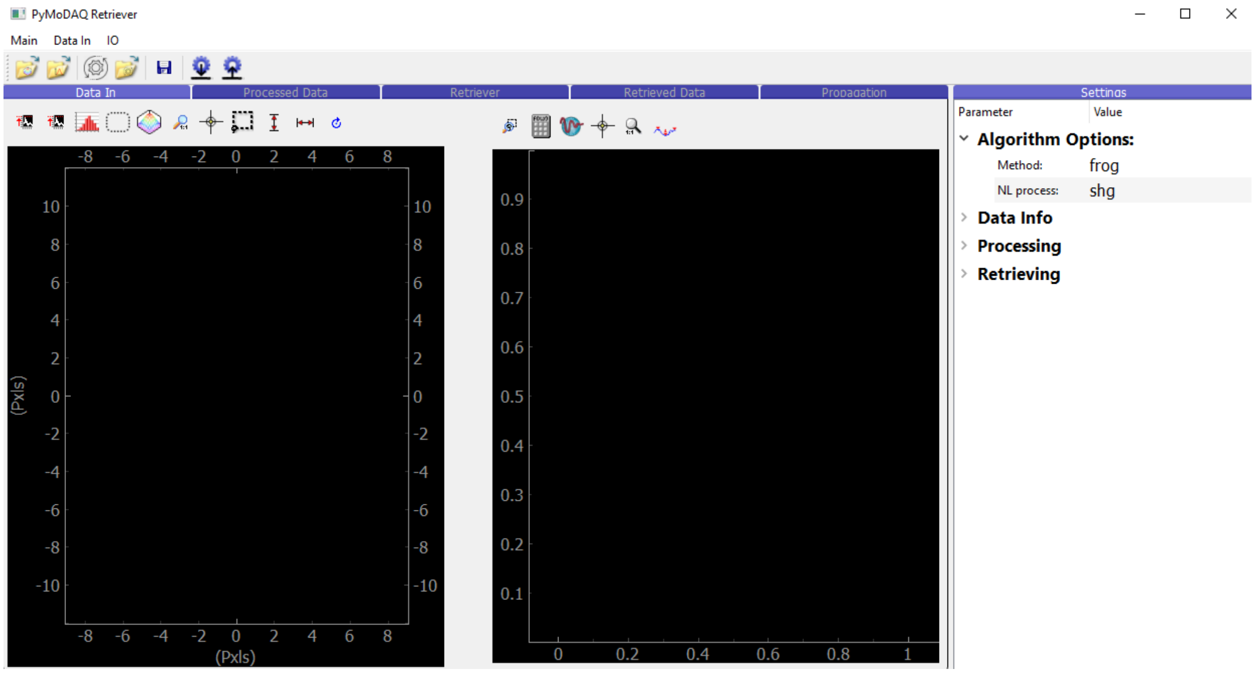
- Load your dashboard
- In the top bar menu, go to Extensions/FemtoRetriever

4.4.2 Overview of the retrieval procedure

The interface of the retriever is shown below:

The blue tabs follow the general retrieval procedure:

1. *Data In*: Data is imported, displayed, and regions of interest can be selected
2. *Processed Data*: The non-linear trace is pre-processed and interpolated on a user-defined grid
3. *Retriever*: An iterative algorithm tries to reproduce the measured trace. Each iteration is displayed during the retrieval
4. *Retrieved Data*: The best solution is displayed and compared to the measurement



Method: frog

NL process: shg

Fig. 4.5: Interface of the retriever module of PyMoDAQ-Femto (before loading any data).

5. *Propagation*: Arbitrary amounts of material (air, glass, etc.) can be added to the pulse to simulate how the pulse will look like after propagation.

4.4.3 Loading data

The retriever handles both data simulated by the *Simulator Module* and experimental data.

We **strongly recommend** that you experiment with the retriever on simulated data before attempting to retrieve experimental data.

4.4.3.1 Loading simulated data generated by the simulator module

Click the wheel ⚙ icon in the top menu to launch the simulator. If you are not familiar with its behavior, check out the documentation here: *Simulator Module*.

Set up the pulse and the trace as you like. Here we will use the default values for everything, except the method: instead of the default SHG frog, we will use PG frog. This should generate the same trace as the PyMoDAQ-Femto logo that you see at the top left of this page!

Then, go back to the retriever and click the 📂 icon. Both the trace and the fundamental spectrum should now be displayed as shown below. The data viewers use the PyMoDAQ viewers, which have many functionalities available: lineouts, regions of interest, cursors, rescaling of axes, colormap changes, etc.

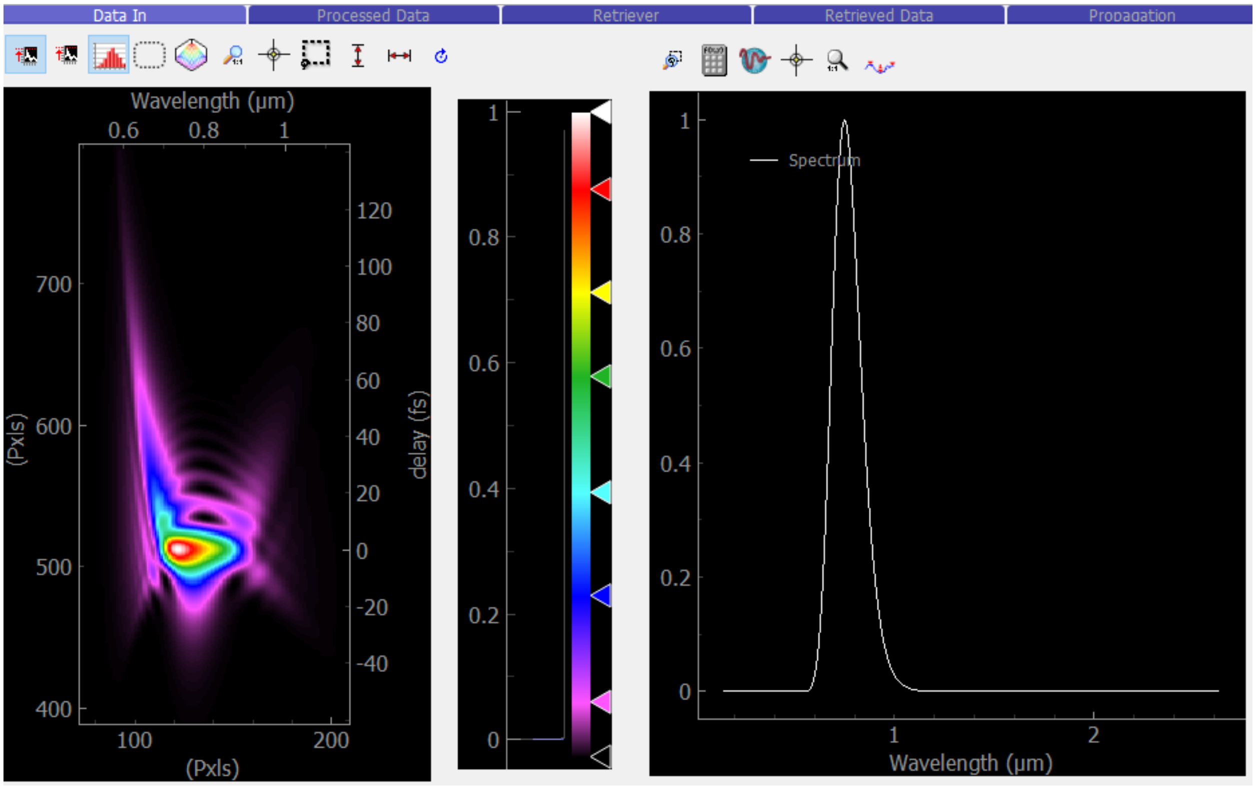


Fig. 4.6: Retriever with data loaded from simulator.

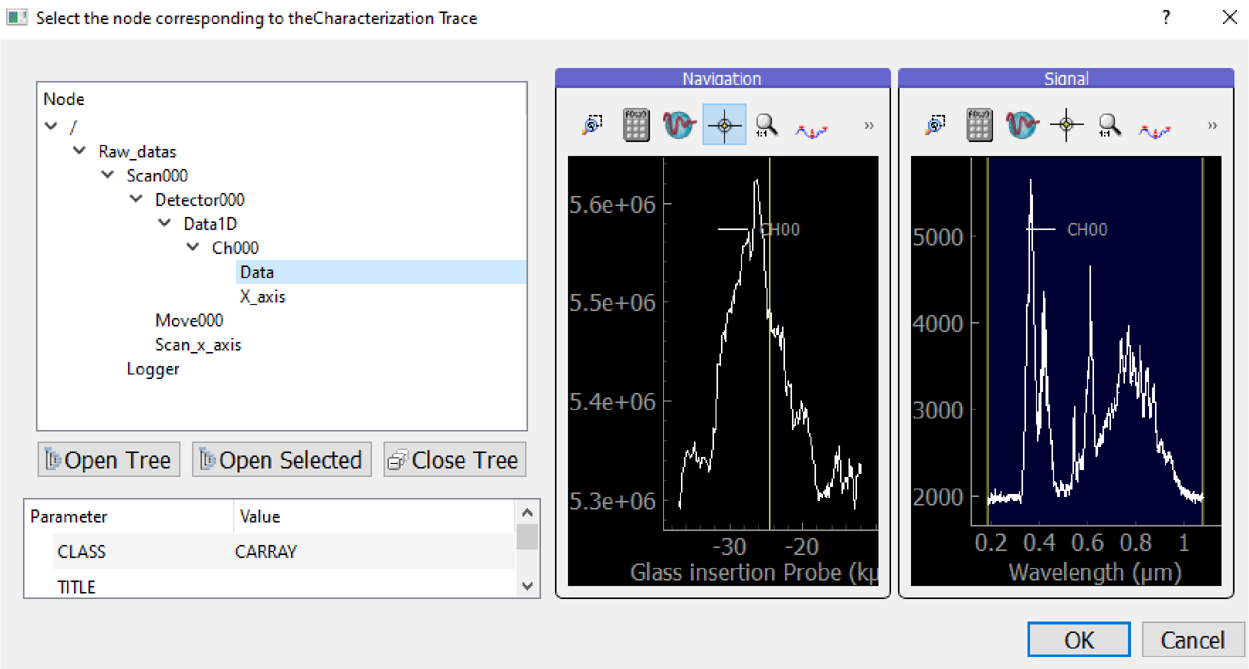
4.4.3.2 Loading experimental data

4.4.3.2.1 Data measured using PyMoDAQ

If your data was measured using PyMoDAQ Scan function, then we have good news for you: they already have the proper formatting and will seamlessly load into PyMoDAQ-Femto!

To load the trace, click the *Load Experimental Trace* icon, and navigate to your .h5 file. It will open your file in H5Browser. Select the Data node corresponding to the trace and double-click it. Lineouts of the trace will be displayed as shown below. Then, click OK to load the trace.

To load the fundamental spectrum, click the *Load Experimental Spectrum* icon and repeat the same steps to load the spectrum.



Select the node corresponding to theCharacterization Trace

Node
/
Raw_datas
Scan000
Detector000
Data1D
Ch000
Data
X_axis
Move000
Scan_x_axis
Logger

Parameter	Value
CLASS	CARRAY
TITLE	

Fig. 4.7: Loading the trace node from a .h5 file.

4.4.3.2.2 Converting raw data to be used in the retriever

If the non-linear trace was measured using another acquisition program than PyMoDAQ, we are slightly disappointed but will nonetheless explain how to convert it the proper format before being loaded into the retriever. PyMoDAQ-Femto uses a binary format known as hdf5, as described in the PyMoDAQ documentation.

We provide an example script to convert raw data, located inside the PyMoDAQ-Femto module, under `pymodaq_femto/utils/convert_to_pymodaq_compatible.py`.

If you are using **conda** with a dedicated environment as suggested in the *Installation* section, this folder will be located inside the `site-packages` folder, for instance in Windows something like: `C:/Miniconda/envs/your_environment_name/Lib/site-packages/pymodaq_femto` This `/utils` folder is also easily accessible on GitHub. If it is not already the case, raw data should be converted to numpy arrays. 5 arrays are needed:

- The 2D trace [N x M numpy array]
- An array corresponding to the parameter axis (delay in Frog, glass insertion in Dscan, etc.) in physical units [N x 1 numpy array]
- An array with the wavelength axis of the trace [M x 1 numpy array]
- The fundamental spectrum (spectrum of light before non-linear conversion) [P x 1 numpy array]
- The wavelength axis of the fundamental spectrum [P x 1 numpy array]

Note: The retriever has an option to rescale any input array, so parameter or wavelength axes can be saved in any units. That being said, it is usually easier to save all data in standard units everytime (meters for wavelengths and insertions, seconds for delays, etc.).

The fundamental spectrum doesn't need to be on the same wavelength axis as the 2D trace, they will get interpolated on a common axis during retrieval. The role of the fundamental is to compare retrieved spectrum with measured one (a good measure of the quality of retriever), and can also be used as an initial guess for the algorithm. But if you don't have one for every trace, just use any spectrum you have and the algorithm will still work.

Once the 5 numpy arrays are loaded, you can use the utility functions of `pymodaq_femto/utils/convert_to_pymodaq_compatible.py` to create a new .h5 file and add all data to it, with the proper structure.

Example: One raw DScan measurement (not measured with PyMoDAQ) is provided in `pymodaq_femto/utils/raw_scans/example_measured_dscan_to_convert.h5`. The 5 numpy arrays are stored in there. The example file `pymodaq_femto/utils/convert_to_pymodaq_compatible.py` converts this file into a PyMoDAQ-Femto-compatible h5 file.

The file is loaded, and the 5 numpy arrays are extracted:

```
parameter_axis = measured_dscan.axes[0]
spectrum_trace_axis = measured_dscan.axes[1]
spectrum_fundamental_intensity = raw_spectrum.intensity
spectrum_fundamental_axis_wavelength = raw_spectrum.wl
trace_data = measured_dscan.data
```

Since this example is a DScan trace, the parameter is the insertion of glass in the beam, expressed in meters. For a FROG trace, the parameter would be the time delay between two pulses, in seconds.

Then the script initializes a new .h5 file and gives it the correct structure:

```
saver = PyMoDAQFemtoCustomSaver()

# Open file and create scan node
saver.init_file(addhoc_file_path=str(pathToSave.joinpath(fileName)), update_h5=True)
scannode = saver.add_scan_group()
scannode.set_attr('scan_type', "Scan1D")
```

And finally we add data to it, using the convenience functions:

```
# Add all data
saver.add_exp_parameter(scannode, parameter_axis, label='Insertion', units='m')
saver.add_exp_trace(scannode, trace_data, spectrum_trace_axis)
saver.add_exp_fundamental(scannode, spectrum_fundamental_intensity, spectrum_fundamental_
↪axis_wavelength)
saver.close_file()
```

The script should save a converted file into `pymodaq_femto/utils/converted_scans/`, that can be directly loaded into the retriever.

4.4.4 Pre-processing data

4.4.4.1 Method definition and data rescaling

Once the data is loaded, the parameter tree on the right of the interface gets populated with several values. When working with data generated by the simulator module, everything will be set correctly. However, it won't be the case for experimental data, and some parameters are important to set correctly:

- **Algorithm Options:**
- Method: `(type: list)` The type of measurement (FROG, DSCAN, etc.). See *Available methods* for a full list of available methods. **This must match the method experimentally!**

- NL process: `(type: list)` The non-linear process to use (second harmonic generation, third harmonic generation, etc.). **This must match the method experimentally!**
- **Data Info**
  - Trace Info
  - Wavelength Scaling `(type: float)` Scaling to convert the wavelength axis of the experimental trace into meters.
  - Parameter Scaling `(type: float)` Scaling to convert the parameter axis of the experimental trace into correct unit (meters for DScan, seconds for FROG, etc).
  - Spectrum Info
  - Wavelength Scaling `(type: float)` Scaling to convert the wavelength axis of the experimental fundamental spectrum into meters.

The scalings allows handling of data taken in any units. They also allow flexibility, for instance, the axis of the measured trace might be the position of a motor. You can use these scalings to convert it to delay, glass insertion, etc.

Note: After using the retriever on a daily basis in our labs, we find that in the majority of cases, weird behaviors come from incorrect scaling parameters, or having the wrong Method select (Frog instead of Dscan)! If the retrieved trace look like rubbish, the first thing to do is to look at the measured trace and verify that its axes are correct!

4.4.4.2 Grid settings

Then, you must choose the grid on which the pulse will be computed. The definition is made using both temporal or spectral property, depending on which is more convenient.

- Grid settings:
- lambda0 (nm): `(type: float)` The central wavelength of the grid in the spectral domain, in nanometres. By default, it uses the center of the measured fundamental spectrum.
- Npoints: `(type: list)` Number of points of the grid (both in spectral and temporal domain). More points will require more type to retrieve the pulse, but increases the spectral resolution and the width of grid in the temporal domain.
- Time resolution (fs): `(type: float)` The time between two points of the grid in the temporal domain.

4.4.4.3 Background and region of interest

- Trace limits:

If you tick this box, a rectangle will appear on the measured trace. It allows to select the region of interest that will be used for the retrieval. You can either move and resize the rectangle, as shown below, or use the x0, y0, width, height parameters in the parameter tree.

- Substract trace background:

If you tick this box, two vertical lines will appear on the measured trace. The data between these lines will be averages and used as the background for the trace. You can either move the two lines, or use the wl0, wl1 parameters in the parameter tree. **The background region must be inside the region of interest**.

- Substract spectrum background:

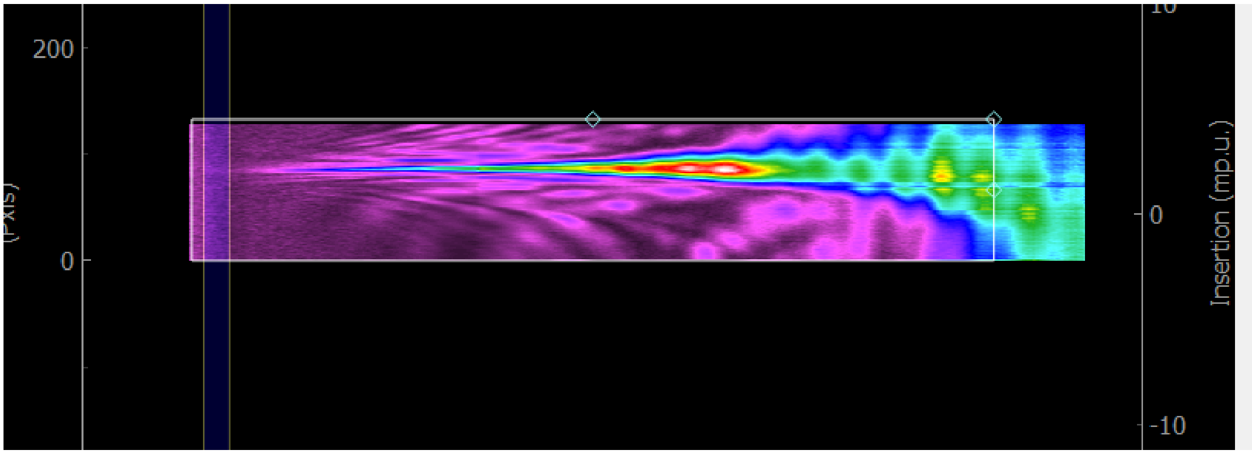


Fig. 4.8: Selecting the region of interest (rectangle) and background region (vertical lines) of a measured trace.

Same as the trace, but for the fundamental spectrum.

Once all this is set, press the button *Process Both* to proceed. This will take you to the next tab (Processed Data) which shows you data ready to be retrieved. The pulse displayed in time and frequency on the right is the Fourier Transform-limited pulse obtained from your fundamental spectrum, interpolated on the defined grid. Verify that the trace looks good, and that your temporal and spectral grids are adequate and proceed to the retrieval.

4.4.5 Running the retrieval algorithm

Here there are a few parameters to choose from. We recommend sticking with default ones for the most part, but feel free to experiment with everything

- **Retrieving**
  - Algo type: `(type: list)` Retrieval algorithm to use. Recommended: copra. These are implemented in pypret, the whole list of available algorithm is shown here: pypret.retrieval doc

Note:

1. We have only tested the software with the COPRA algorithm [Geib2019], which is universal and works with all non-linear methods. If you experiment with other algorithms and can provide feedback, let us know.
2. Currently all algorithms use their default parameters defined by pypret. Most of them have parameters that typically allow balancing the convergence speed and accuracy. If there is a need to tune the algorithm parameters, feel free to reach out or share it if you implement it yourself!

  - Verbose Info: `(type: bool)` If ticked, the retrieval error will be written at each iteration of the algorithm.
  - Max iteration: `(type: int)` Short description
  - Uniform spectral response: `(type: bool)` Short description
  - Keep spectral intensity fixed `(type: bool)` Short description
  - Initial guess: `(type: list)` Short description
  - Initial Pulse Guess `(type: group)` Short description

- FWHM (fs): `(type: float)` Short description
- Phase amp. (rad): `(type: float)` Short description

4.4.6 Analyzing results: metrics and propagation of the retrieved pulse

4.4.7 Saving and exporting data

4.4.8 Saving and recalling settings between sessions

4.5 Contributors

Here is a list of the main contributors:

4.5.1 Main modules

4.5.1.1 Simulator

- Sébastien Weber, Research Engineer at CEMES/CNRS

4.5.1.2 Retriever

- Sébastien Weber, Research Engineer at CEMES/CNRS
- Romain Géneaux, Researcher at CEA Saclay, LIDYL

4.5.1.3 Propagator

- Romain Géneaux, Researcher at CEA Saclay, LIDYL

4.5.1.4 Cleaning

- Sébastien Weber, Research Engineer at CEMES/CNRS
- Romain Géneaux, Researcher at CEA Saclay, LIDYL

4.5.2 Documentation

- Romain Géneaux, Researcher at CEA Saclay, LIDYL

4.5.3 You want to contribute?

If you're willing to help, you can clone the up-to-date GitHub repo: https://github.com/PyMoDAQ/pymodaq_femto using git command line or GitHub Desktop. I advise to create a dedicated conda environment for this and install PyMoDAQ-femto's package as a developer:

- `conda create -n dev_env`
- `conda activate dev_env`
- `cd` to the location of the folder where you downloaded or cloned the repository.
- install the package as a developer using the command `pip install -e .`.

Any change on the code will be *seen* by python interpreter. When ready, you can ask to push your code into the main development branch. A simpler way is to raise *Issues* on PyMoDAQ-femto's github page.