



**DataLab**  
*Release 0.20.0*

Apr 28, 2025



# CONTENTS

<b>1</b>	<b>Getting started</b>	<b>3</b>
1.1	Installation . . . . .	3
1.2	Use cases, main features and key strengths . . . . .	9
1.3	Key features . . . . .	11
1.4	Tutorials . . . . .	14
<b>2</b>	<b>Features</b>	<b>81</b>
2.1	Validation . . . . .	81
2.2	General features . . . . .	90
2.3	Signal processing . . . . .	137
2.4	Image processing . . . . .	164
<b>3</b>	<b>API</b>	<b>201</b>
3.1	Algorithms ( <code>cdl.algorithms</code> ) . . . . .	201
3.2	Parameters ( <code>cdl.param</code> ) . . . . .	220
3.3	Object model ( <code>cdl.obj</code> ) . . . . .	264
3.4	Computation ( <code>cdl.computation</code> ) . . . . .	302
3.5	Proxy objects ( <code>cdl.proxy</code> ) . . . . .	382
3.6	GUI . . . . .	401
3.7	Main window . . . . .	402
3.8	Panel . . . . .	409
3.9	Action handler . . . . .	427
3.10	Object view . . . . .	432
3.11	Plot handler . . . . .	435
3.12	ROI editor . . . . .	438
3.13	Processor . . . . .	439
3.14	Docks . . . . .	456
3.15	HDF5 I/O . . . . .	457
<b>4</b>	<b>Contributing</b>	<b>459</b>
4.1	Share your ideas and experiences . . . . .	459
4.2	Share your scientific/technical knowledge . . . . .	459
4.3	Contribute to new features . . . . .	460
4.4	Develop new features . . . . .	460
	<b>Python Module Index</b>	<b>505</b>





DataLab is an **open-source platform for signal and image processing and visualization** for research, education and industry. Leveraging the richness of the scientific Python ecosystem<sup>1</sup>, DataLab is the ideal complement to your data analysis workflows as it can be extended with your Python code through *Plugins* or directly from *your IDE* or *your Jupyter notebooks*. Go to *Installation* to get started!

To immediately see DataLab in action, you have two options:

- Read or view our *Tutorials*,
- Try DataLab online, without installation, using our *Binder environment*.

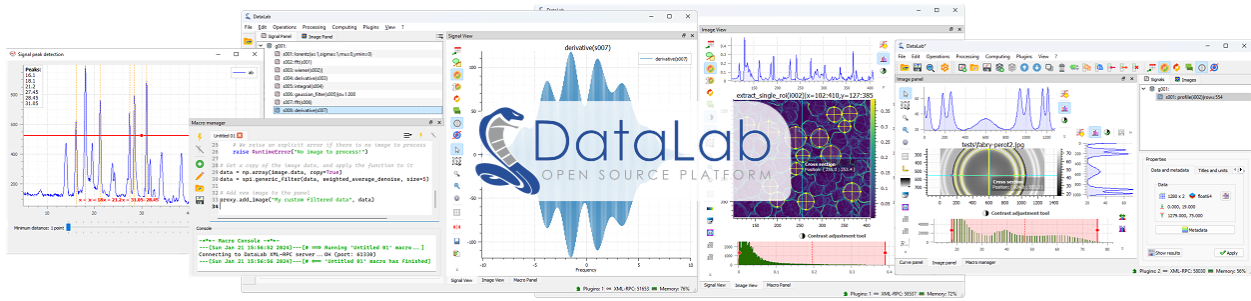


Fig. 1: Signal and image visualization in DataLab

DataLab has been funded, chronologically, by the following takeholders:

**CEA**, the French Alternative Energies and Atomic Energy Commission, is the major investor in DataLab, and is the main contributor to the project.

**CODRA**, a software engineering and editor firm, has supported DataLab open-source journey since its inception (see [here](#)).

**NLnet Foundation**, as part of the NGIO Commons Fund, backed by the European Commission, has funded the redesign of DataLab's core architecture.



Fig. 2: DataLab is powered by **PlotPyStack**, the scientific Python-Qt visualization and graphical user interface stack.

<sup>1</sup> DataLab processing features are mainly based on **NumPy**, **SciPy**, **scikit-image**, **OpenCV** and **PyWavelets** libraries. DataLab visualization capabilities are based on **PlotPyStack** toolkit, a set of Python libraries for building scientific applications with Qt graphical user interfaces.



## GETTING STARTED

DataLab is an open platform for signal and image processing, designed to be used by scientists, engineers, and researchers in academia and industry, while offering the reliability of industrial-grade software. It is a versatile software that can be used for a wide range of applications, from simple data analysis to complex signal processing and image analysis tasks.

DataLab integrates seamlessly into your workflow thanks to three main operating modes:

**Stand-alone application**, with a graphical user interface that allows you to interact with your data and visualize the results of your analysis in real time.

**Python library**, allowing you to integrate DataLab functions (or graphical user interfaces) into your own Python scripts and programs or Jupyter notebooks.

**Remotely controlled** from your own software, or from an IDE (e.g., Spyder) or a Jupyter notebook, using the DataLab API.

DataLab leverages the power of Python and its scientific ecosystem, through the use of the following libraries:

- [NumPy](#) for numerical computing (arrays, linear algebra, etc.)
- [SciPy](#) for scientific computing (interpolation, special functions, etc.)
- [scikit-image](#) and [OpenCV](#) for image processing
- [PyWavelets](#) for wavelet transform
- [PlotPyStack](#) for Qt-based interactive data visualization

### 1.1 Installation

This section provides information on how to install DataLab on your system. Once installed, you can start DataLab by running the `cdl` command in a terminal, or by clicking on the DataLab shortcut in the Start menu (on Windows).

**See also:**

For more details on how to execute DataLab and its command-line options, see [Command line features](#).

### 1.1.1 How to install

DataLab is available in several forms:

- As a *Conda package*.
- As a Python package, which can be installed using the *Package manager pip*.
- Windows As a stand-alone application, which does not require any Python distribution to be installed. Just run the *All-in-one installer* and you're good to go!
- Windows Within a ready-to-use *Python distribution*, based on *WinPython*.
- As a precompiled *Wheel package*, which can be installed using `pip`.
- As a *Source package*, which can be installed using `pip` or manually.

**See also:**

Impatient to try the next version of DataLab? You can also install the latest development version of DataLab from the master branch of the Git repository. See *Development version* for more information.

#### Conda package

GNU/Linux Windows macOS

To install `datalab` package from the *conda-forge* channel (<https://anaconda.org/conda-forge/datalab>), run the following command:

```
$ conda install conda-forge::datalab
```

#### Package manager pip

GNU/Linux Windows macOS

DataLab's package `cdl` is available on the Python Package Index (PyPI) on the following URL: <https://pypi.python.org/pypi/cdl>.

Installing DataLab from PyPI with Qt is as simple as running this command (you may need to use `pip3` instead of `pip` on some systems):

```
$ pip install cdl[qt]
```

Or, if you prefer, you can install DataLab without the Qt library (not recommended):

```
$ pip install cdl
```

**Note:** If you already have a previous version of DataLab installed, you can upgrade it by running the same command with the `--upgrade` option:

```
$ pip install --upgrade cdl[qt]
```

## All-in-one installer

### Windows

DataLab is available as a stand-alone application for Windows, which does not require any Python distribution to be installed. Just run the installer and you're good to go!

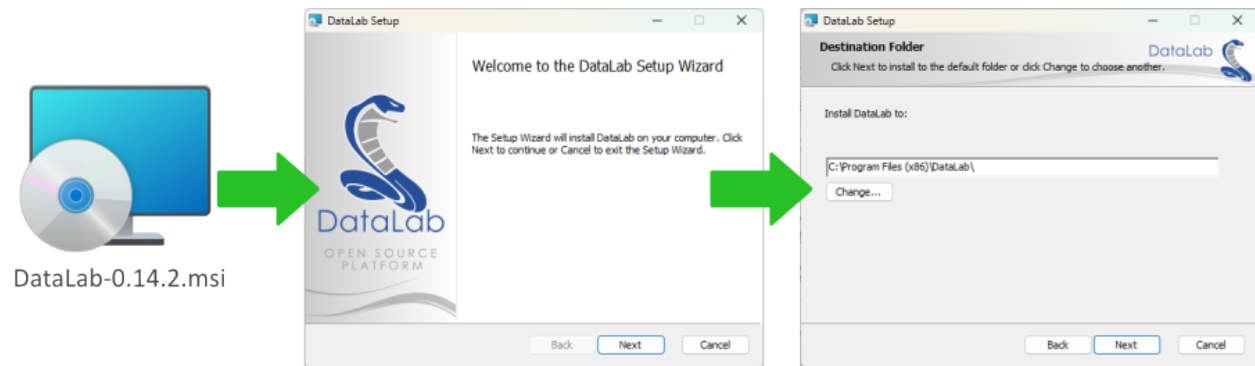


Fig. 1: DataLab all-in-one installer for Windows

The installer package is available in the [Releases](#) section. It supports automatic uninstall and upgrade feature (no need to uninstall DataLab before running the installer of another version of the application).

**Warning:** DataLab Windows installer is available for Windows 7 SP1, 8, 10 and 11.

On Windows 7 SP1, before running DataLab (or any other Python 3 application), you must install Microsoft Update *KB2533623 (Windows6.1-KB2533623-x64.msu)* and also may need to install [Microsoft Visual C++ 2015-2022 Redistributable package](#).

## Python distribution

### Windows

DataLab is also available within a ready-to-use Python distribution, based on [WinPython](#). This distribution is called [DataLab-WinPython](#) and is available in the [DataLab-WinPython Releases](#) section.



Fig. 2: DataLab-WinPython is a ready-to-use Python distribution including the DataLab platform.

The main difference with the all-in-one installer is that you can use the Python distribution for other purposes than running DataLab, and you may also extend it with additional packages. On the downside, it is also *much bigger* than the all-in-one installer because it includes a full Python distribution.

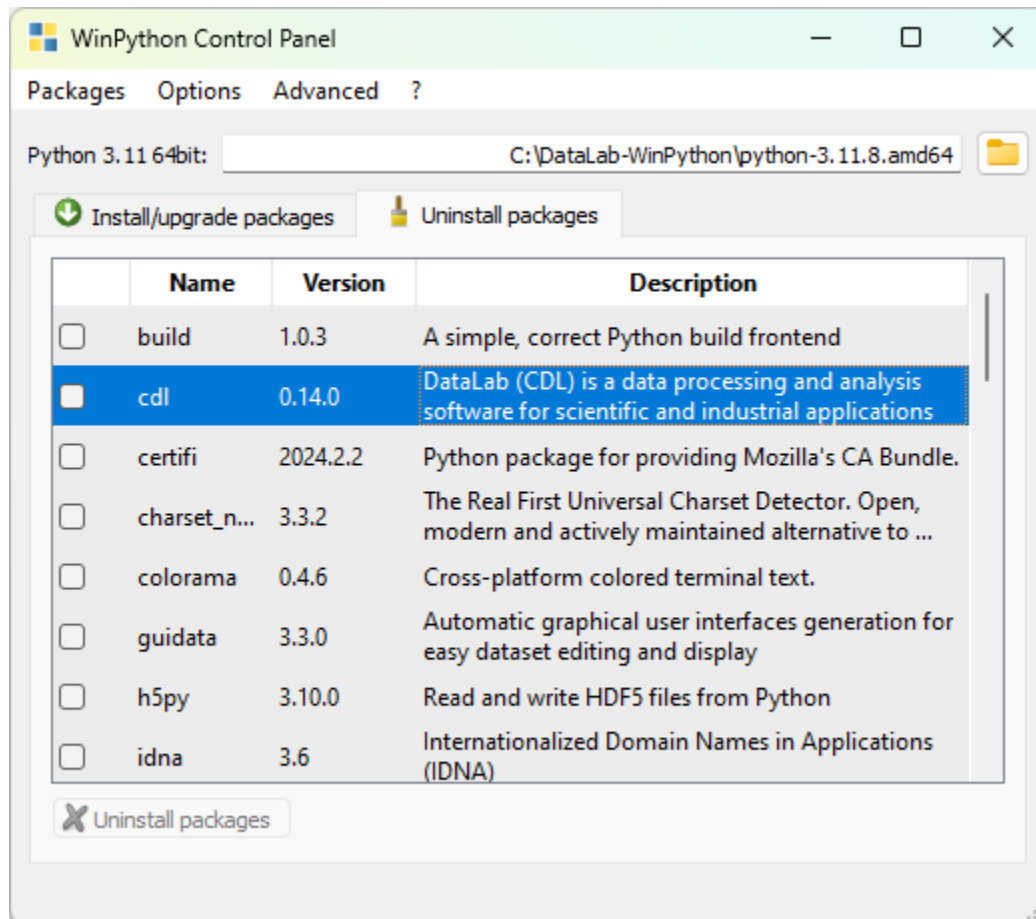


Fig. 3: DataLab-WinPython Control Panel

**Warning:** Whereas the all-in-one installer provides a monolithic package that guarantees the compatibility of all its components because it cannot be modified by the user, the WinPython distribution is more flexible and thus can be broken by a bad manipulation of the Python distribution by the user. This should be taken into account when choosing the installation method.

## Wheel package

GNU/Linux Windows macOS

On any operating system, using pip and the Wheel package is the easiest way to install DataLab on an existing Python distribution:

```
$ pip install --upgrade DataLab-0.11.1-py2.py3-none-any.whl
```

## Source package

GNU/Linux Windows macOS

Installing DataLab directly from the source package may be done using pip:

```
$ pip install --upgrade cdl-0.11.1.tar.gz
```

Or, if you prefer, you can install it manually by running the following command from the root directory of the source package:

```
$ pip install --upgrade .
```

Finally, you can also build your own Wheel package and install it using pip, by running the following command from the root directory of the source package (this requires the `build` and `wheel` packages to be installed):

```
$ pip install build wheel # Install build and wheel packages (if needed)
$ python -m build # Build the wheel package
$ pip install --upgrade dist/cdl-0.11.1-py2.py3-none-any.whl # Install the wheel package
```

## Development version

GNU/Linux Windows macOS

If you want to try the latest development version of DataLab, you can install it directly from the master branch of the Git repository.

The first time you install DataLab from the Git repository, enter the following command:

```
$ pip install git+https://github.com/DataLab-Platform/DataLab.git
```

Then, if at some point you want to upgrade to the latest version of DataLab, just run the same command with options to force the reinstall of the package without handling dependencies (because it would reinstall all dependencies):

```
$ pip install --force-reinstall --no-deps git+https://github.com/DataLab-Platform/
↪DataLab.git
```

---

**Note:** If dependencies have changed, you may need to execute the same command as above, but without the `--no-deps` option.

---

## 1.1.2 Dependencies

---

**Note:** The DataLab all-in-one installer already include all those required libraries as well as Python itself.

---

The `cdl` package requires the following Python modules:

Name	Version	Summary
Python	>=3.9, <4	Python programming language
guidata	>= 3.9	Automatic GUI generation for easy dataset editing and display
PlotPy	>= 2.7.4	Curve and image plotting tools for Python/Qt applications
SciPy	>= 1.5, < 1.15.0	Fundamental algorithms for scientific computing in Python
scikit-image	>= 0.18	Image processing in Python
pandas	>= 1.2	Powerful data structures for data analysis, time series, and statistics
Py-Wavelets	>= 1.1	PyWavelets, wavelet transform module
psutil	>= 5.7	Cross-platform lib for process and system monitoring in Python. NOTE: the syntax of this script MUST be kept compatible with Python 2.7.
PyQt5	>=5.11	Python bindings for the Qt cross platform application toolkit

Optional modules for development:

Name	Version	Summary
ruff		An extremely fast Python linter and code formatter, written in Rust.
pylint		python code static checker
Coverage		Code coverage measurement for Python
pyinstaller	>=6.0	PyInstaller bundles a Python application and all its dependencies into a single package.

Optional modules for building the documentation:

Name	Version	Summary
PyQt5		Python bindings for the Qt cross platform application toolkit
sphinx		Python documentation generator
sphinx_intl		Sphinx utility that make it easy to translate and to apply translation.
sphinx-sitemap		Sitemap generator for Sphinx
myst_parser		An extended [CommonMark]( <a href="https://spec.commonmark.org/">https://spec.commonmark.org/</a> ) compliant parser,
sphinx_design		A sphinx extension for designing beautiful, view size responsive web components.
sphinx-copybutton		Add a copy button to each of your code cells.
pydata-sphinx-theme		Bootstrap-based Sphinx theme from the PyData community



Optional modules for running test suite:

Name	Version	Summary
pytest		pytest: simple powerful testing with Python
pytest-xvfb		A pytest plugin to run Xvfb (or Xephyr/Xvnc) for tests.

---

**Note:** Python 3.11 and PyQt5 are the reference for production release

---

## 1.2 Use cases, main features and key strengths

DataLab is a platform for data processing and visualization (signals or images) that includes many functions. Developed in Python, it benefits from the richness of the associated ecosystem in terms of scientific and technical libraries.

### 1.2.1 What are the applications for Datalab?

#### Real world examples

A few concrete and specific examples illustrate the nature of the work that can be carried out with DataLab:

- Processing of experimental data (signals and images) acquired on a scientific facility in the nuclear field
- Processing of data acquired by a sensor in an industrial context
- Processing of images acquired by a camera in a medical context
- Automatic detection of defects on a surface, in the context of quality control
- Automatic detection of laser spots on a target, in the context of laser alignment
- Instrument alignment through image processing
- Automatic pattern detection on images and geometric correction of the images, in the context of non destructive testing

#### Usage modes

Depending on the application, DataLab can be used in three different modes:

- **Stand-alone mode:** DataLab is a full-fledged processing application that can be adapted to the client's needs through the addition of industry-specific plugins.
- **Embedded mode:** DataLab is integrated into your application to provide the necessary processing and visualization features.
- **Remote-controlled mode:** DataLab communicates with your application, allowing it to benefit from its functionality without disrupting the user experience.

## Use cases

### See also:

For practical examples of use cases, see the [Tutorials](#) section:

- Most of the tutorials are describing concrete examples of use of DataLab in a scientific or technical context.
- Regarding the use of DataLab with an IDE (Integrated Development Environment) such as Visual Studio Code or Spyder, see the tutorial [DataLab and Spyder: a perfect match](#).
- As for the use of DataLab with Jupyter notebooks, that is one of the topics covered in the tutorial [Add your own features](#).

DataLab is a versatile tool that can be used in different contexts:

### Data processing

DataLab is a powerful tool for processing signals and images. It can be used to develop complex algorithms, or to quickly prototype a processing chain.

See our [Tutorials](#) for practical examples of use in data processing.

### Companion tool for scientific/technical work

DataLab can be used as a companion tool for scientific/technical work. It allows you to visualize and process data, and to share your results with your colleagues. It can easily be adapted to your needs through the addition of plugins, and it may even be used together with your every day tools (e.g., Visual Studio Code, Spyder... or Jupyter notebooks).

See our [Tutorials](#) for practical examples of use in a scientific/technical context.

### Prototyping a data processing application

DataLab can be used to quickly prototype a data processing application. It can then be used as a basis for the development of a full-fledged application.

See the tutorial [Prototyping a custom processing pipeline](#) for a concrete example.

### Debugging a data processing application

DataLab can be used as an advanced debugging tool for your data processing applications, independently from the development environment or the language used (Python, C#, C++, etc.). All you need is to be able to communicate with DataLab via its remote control interface (standard XML-RPC protocol). This allows you to send data to DataLab (signals, images or even geometric shapes), visualize the data at each step of the processing chain, manipulate them to better understand the behavior of your algorithms, and even modify them to test the robustness of your code.

See the tutorial [Debugging your algorithm with DataLab](#) for a quick overview of this feature.

---

**Note:** DataLab can also be controlled from your familiar development environment (e.g., Visual Studio Code, Spyder...) or from a Jupyter notebook, in order to perform calculations using your processing functions while leveraging the advanced features of DataLab. See the tutorials [Prototyping a custom processing pipeline](#) or [DataLab and Spyder: a perfect match](#) for examples of use.

---

With its user-friendly experience and versatile usage modes, DataLab enables efficient development of your data processing and visualization applications while benefiting from an industrial-grade technological platform.

### 1.2.2 Main features

The main technical features of DataLab include:

- Support for numerous standard and proprietary data formats
- Opening an arbitrary number of objects (signals or images) for batch processing, with the possibility of defining groups of objects
- Simultaneous viewing of multiple objects with annotation support
- Standard operations and processing on signals and images
- Advanced image processing (restoration, morphology, edge detection, etc.)
- Management of multiple regions of interest (calculations, extractions)
- Macro-command editor
- Remote-controllable API
- Embedded interactive Python console

### 1.2.3 Key strengths

To summarize, the four key strengths of DataLab are:

#### **Extensibility**

The DataLab plugin system makes it easy to code new features (specific processing, specific file formats, custom graphical interfaces). It can also be used as a customizable platform.

#### **Interoperability**

DataLab can also be embedded in your own application. For example, within data processing software, machine-level control systems, or test bench applications.

#### **Automation**

A high-level public API allows for full remote control of DataLab to open and process data.

#### **Maintainability and testability**

DataLab is an industrial-grade scientific and technical processing software. The built-in automated tests in DataLab cover 90% of its features, which is significant for software with graphical interfaces and helps mitigate regression risks. Moreover, the test suite includes validation tests based either on ground truth data or analytical solutions.

#### **See also:**

See section [Validation](#) for more information on DataLab's validation strategy.

## 1.3 Key features

This page presents briefly DataLab key features.

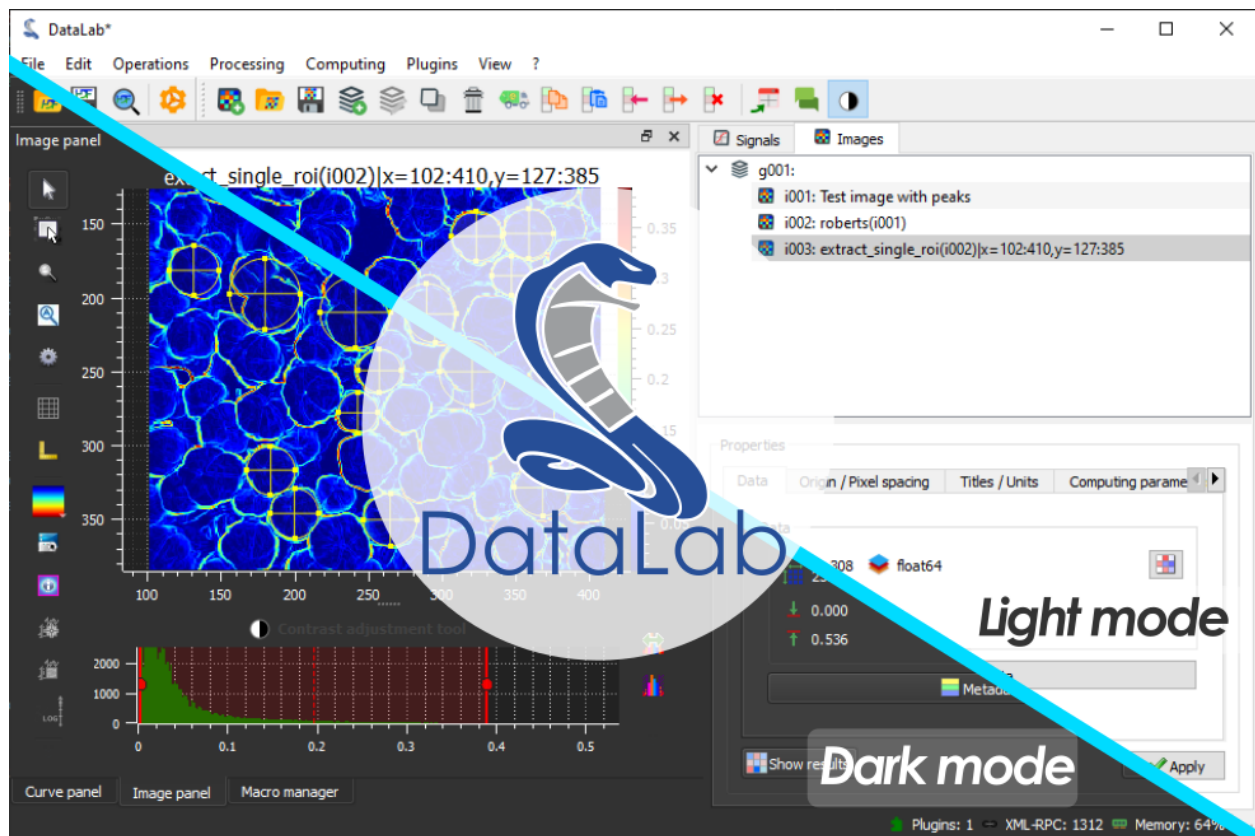


Fig. 4: DataLab supports dark and light mode depending on your platform settings (this is handled by the `guidata` package, and may be overridden by setting the `QT_COLOR_MODE` environment variable to `dark` or `light`).

### 1.3.1 Data visualization

Signal	Image	Feature
✓	✓	Screenshots (save, copy)
✓	Z-axis	Lin/log scales
✓	✓	Data table editing
✓	✓	Statistics on user-defined ROI
✓	✓	Markers
	✓	Aspect ratio (1:1, custom)
	✓	50+ available colormaps (customizable)
	✓	Intensity profiles (line, average, radial)
✓	✓	Annotations
✓	✓	Persistence of settings in workspace
	✓	Distribute images on a grid
✓	✓	Single or superimposed views

### 1.3.2 Data processing

Signal	Image	Feature
✓	✓	Process isolation for running computations
✓	✓	Remote control from Jupyter, Spyder or any IDE
✓	✓	Remote control from a third-party application
✓	✓	Sum, average, difference, product...
✓	✓	Operations with a constant
✓	✓	Square root, power, logarithm, exponential...
✓		Derivative, integral
✓	✓	ROI extraction, Swap X/Y axes
✓		Semi-automatic multi-peak detection
✓		Convolution
	✓	Flat-field correction
	✓	Flip, rotation, scaling...
	✓	Intensity profiles (line, average, radial)
	✓	Pixel binning
✓	✓	Linear calibration
✓	✓	Normalization, Clipping, Offset correction
✓		Reverse X-axis
	✓	Thresholding (manual, Otsu...)
✓	✓	Gaussian filter, Wiener filter
✓	✓	Moving average, moving median
✓	✓	FFT, inverse FFT, Power/Phase/Magnitude spectrum, Power Spectral Density
✓		Interpolation, resampling
✓		Detrending
✓		X-Y mode
✓		Interactive fit: Gauss, Lorentz, Voigt, polynomial, CDF...
✓		Interactive multigaussian fit
✓		Frequency filters (low-pass, high-pass, band-pass, band-stop)
✓		Windowing (Hamming, Hanning...)
	✓	Butterworth filter
	✓	Exposure correction (gamma, log...)

continues on next page

Table 1 – continued from previous page

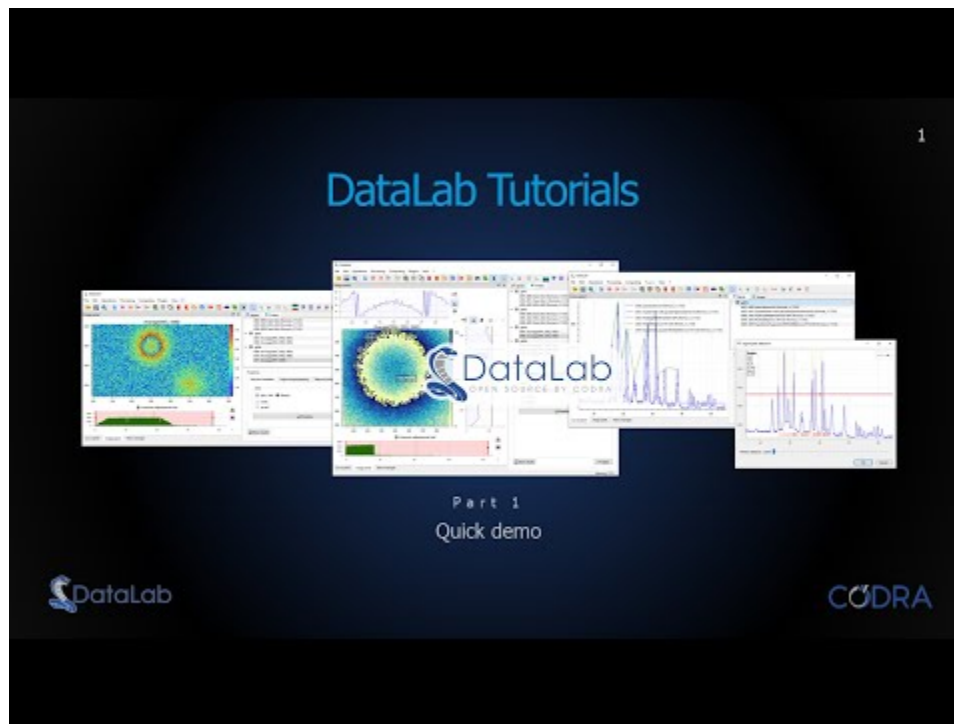
Signal	Image	Feature
	✓	Restauration (Total Variation, Bilateral...)
	✓	Morphology (erosion, dilation...)
	✓	Edges detection (Roberts, Sobel...)
✓	✓	Analysis on custom ROI
✓		FWHM, FW @ $1/e^2$
✓		Dynamic parameters (ENOB, SNR...), Sampling period/Rate
	✓	Centroid (robust method w/r noise)
	✓	Minimum enclosing circle center
	✓	2D peak detection
	✓	Contour detection
	✓	Circle Hough transform
	✓	Blob detection (OpenCV, Laplacian of Gaussian...)

## 1.4 Tutorials

### 1.4.1 Video Tutorials

DataLab video tutorials intend to provide a complementary material to the documentation and other tutorials available [here](#).

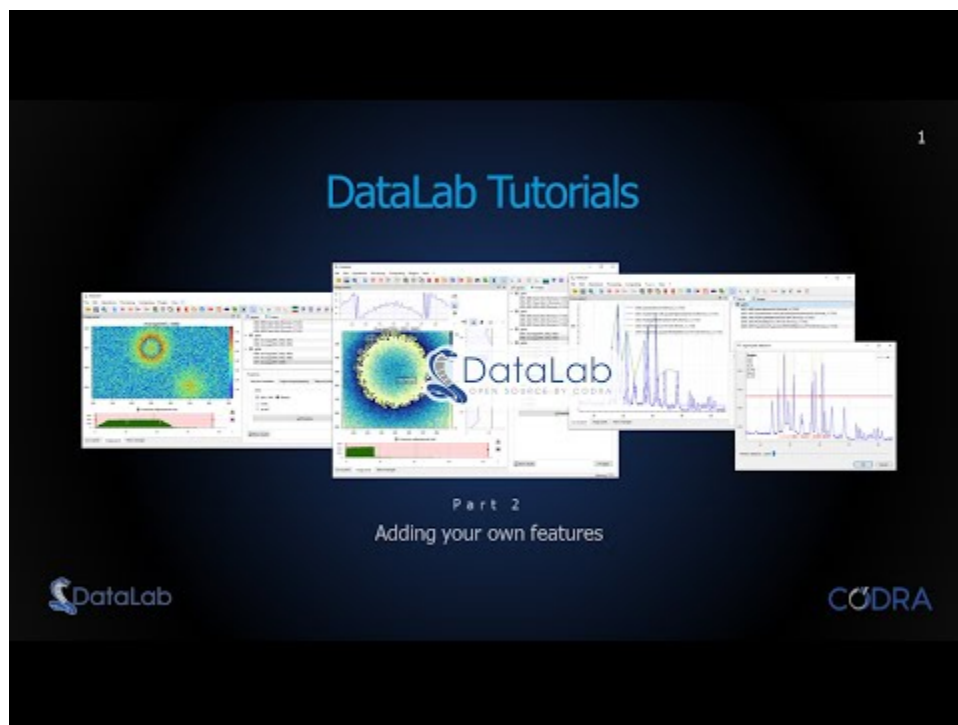
#### Quick demo



In this first video, we will quickly go through the main features of DataLab, and show how to do some basic signal and image processing.

**Warning:** This video is intentionally short and does not cover all the features of DataLab. It is meant to give you a quick overview of the software. For a more detailed description of the features, please watch the other videos or read the documentation.

## Add your own features



In this tutorial, we will show how to add your own features to DataLab using three different approaches:

1. Macro-commands, using the integrated macro manager
2. Remote control of DataLab from an external IDE (e.g. Spyder) or a Jupyter notebook
3. Plugins

The first common point between these three approaches is that they all rely on DataLab's high-level API, which allow to interact with almost every aspect of the software. This API is here accessed using Python scripts, but it may also be accessed using any other language when using the remote control approach (because it relies on a standard communication protocol, XML-RPC).

The second common point is that they all use Python code, and compatible proxy objects, so that the same code can be at least partially reused in the three approaches.

## 1.4.2 Other Tutorials

The following tutorials are detailed step-by-step guides to perform specific tasks with DataLab, or to illustrate features of the software in the context of a scientific or technical problem. Each tutorial focuses on a specific aspect of the software and is intended to be self-contained.

### Processing a spectrum

This example shows how to process a spectrum with DataLab:

- Read the spectrum from a file
- Apply a filter to the spectrum
- Extract a region of interest
- Fit a model to the spectrum
- Save the workspace to a file

First, we open DataLab and read the spectrum from a file.

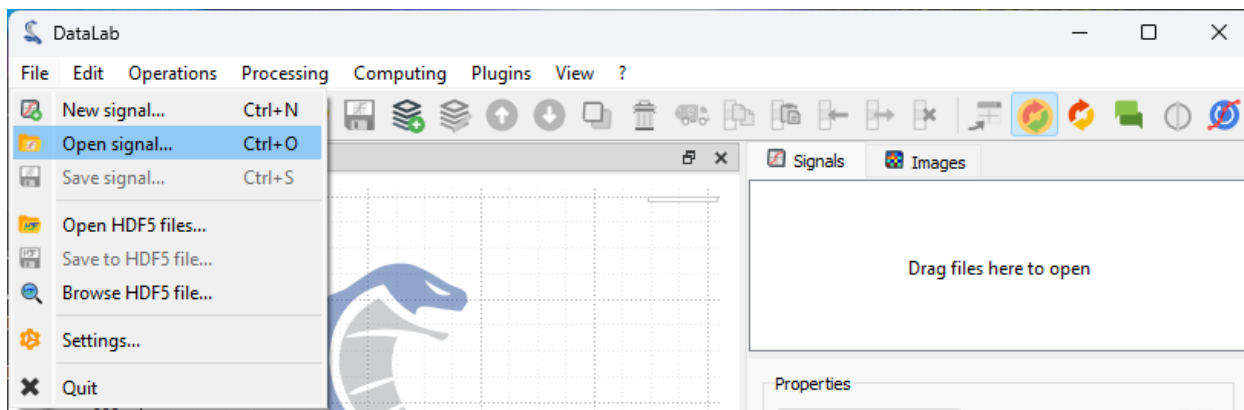


Fig. 5: Open the spectrum file with “File > Open...”, or with the button in the toolbar, or by dragging and dropping the file into DataLab (on the panel on the right).

Here, we are actually generating the signal from a test data file (using “Plugins > Test data > Load spectrum of paracetamol”), but the principle is the same.

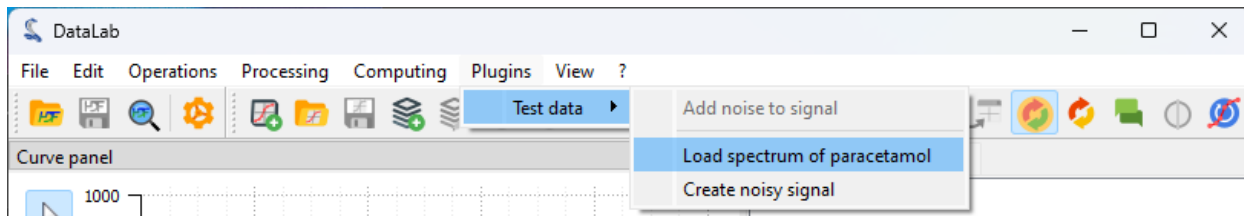


Fig. 6: Using the “Test data” plugin is a convenient way to generate test data for tutorials, but you can use any file containing a spectrum, such as a spectrum from a real experiment.

The spectrum is displayed in the main window.

Now, let’s process this spectrum by applying a filter to it. We will use a Wiener filter, which is a filter that can be used to remove noise from a signal, even if this is not absolutely necessary in this case.



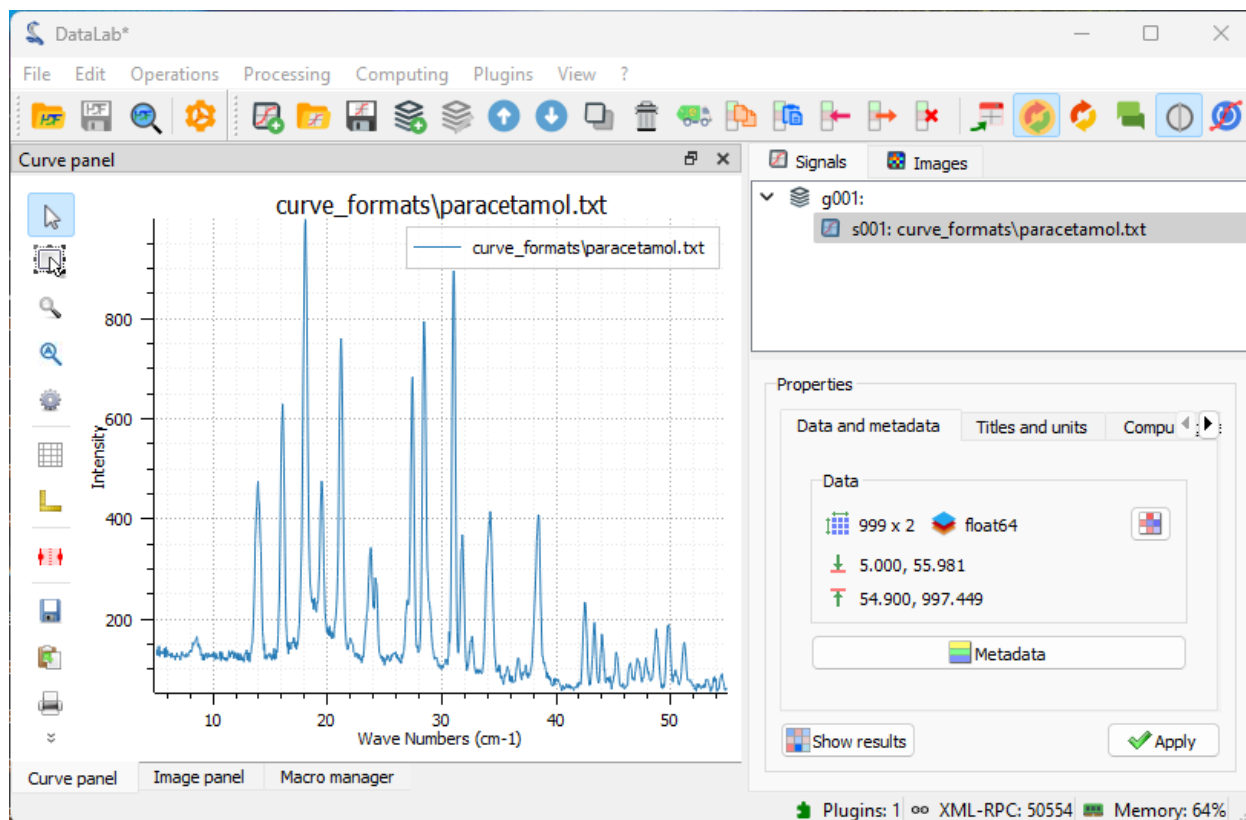


Fig. 7: The spectrum is a 1D signal, so it is displayed as a curve. The horizontal axis is the energy axis, and the vertical axis is the intensity axis.

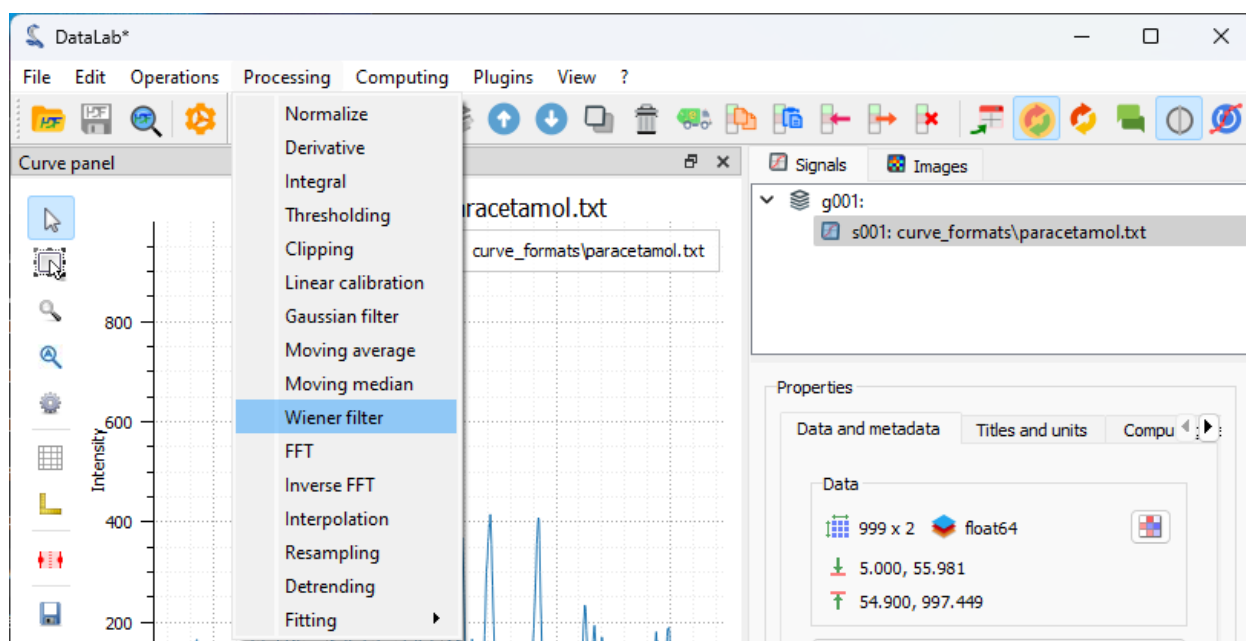


Fig. 8: Open the filter window with “Processing > Wiener filter”.

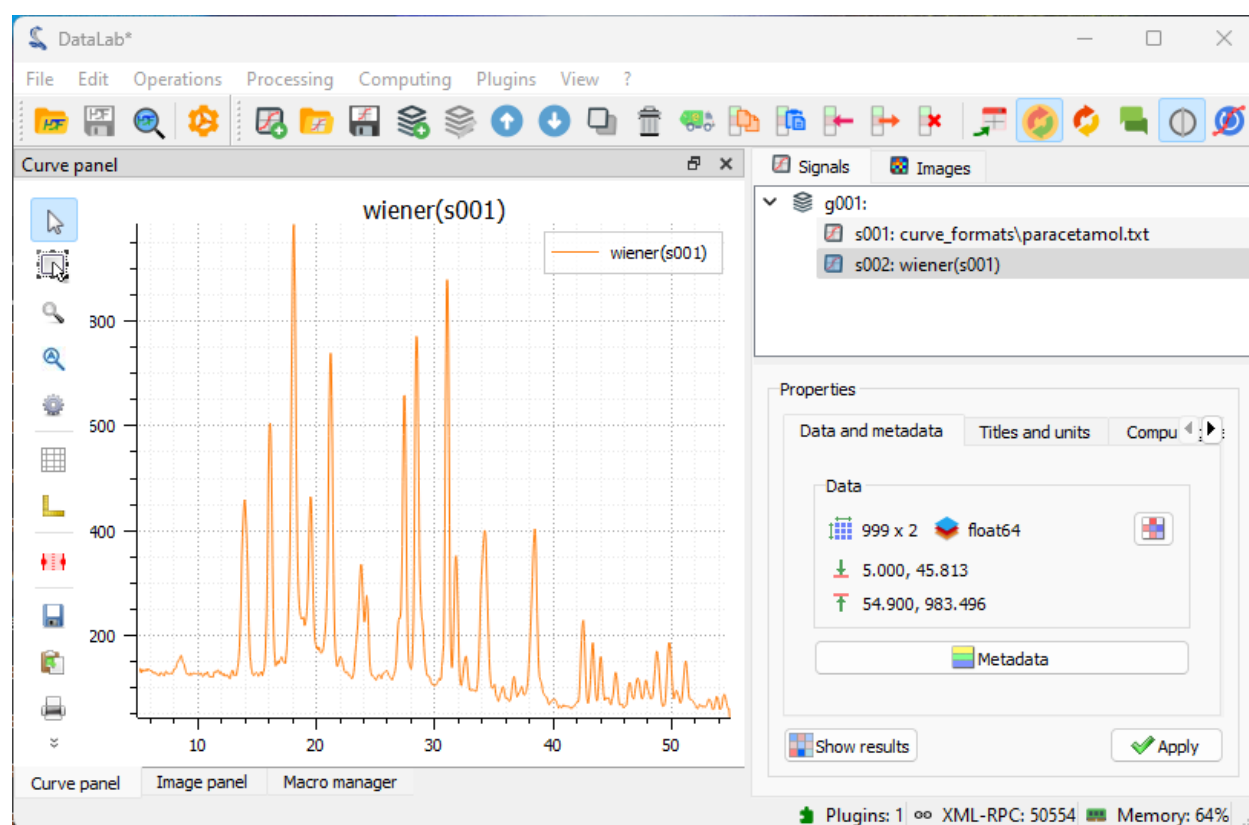


Fig. 9: The result of the filter is displayed in the main window.

If we want to analyze a specific region of the spectrum, we can extract it from the spectrum using the “ROI extraction” feature from the “Operations” menu.

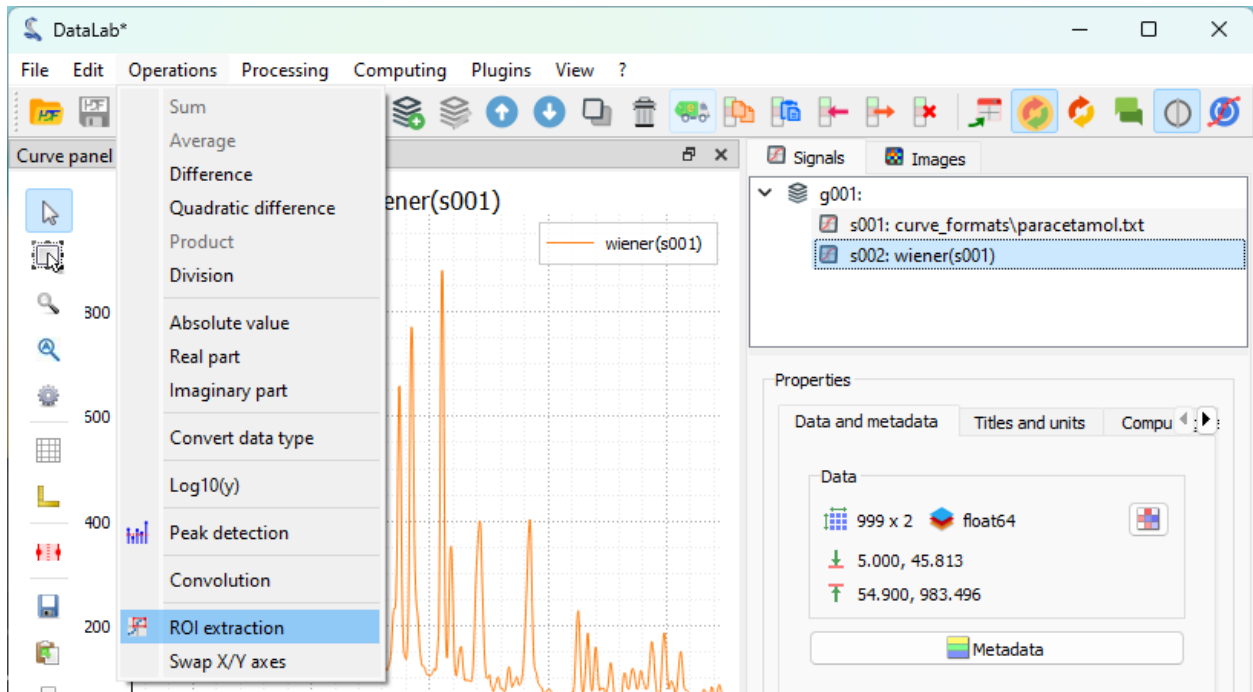


Fig. 10: Open the ROI extraction window with “Operations > ROI extraction”.

Let’s try to fit a model to the spectrum. We will use a Gaussian model, which is a model that can be used to fit a peak in a spectrum.

To demonstrate another processing feature, we can also try to detrend the spectrum.

When analyzing a spectrum, it can be useful to try to identify the peaks in the spectrum. We can do this by fitting a multi-Gaussian model to the spectrum, using the “Processing > Fitting > Multi-Gaussian fit” feature.

We also could have used the “Peak detection” feature from the “Operations” menu to detect the peaks in the spectrum.

Finally, we can save the workspace to a file. The workspace contains all the signals that were loaded in DataLab, as well as the processing results. It also contains the visualization settings (curve colors, etc.).

If you want to load the workspace again, you can use the “File > Open HDF5 file...” (or the button in the toolbar) to load the whole workspace, or the “File > Browse HDF5 file...” (or the button in the toolbar) to load only a selection of data sets from the workspace.

## Detecting blobs on an image

This example shows how to detect blobs on an image with DataLab, and also covers other features such as the plugin system:

- Add a new plugin to DataLab
- Denoise an image
- Detect blobs on an image
- Save the workspace to a file

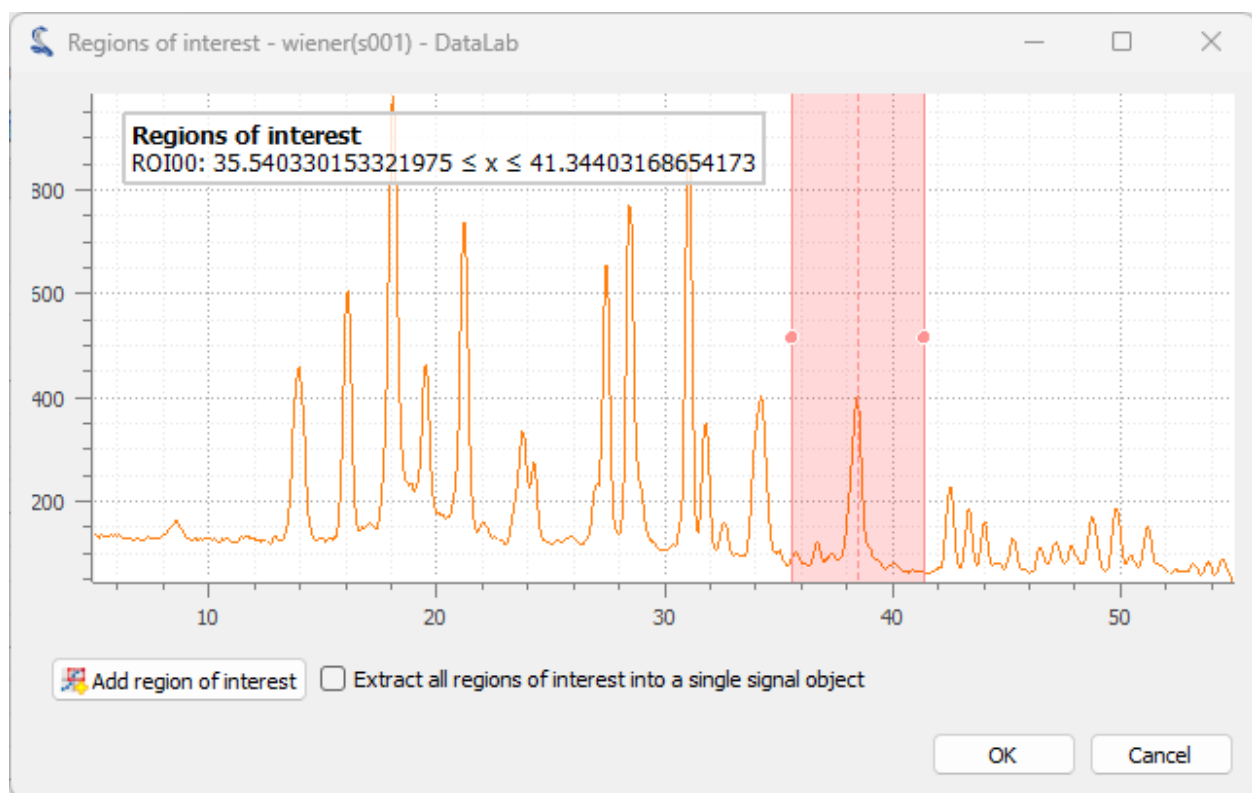


Fig. 11: The “Regions of interest” dialog box is displayed. Click on “Add ROI” and resize the horizontal window to select the area. Then, click on “OK”.

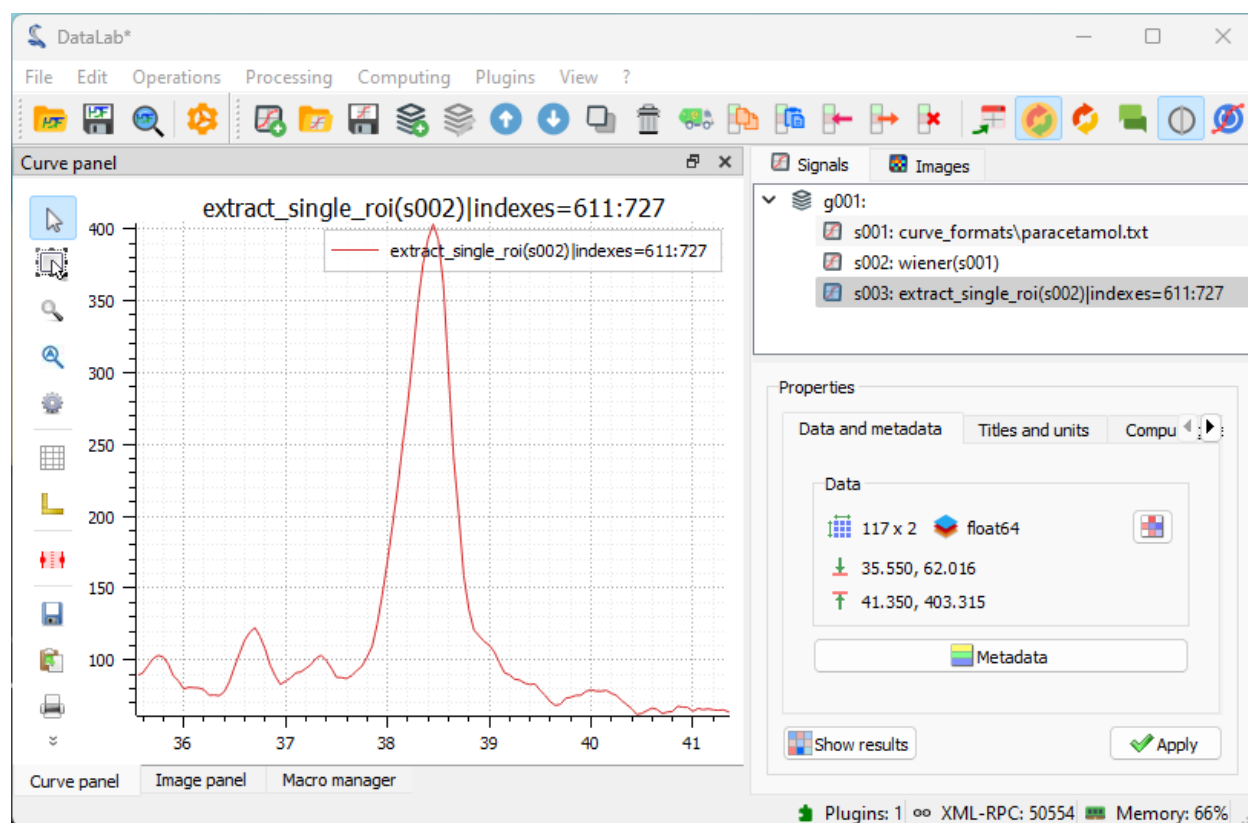


Fig. 12: The region of interest is displayed in the main window.

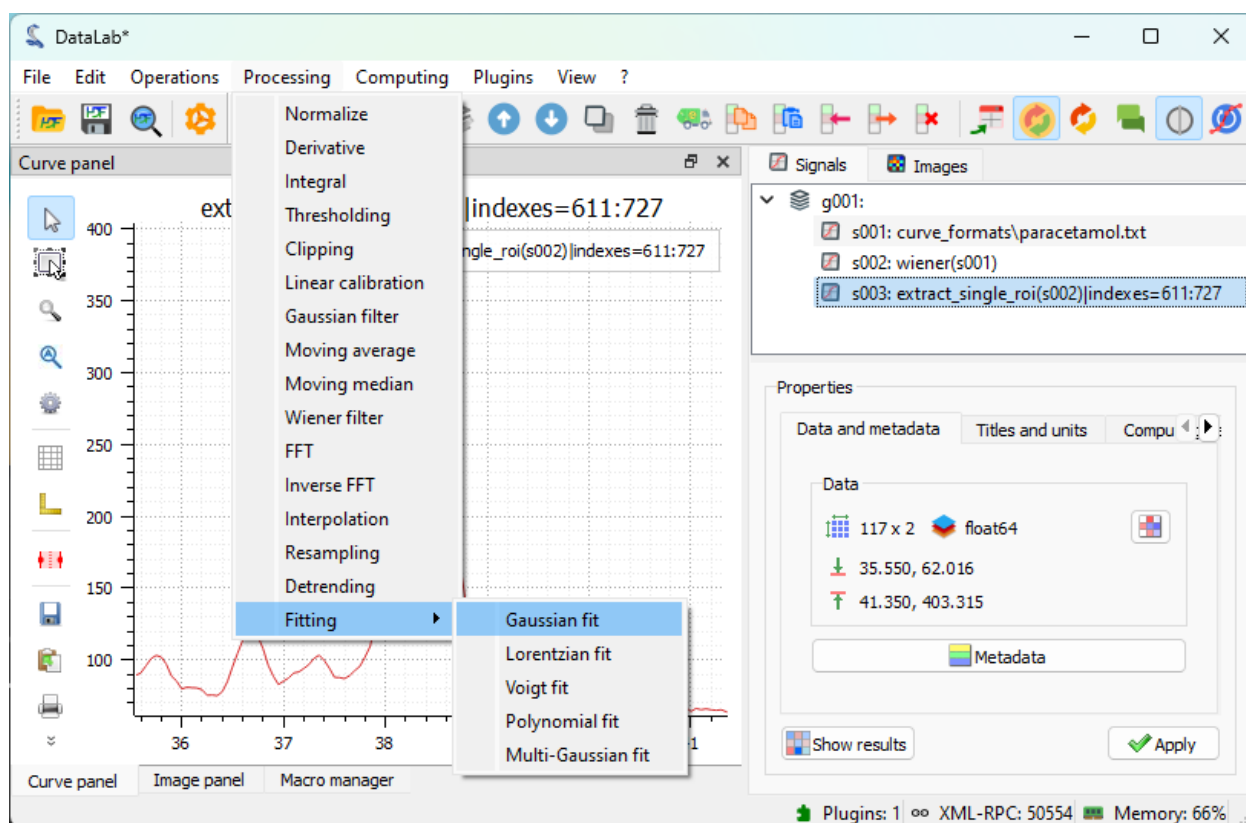


Fig. 13: Open the model fitting window with “Processing > Fitting > Gaussian fit”.

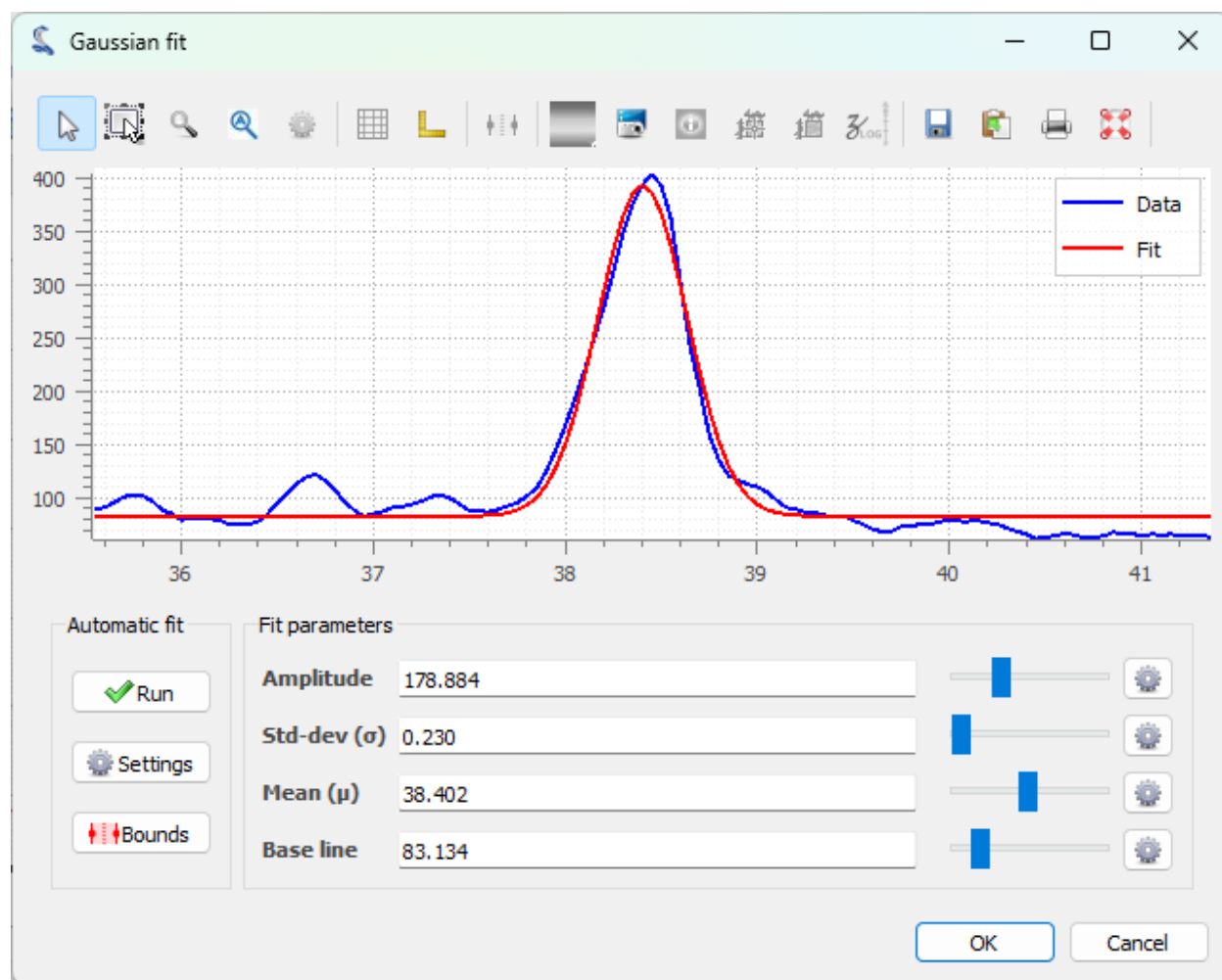


Fig. 14: The “Gaussian fit” dialog box is displayed. An automatic fit is performed by default. Click on “OK” (or eventually try to fit the model manually by adjusting the parameters or the sliders, or try to change the automatic fitting parameters).

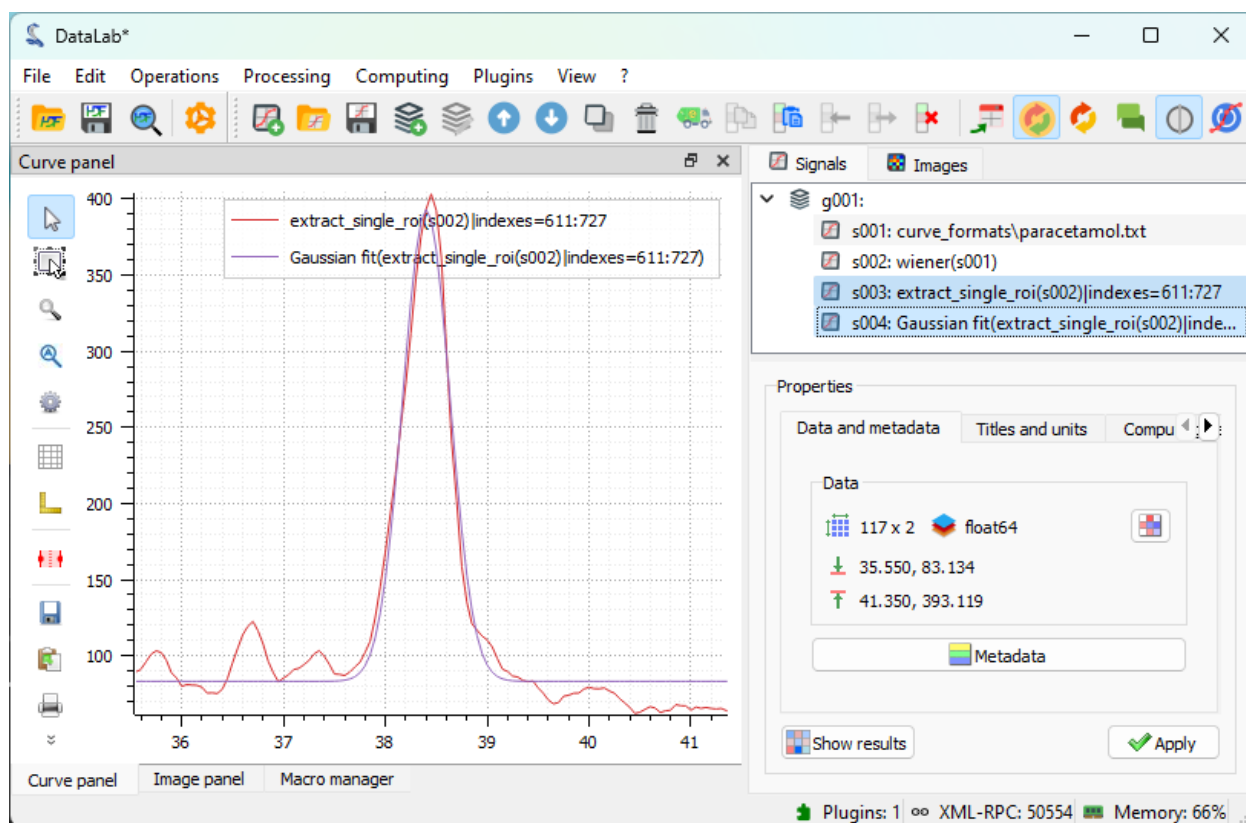


Fig. 15: The result of the fit is displayed in the main window. Here we selected both the spectrum and the fit in the “Signals” panel on the right, so both are displayed in the visualization panel on the left.



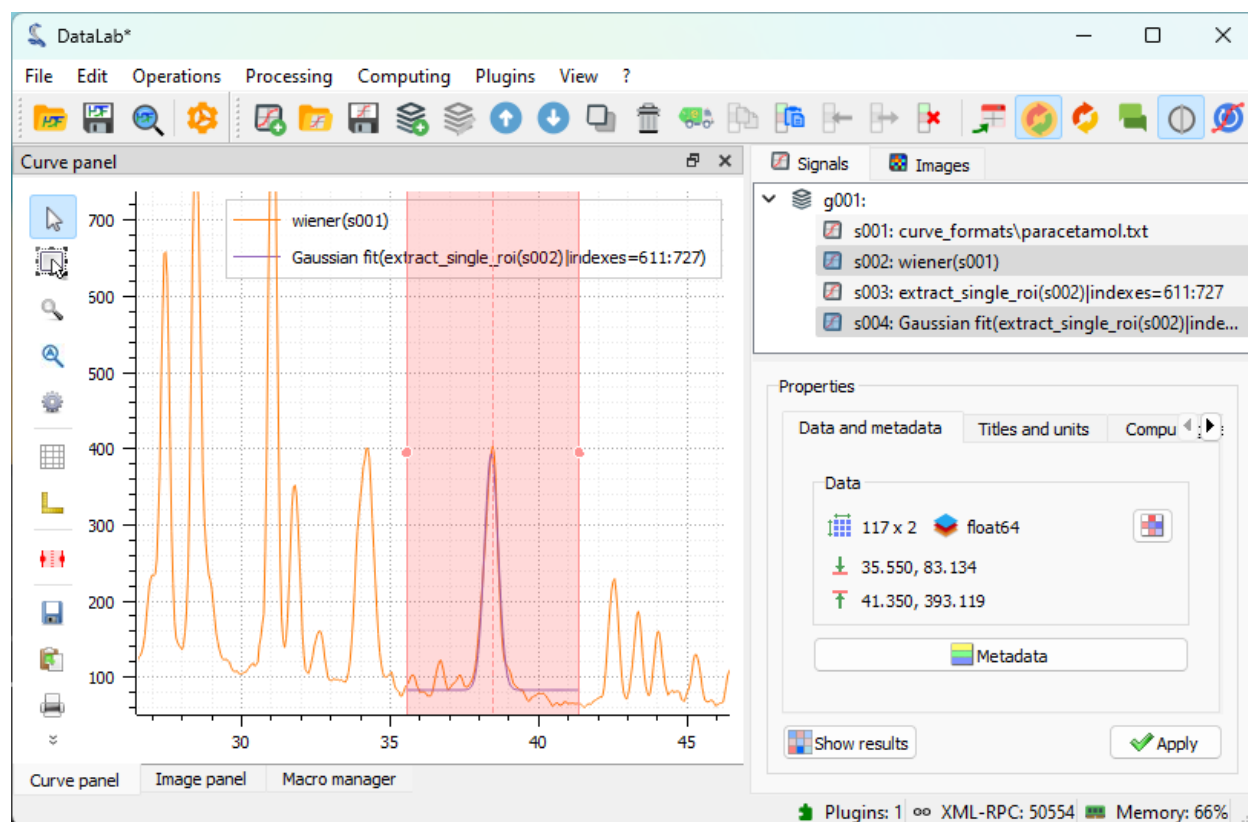


Fig. 16: We may also select the full spectrum and the fit in the “Signals” panel on the right, so that both are displayed in the visualization panel on the left, if this has a sense for the analysis we want to perform. Note that the full spectrum visualization also contains the region of interest we extracted previously.

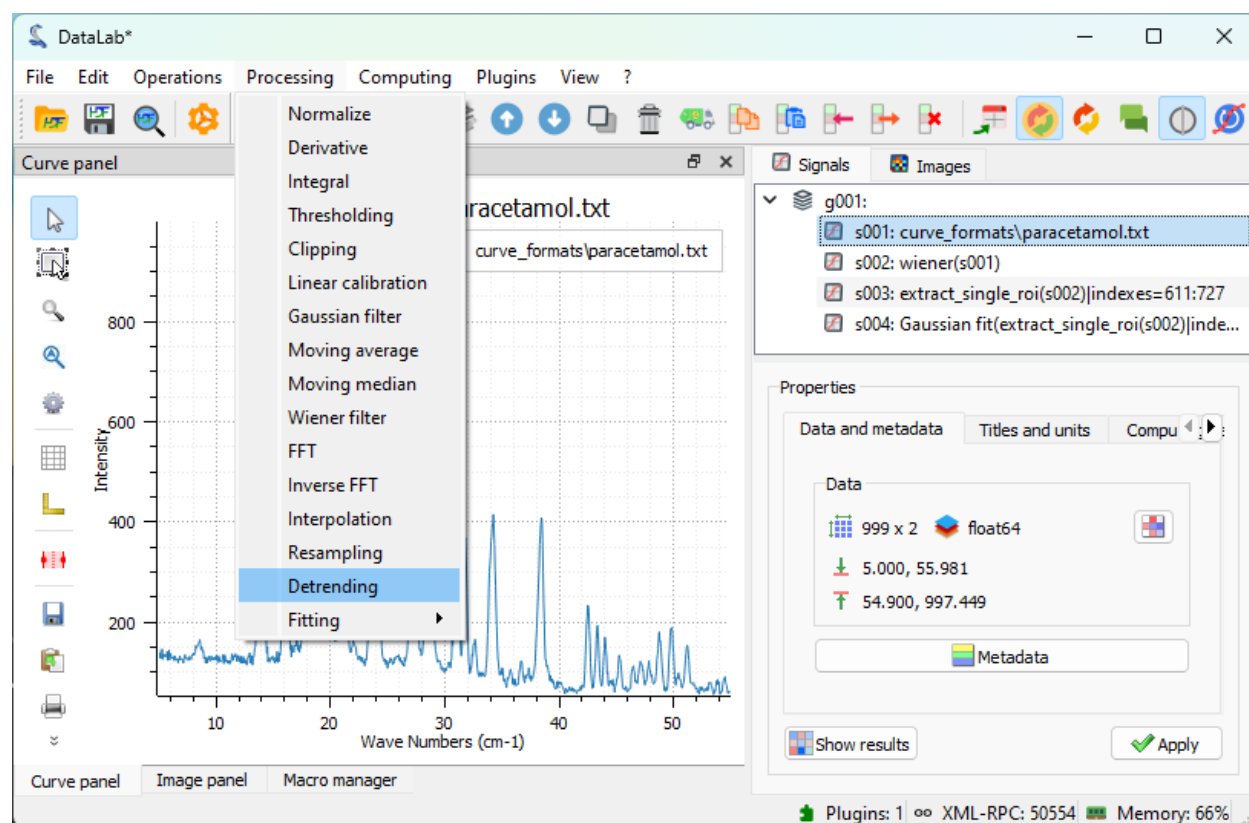


Fig. 17: Execute the “Processing > Detrending” feature.

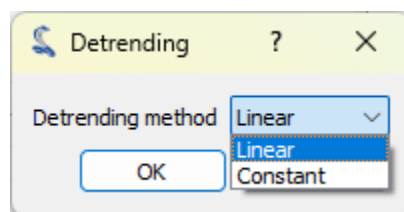


Fig. 18: We choose a linear detrending method, and we click on “OK”.

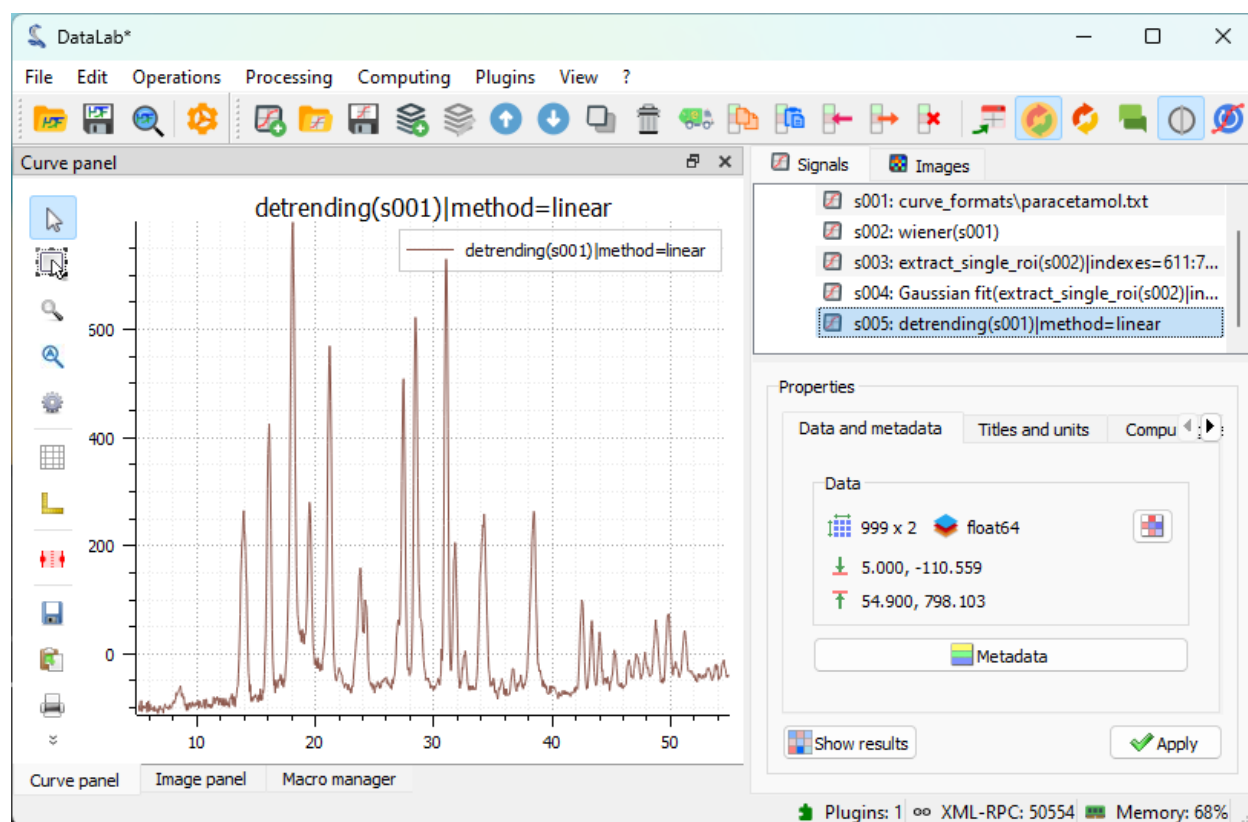


Fig. 19: The result of the detrending is displayed in the main window (in that specific case, the detrending may not be appropriate, but it is just to demonstrate the feature).

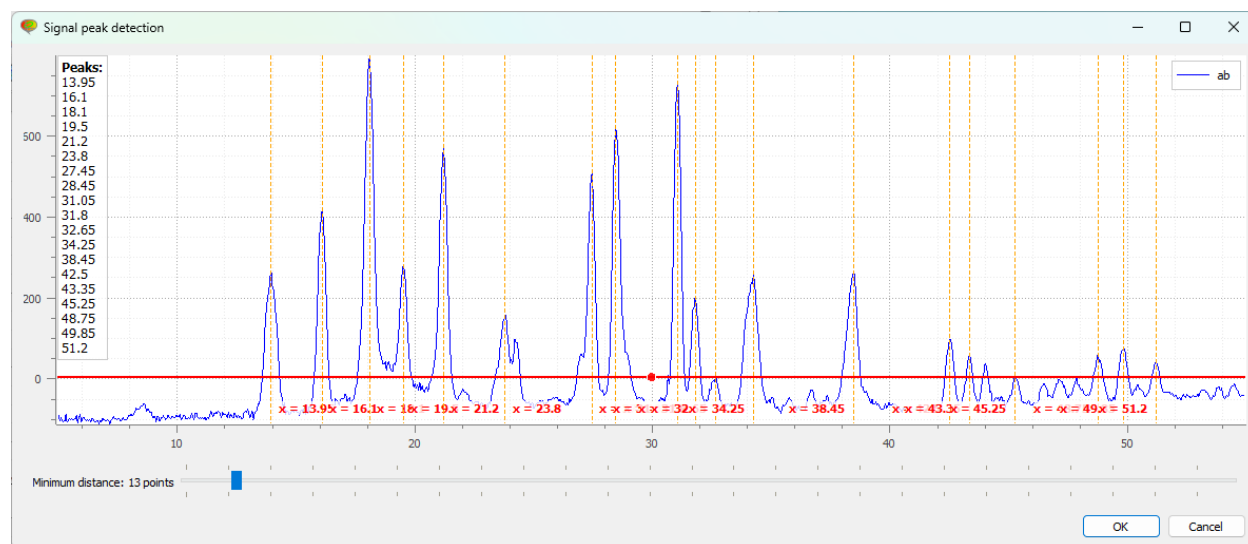


Fig. 20: First, a “Signal peak detection” dialog box is displayed. We can adjust the the vertical cursor position to select the threshold for the peak detection, as well as the minimum distance between two peaks. Then, we click on “OK”.

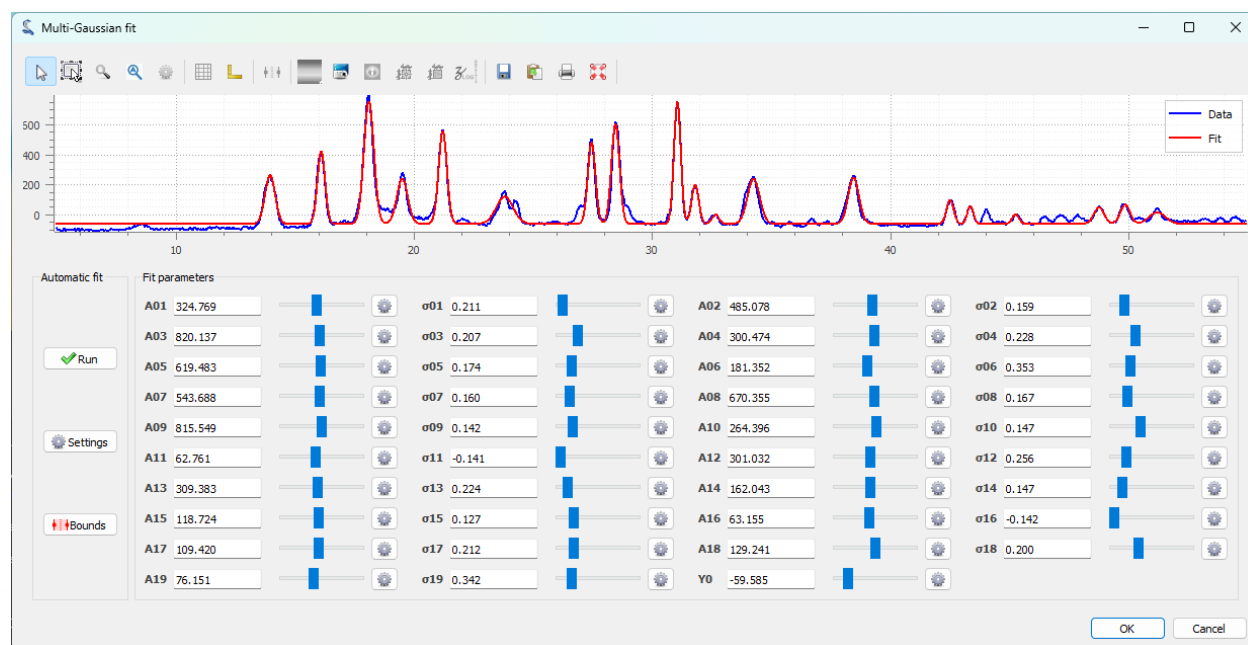


Fig. 21: The “Multi-Gaussian fit” dialog box is displayed. An automatic fit is performed by default. Click on “OK” (or eventually try to fit the model manually by adjusting the parameters or the sliders, or try to change the automatic fitting parameters).

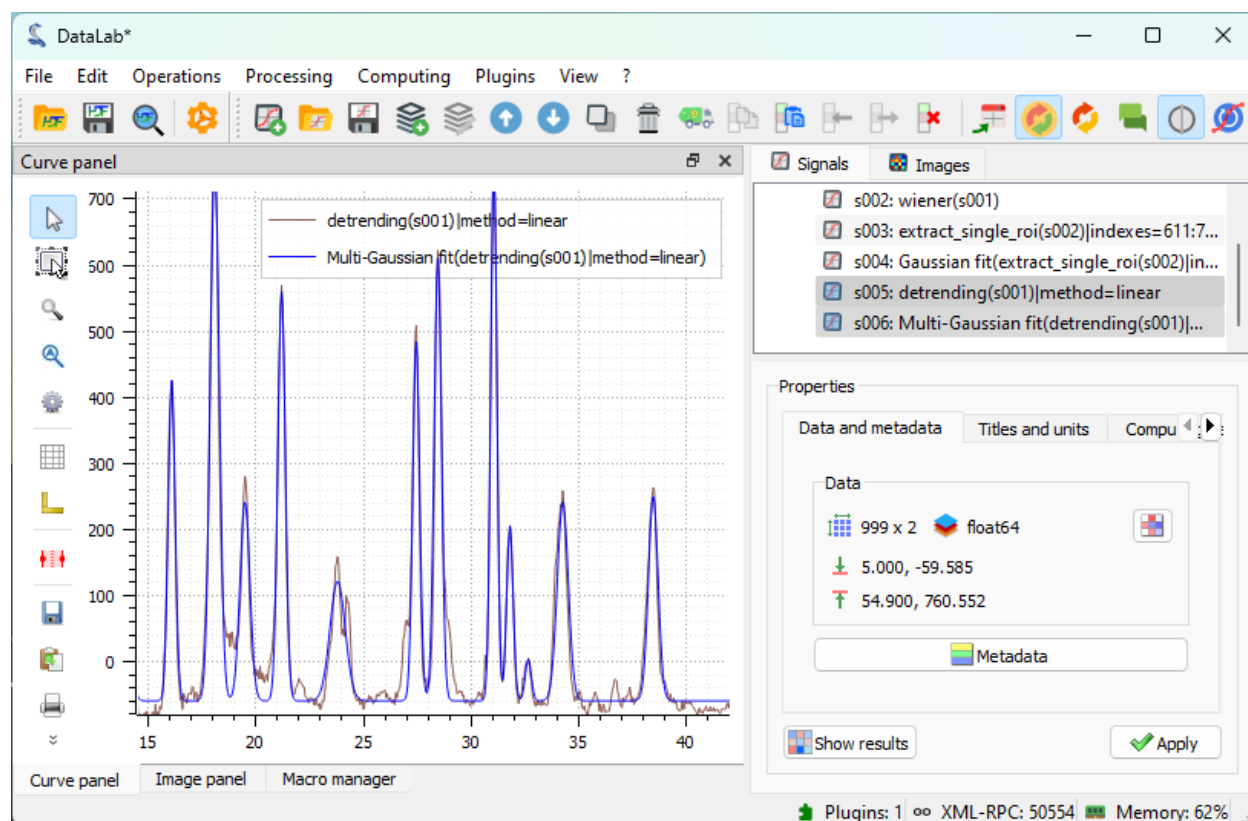


Fig. 22: The result of the fit is displayed in the main window. Here we selected both the spectrum and the fit in the “Signals” panel on the right, so both are displayed in the visualization panel on the left.

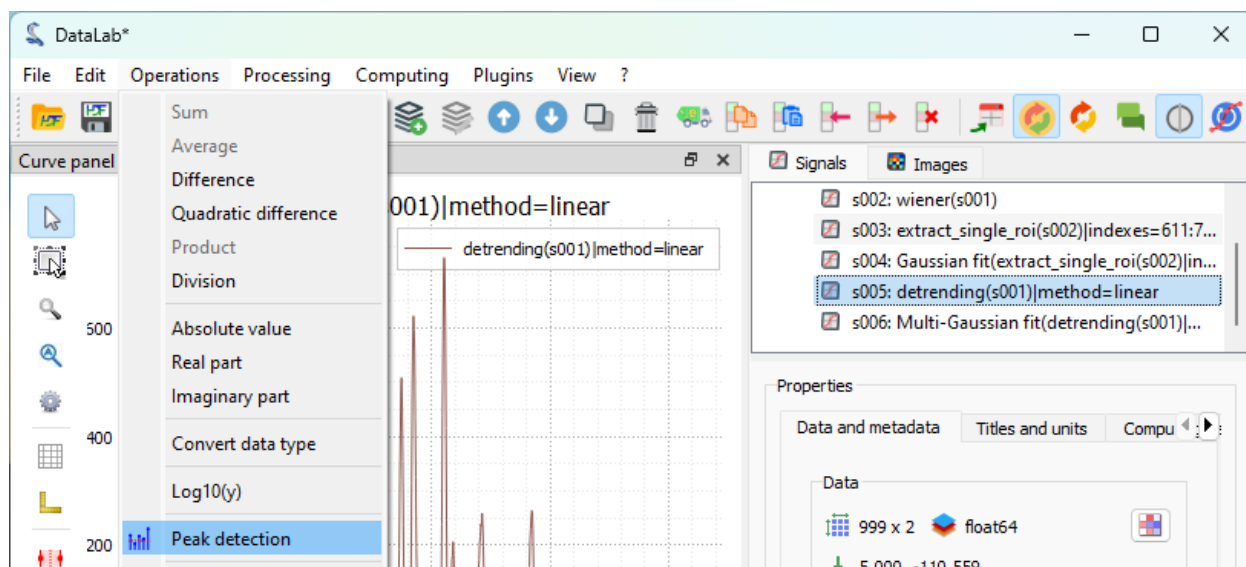


Fig. 23: Open the “Peak detection” window with “Operations > Peak detection”.

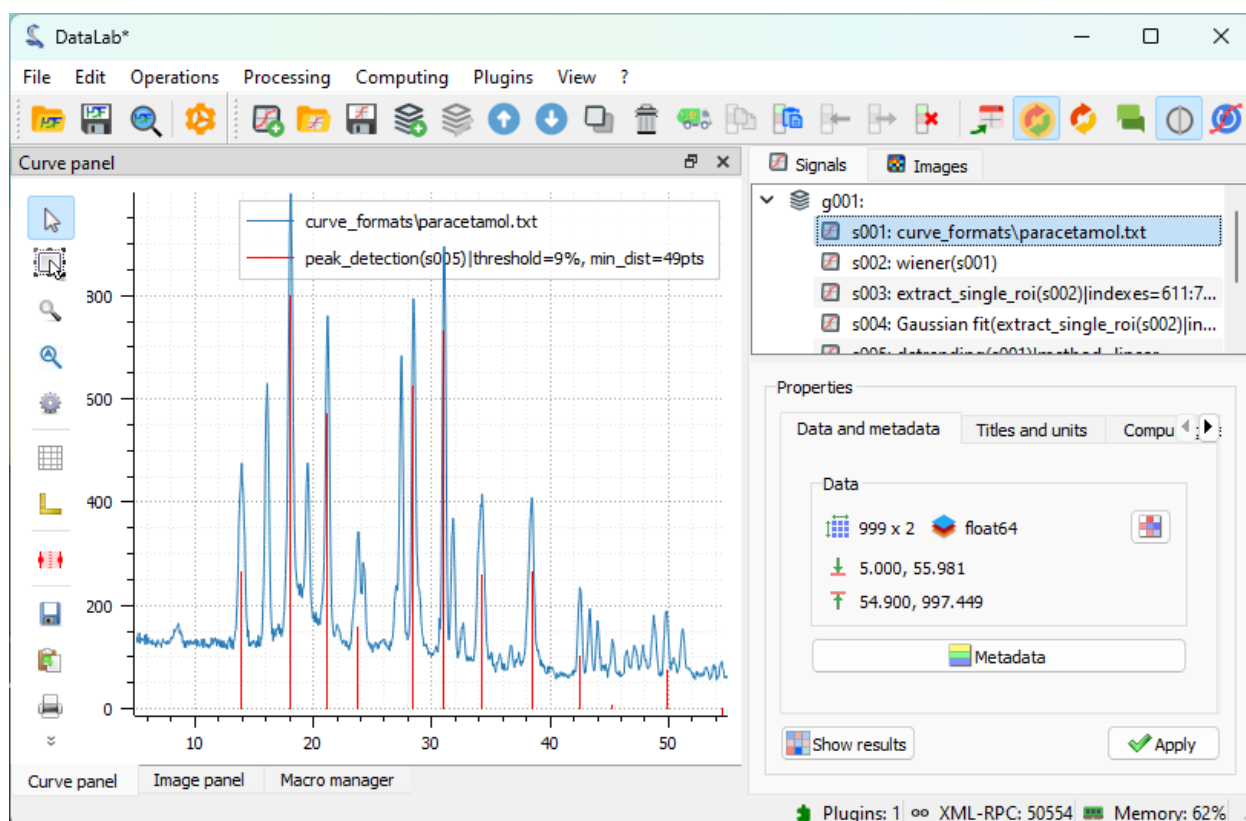


Fig. 24: After having adjusted the parameters of the peak detection dialog (same dialog as the one used for the multi-Gaussian fit), click on “OK”. Then, we select the “peak\_detection” and the original spectrum in the “Signals” panel on the right, so that both are displayed in the visualization panel on the left.

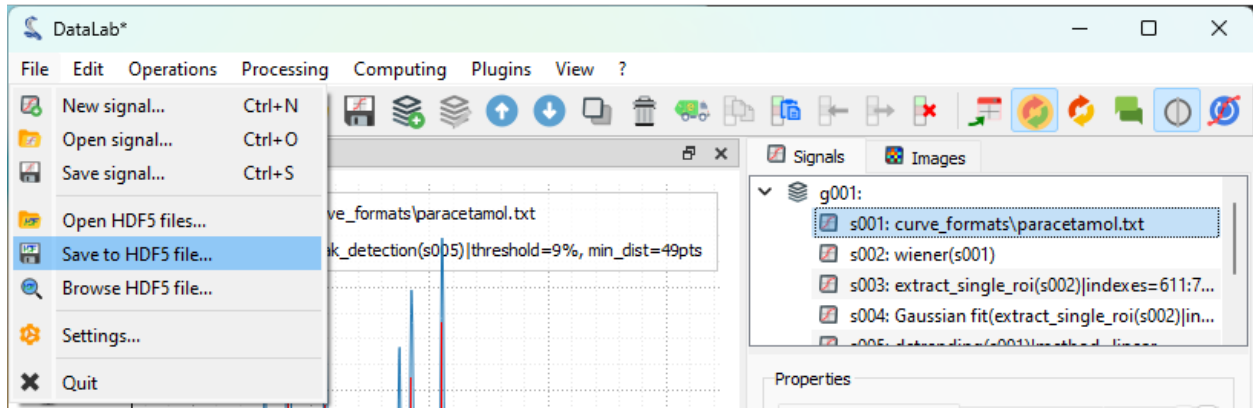


Fig. 25: Save the workspace to a file with “File > Save to HDF5 file...”, or the button in the toolbar.

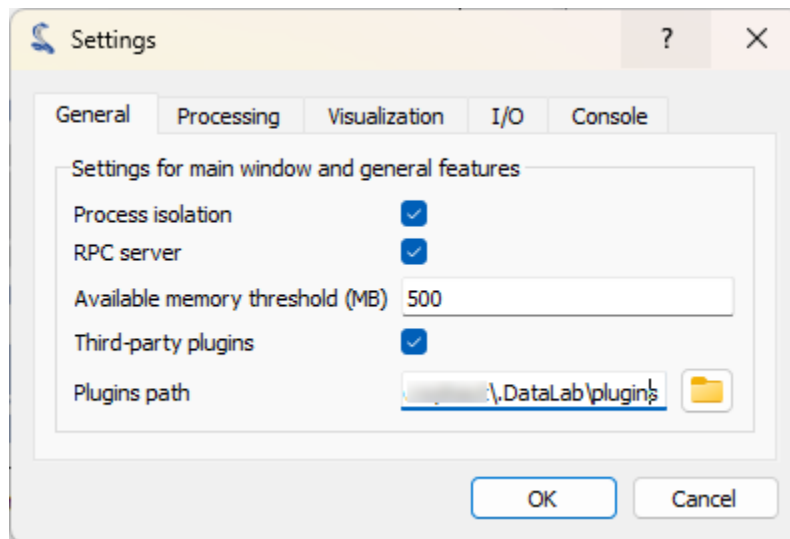



Fig. 26: In the “General” tab, we can see the “Plugins path” field. This is the path where DataLab will look for plugins. We can add a new plugin by copying/pasting the plugin file in this directory.

First, we open DataLab, and open the settings dialog (using “File > Settings...”, or the  icon in the toolbar).

**See also:**

The plugin system is described in the [Plugins](#) section.

Let’s add the `cdl_example_imageproc.py` plugin to DataLab (this is an example plugin that is shipped with DataLab source package, or may be downloaded from [here on GitHub](#)).

If we close and reopen DataLab, we can see that the plugin is now available in the “Plugins” menu: there is a new “Extract blobs (example)” entry.

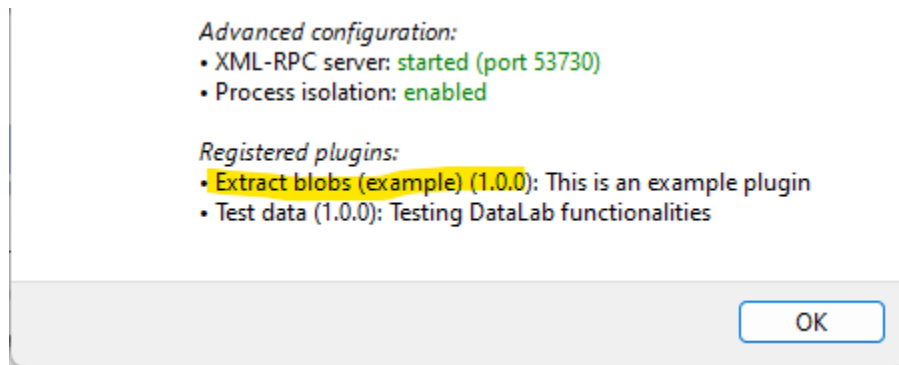


Fig. 27: The “About DataLab” dialog shows the list of available plugins.

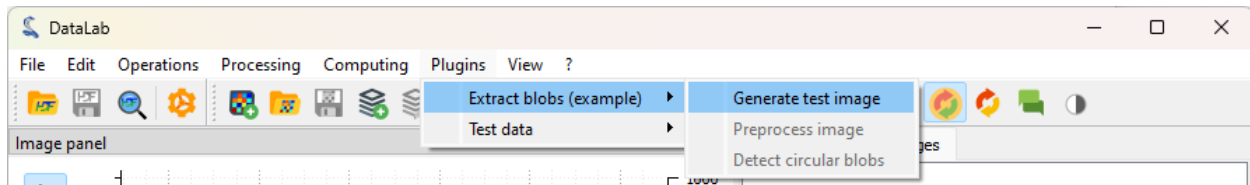


Fig. 28: Let’s click on “Extract blobs (example) > Generate test image”

For information, the image is generated by the plugin using the following code:

```
def generate_test_image(self) -> None:
    """Generate test image"""
    # Create a NumPy array:
    arr = np.random.normal(10000, 1000, (2048, 2048))
    for _ in range(10):
        row = np.random.randint(0, arr.shape[0])
        col = np.random.randint(0, arr.shape[1])
        rr, cc = skimage.draw.disk((row, col), 40, shape=arr.shape)
        arr[rr, cc] -= np.random.randint(5000, 6000)
    icenter = arr.shape[0] // 2
    rr, cc = skimage.draw.disk((icenter, icenter), 200, shape=arr.shape)
    arr[rr, cc] -= np.random.randint(5000, 8000)
    data = np.clip(arr, 0, 65535).astype(np.uint16)

    # Create a new image object and add it to the image panel
    image = cdl.obj.create_image("Test image", data, units=("mm", "mm", "lsb"))
    self.proxy.add_object(image)
```

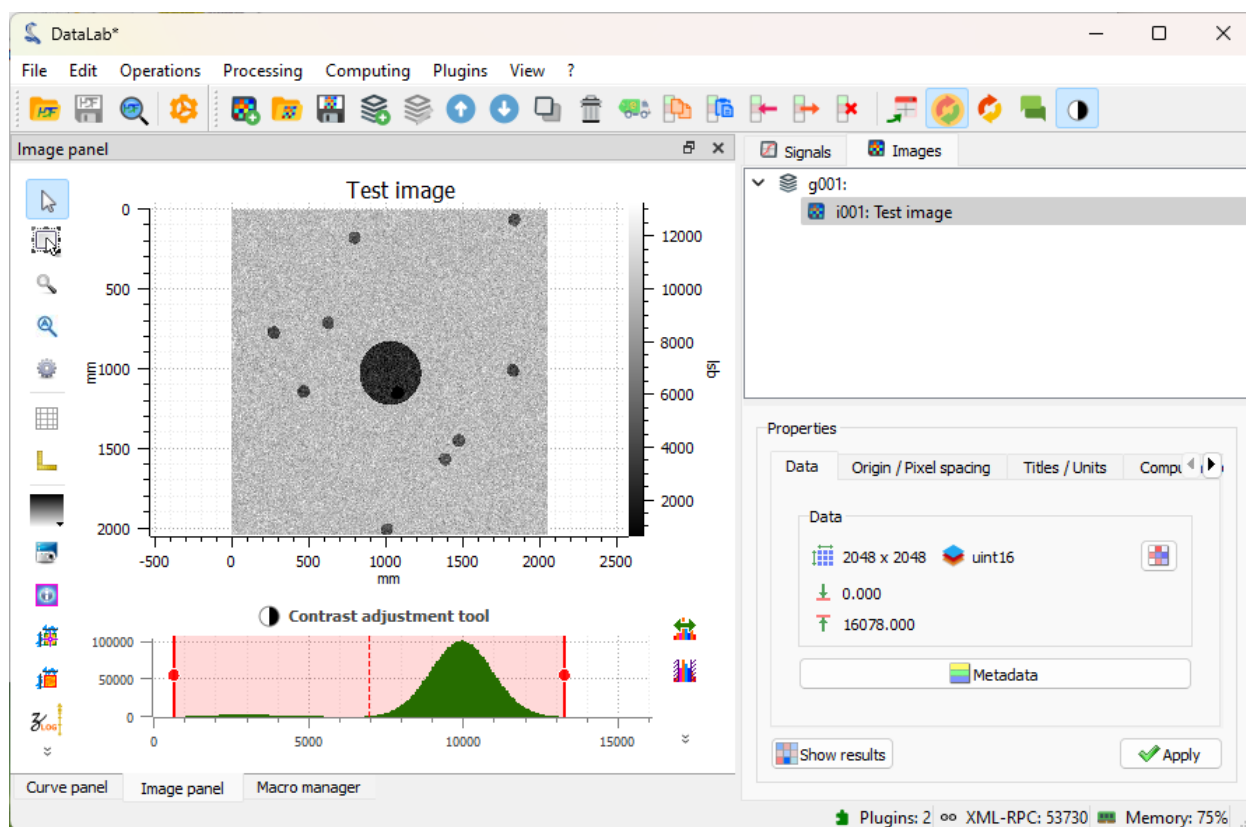


Fig. 29: The plugin has generated a test image, and added it to the “Images” panel. The image shows a few blobs, with a central dark disk, and a noisy background.



The plugin has other features, such as denoising the image, and detecting blobs on the image, but we won't cover them here: we will use the same DataLab native features as the plugin, manually.

The image is a bit noisy, and also quite large. Let's reduce the size of the image while denoising it a bit by binning the image by a factor of 2.

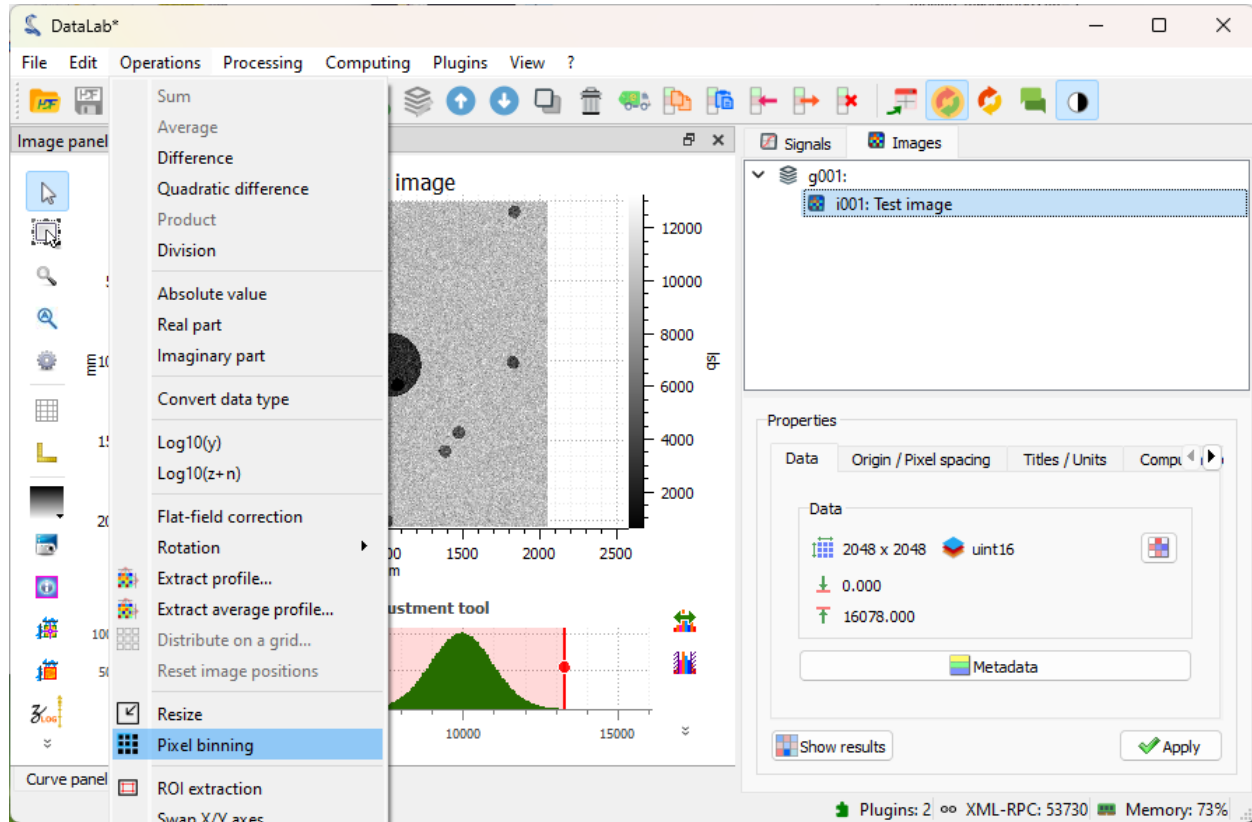
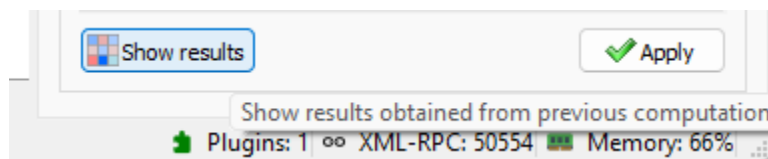


Fig. 30: Click on “Operations > Pixel binning”.

Let's apply a moving median filter to the image, to denoise it a bit more.

Now, let's detect the blobs on the image.

**Note:** If you want to show the analysis results again, you can select the “Show results” entry in the “Analysis” menu, or the “Show results” button, below the image list:



Finally, we can save the workspace to a file. The workspace contains all the images that were loaded in DataLab, as well as the processing results. It also contains the visualization settings (colormaps, contrast, etc.), the metadata, and the annotations. To save the workspace, click on “File > Save to HDF5 file...”, or the button in the toolbar.

If you want to load the workspace again, you can use the “File > Open HDF5 file...” (or the button in the toolbar) to load the whole workspace, or the “File > Browse HDF5 file...” (or the button in the toolbar) to load only a selection of data sets from the workspace.

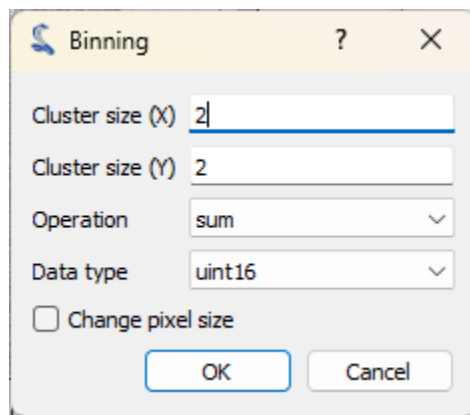


Fig. 31: The “Binning” dialog opens. Set the binning factor to 2, and click on “OK”.

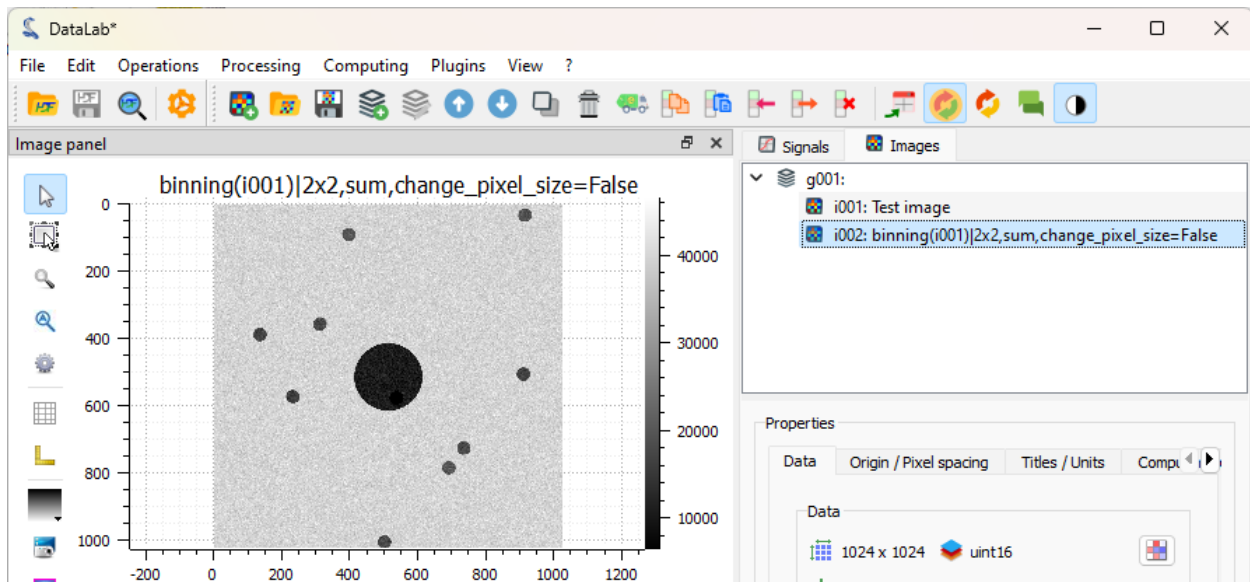


Fig. 32: The binned image is added to the “Images” panel. It is now easier to see the blobs (even if they were already quite visible on the original image: this is just an example), and the image will be faster to process.

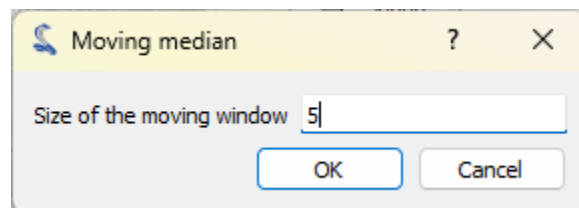


Fig. 33: Click on “Processing > Moving median” entry, and set the window size to 5.

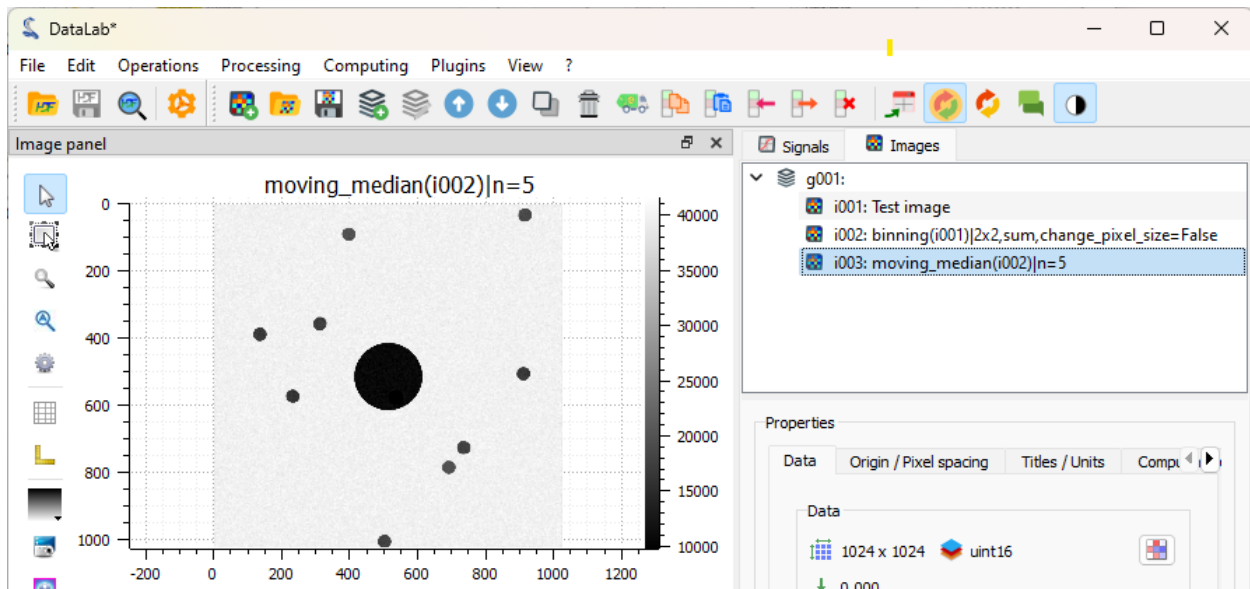


Fig. 34: The filtered image is added to the “Images” panel. Denoising is quite efficient.

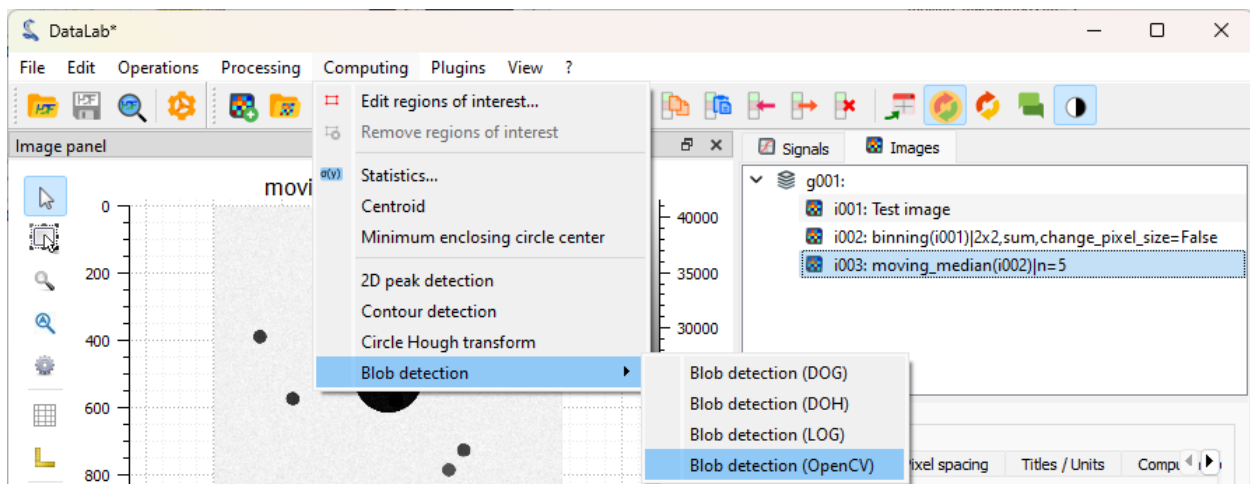


Fig. 35: Click on “Analysis > Blob detection > Blob detection (OpenCV)”.

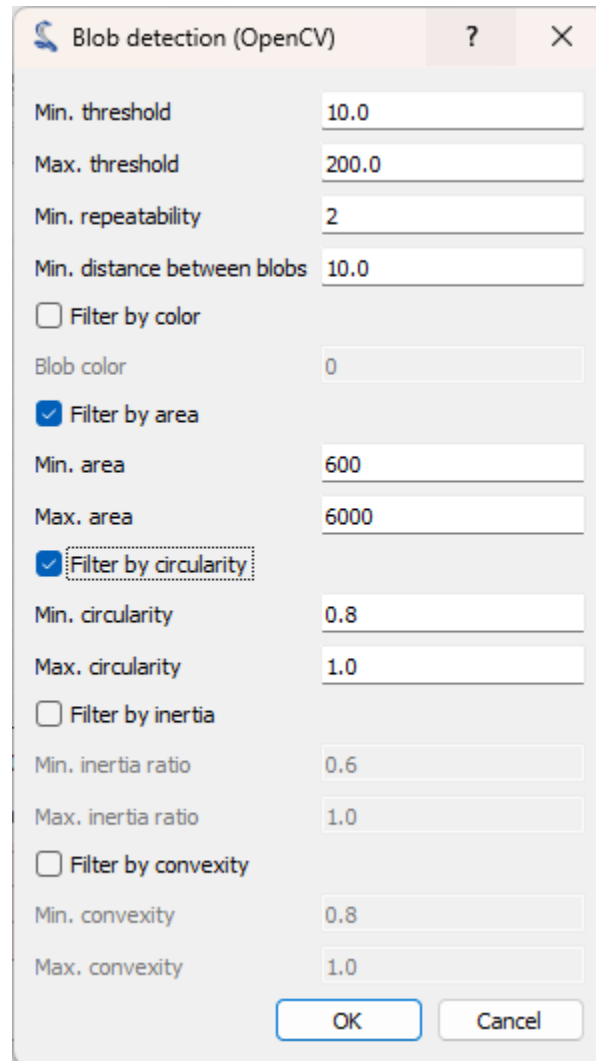
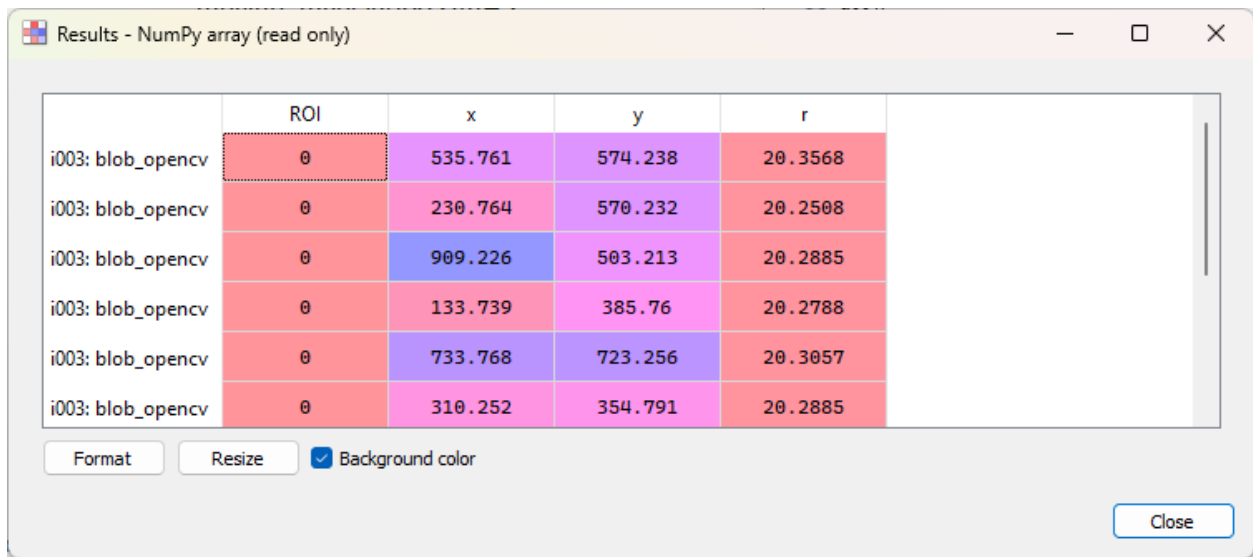


Fig. 36: The “Blob detection (OpenCV)” dialog opens. Set the parameters as shown on the screenshot, and click on “OK”.



The "Results" dialog window displays a table of detected blobs. The table has five columns: ROI, x, y, and r. The first column lists the operation used for each blob. The ROI column contains the value 0 for all blobs. The x, y, and r columns contain the coordinates and radius for each blob. The background color of the table is set to blue.

	ROI	x	y	r
i003: blob_opencv	0	535.761	574.238	20.3568
i003: blob_opencv	0	230.764	570.232	20.2508
i003: blob_opencv	0	909.226	503.213	20.2885
i003: blob_opencv	0	133.739	385.76	20.2788
i003: blob_opencv	0	733.768	723.256	20.3057
i003: blob_opencv	0	310.252	354.791	20.2885

Buttons: Format, Resize, ☒ Background color, Close

Fig. 37: The “Results” dialog opens, showing the detected blobs: one line per blob, with the blob coordinates and radius.

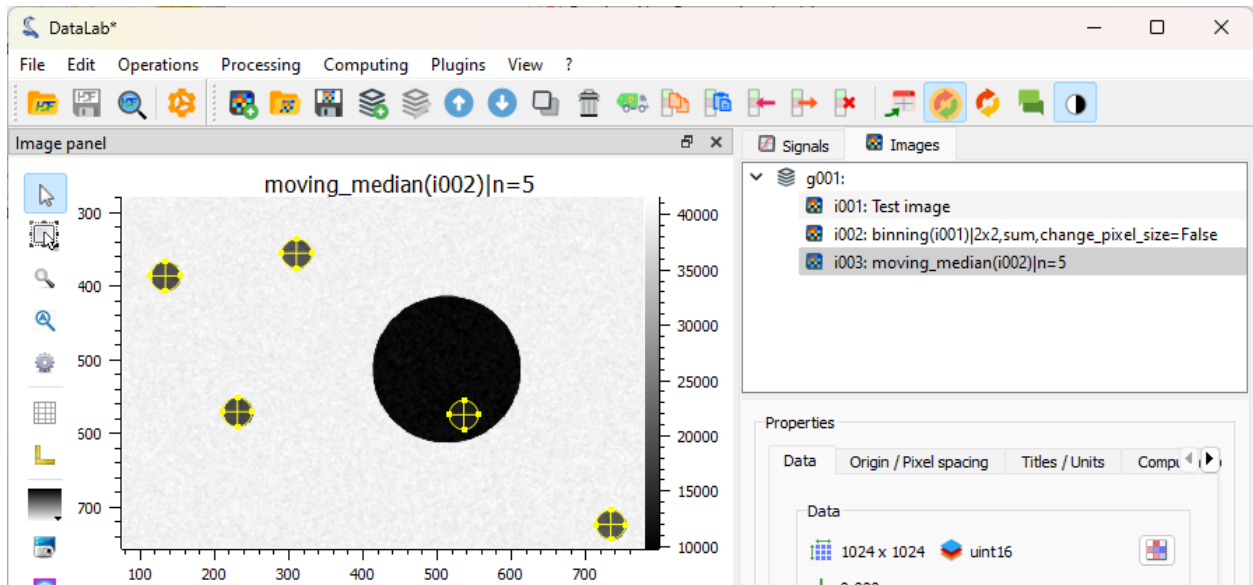


Fig. 38: The detected blobs are also added to the image metadata, and can be seen in the visualization panel on the left.

## Measuring Fabry-Perot fringes

This example shows how to measure Fabry-Perot fringes using the image processing features of DataLab:

- Load an image of a Fabry-Perot interferometer
- Define a circular region of interest (ROI) around the central fringe
- Detect contours in the ROI and fit them to circles
- Show the radius of the circles
- Annotate the image
- Copy/paste the ROI to another image
- Extract the intensity profile along the X axis
- Save the workspace

First, we open DataLab and load the images:

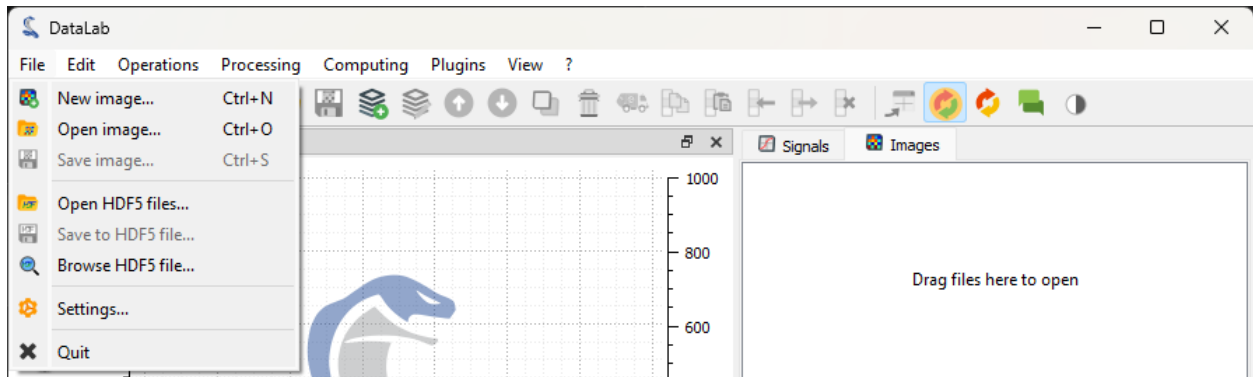


Fig. 39: Open the image files with “File > Open...”, or with the button in the toolbar, or by dragging and dropping the files into DataLab (on the panel on the right).

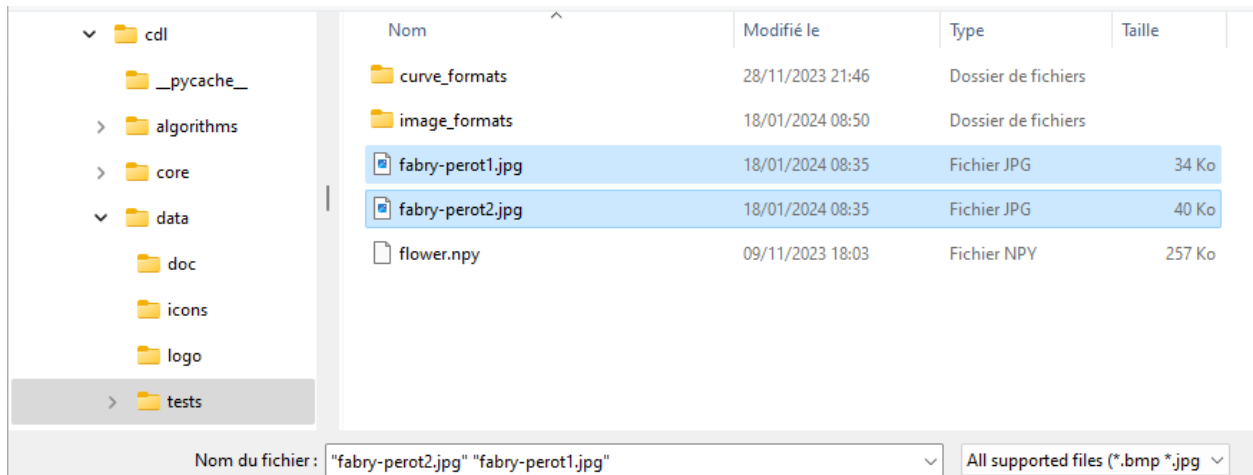


Fig. 40: Select the test images “fabry\_perot1.jpg” and “fabry\_perot2.jpg” and click “Open”.

The selected image is displayed in the main window. We can zoom in and out by pressing the right mouse button and dragging the mouse up and down. We can also pan the image by pressing the middle mouse button and dragging the mouse.

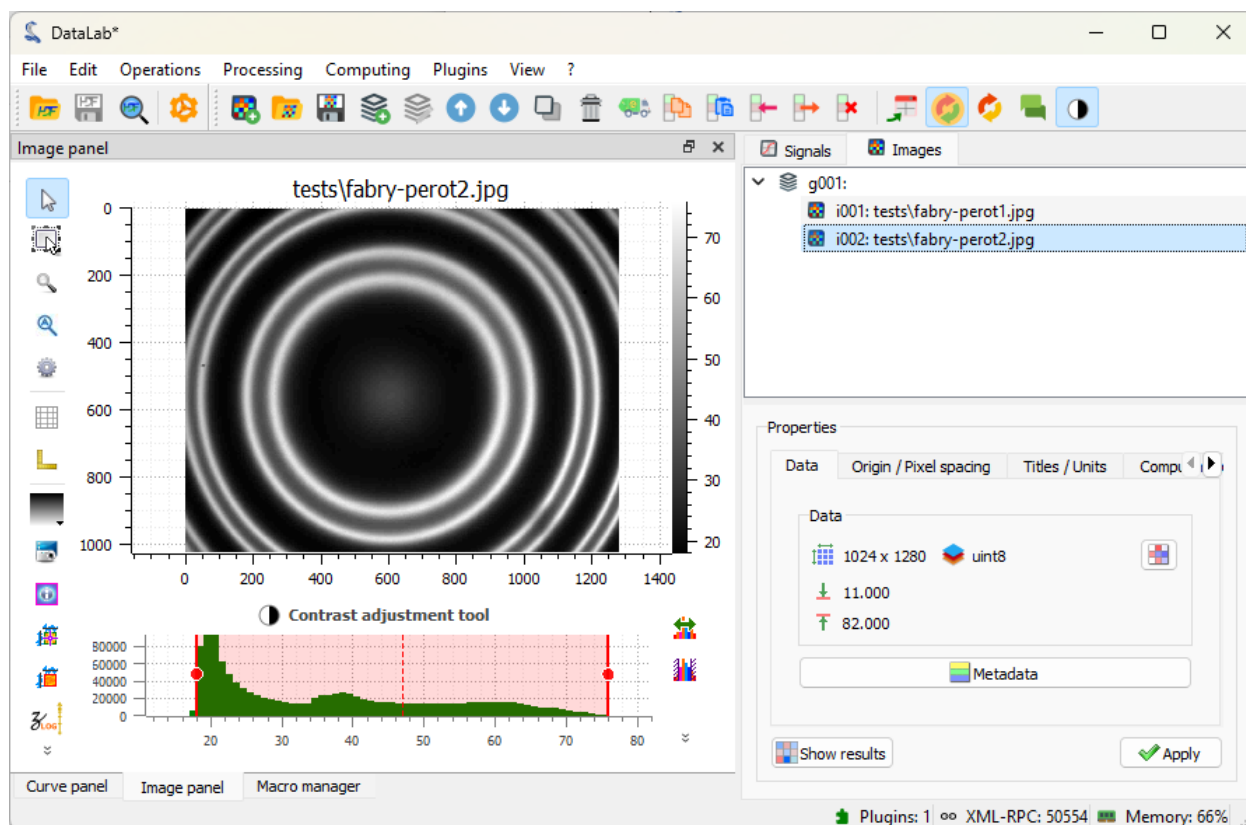



Fig. 41: Zoom in and out with the right mouse button. Pan the image with the middle mouse button.

**Note:** When working on application-specific images (e.g. X-ray radiography images, or optical microscopy images), it is often useful to change the colormap to a grayscale colormap. If you see a different image colormap than the one shown in the figure, you can change it by selecting the image in the visualization panel, and the selecting the colormap in the vertical toolbar on the left of the visualization panel.

Or, even better, you can change the default colormap in the DataLab settings by selecting “Edit > Settings...” in the menu, or the  button in the toolbar.

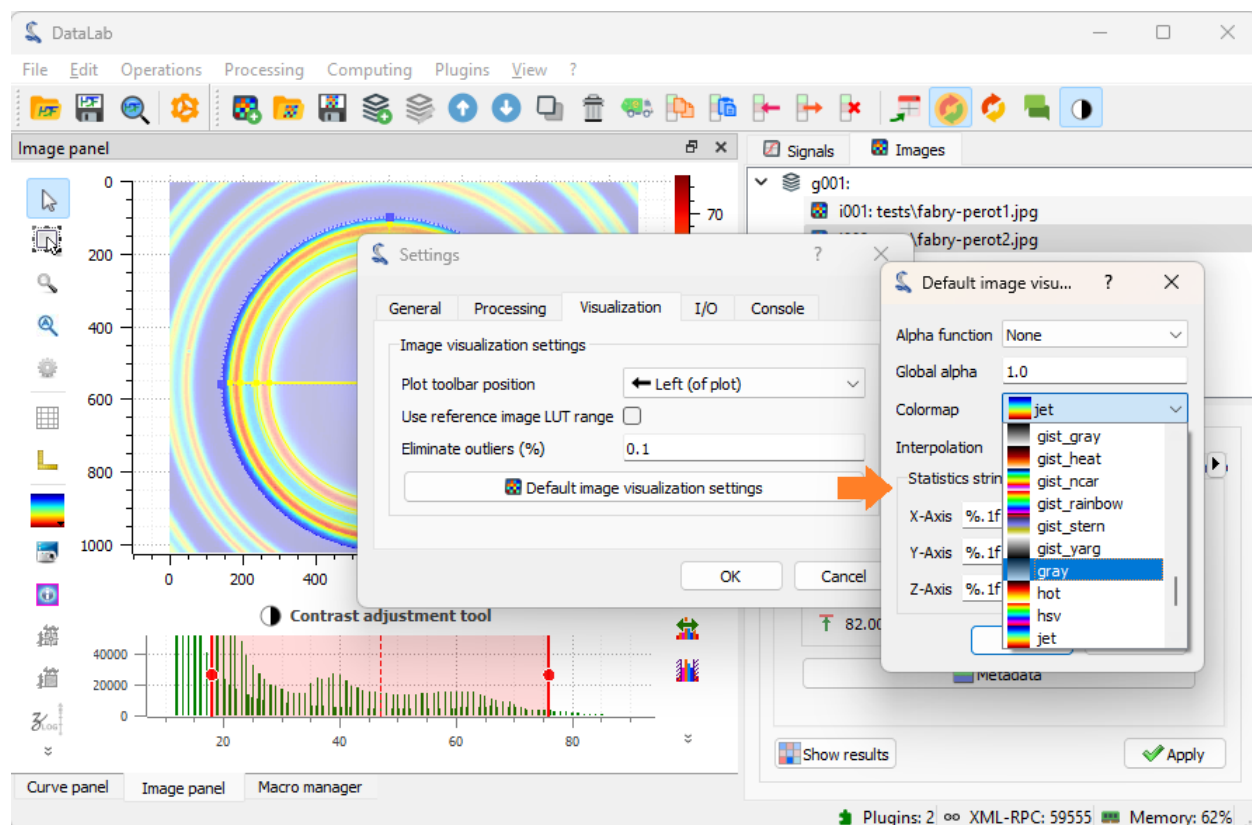
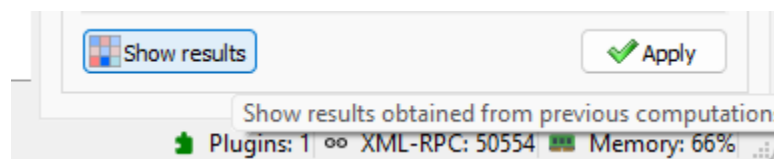



Fig. 42: Select the “Visualization” tab, and select the “gray” colormap.

Then, let’s define a circular region of interest (ROI) around the central fringe.

Now, let’s detect the contours in the ROI and fit them to circles.

**Note:** If you want to show the analysis results again, you can select the “Show results” entry in the “Analysis” menu, or the “Show results” button, below the image list:



The images (or signals) can also be displayed in a separate window, by clicking on the “View in a new window” entry in the “View” menu (or the  button in the toolbar). This is useful to compare side by side images or signals.



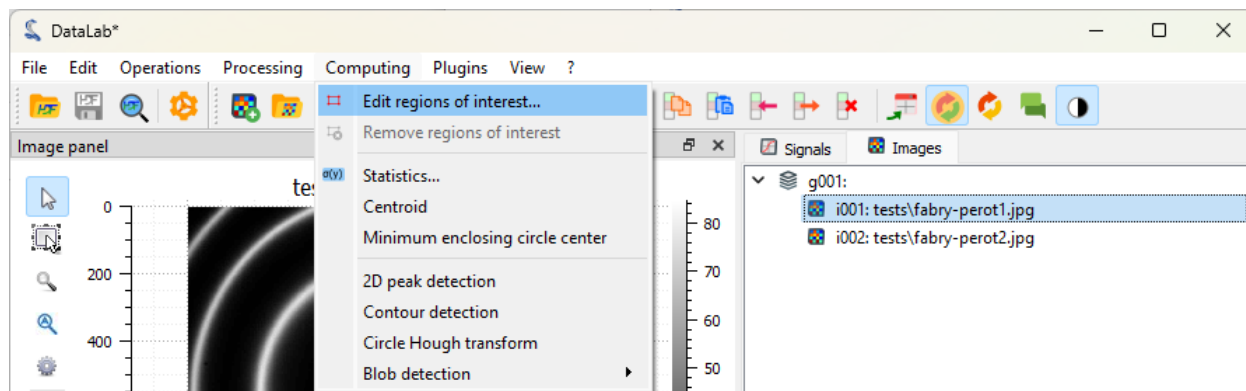


Fig. 43: Select the “Edit regions of interest” tool in the “Analysis” menu.

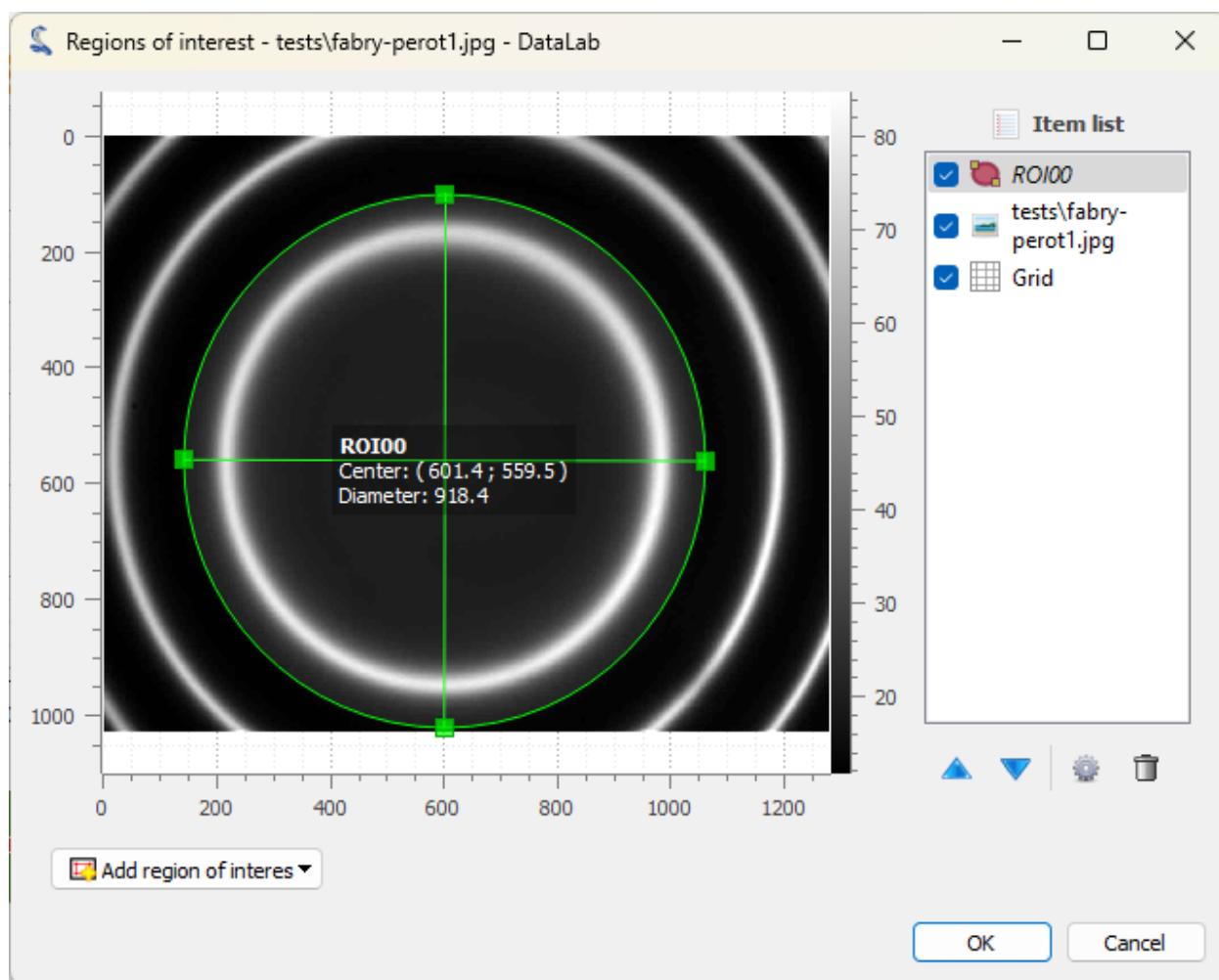


Fig. 44: The “Regions of interest” dialog opens. Click “Add ROI” and select a circular ROI. Resize the predefined ROI by dragging the handles. Note that you may change the ROI radius while keeping its center fixed by pressing the “Ctrl” key. Click “OK” to close the dialog.

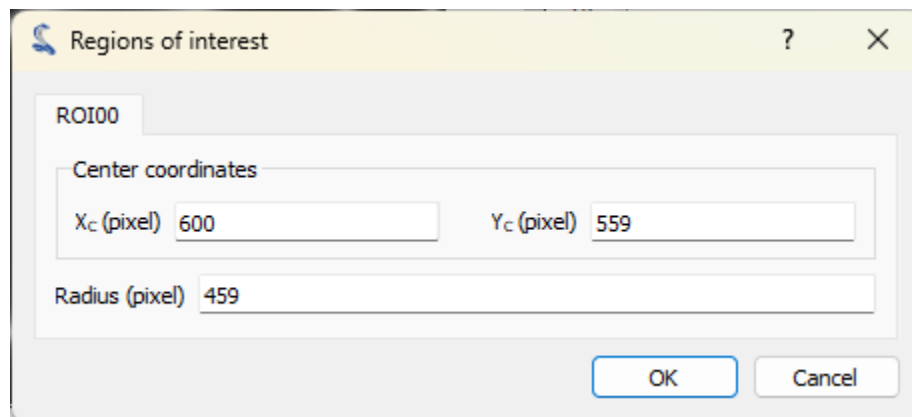


Fig. 45: Another dialog box opens, and asks you to confirm the ROI parameters. Click “OK”.

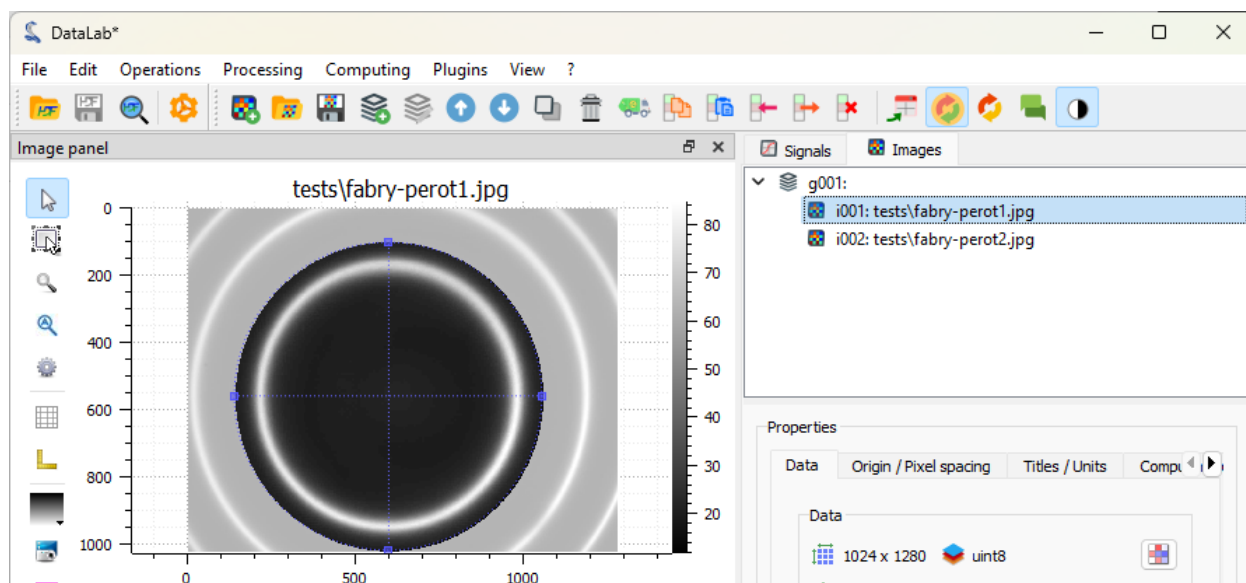


Fig. 46: The ROI is displayed on the image: masked pixels are grayed out, and the ROI boundary is displayed in blue (note that, internally, the ROI is defined by a binary mask, i.e. image data is represented as a NumPy masked array).

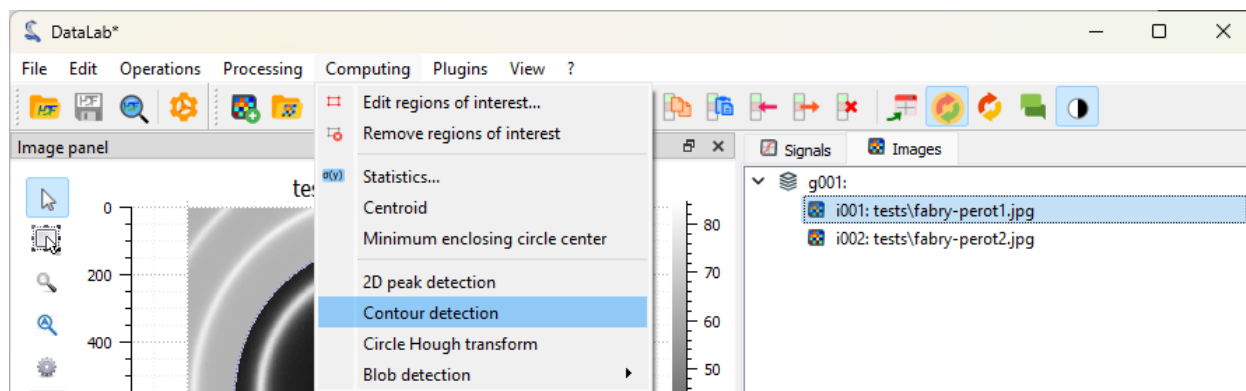


Fig. 47: Select the “Contour detection” tool in the “Analysis” menu.

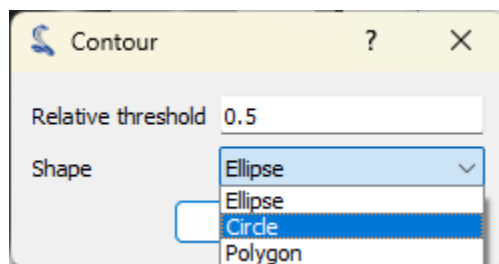


Fig. 48: The “Contour” parameters dialog opens. Select the shape “Circle” and click “OK”.

	ROI	x	y	r
contour_shape(i001)	0	599.144	554.736	403.22
contour_shape(i001)	0	595.39	556.167	367.349

Fig. 49: The “Results” dialog opens, and displays the fitted circle parameters. Click “OK”.

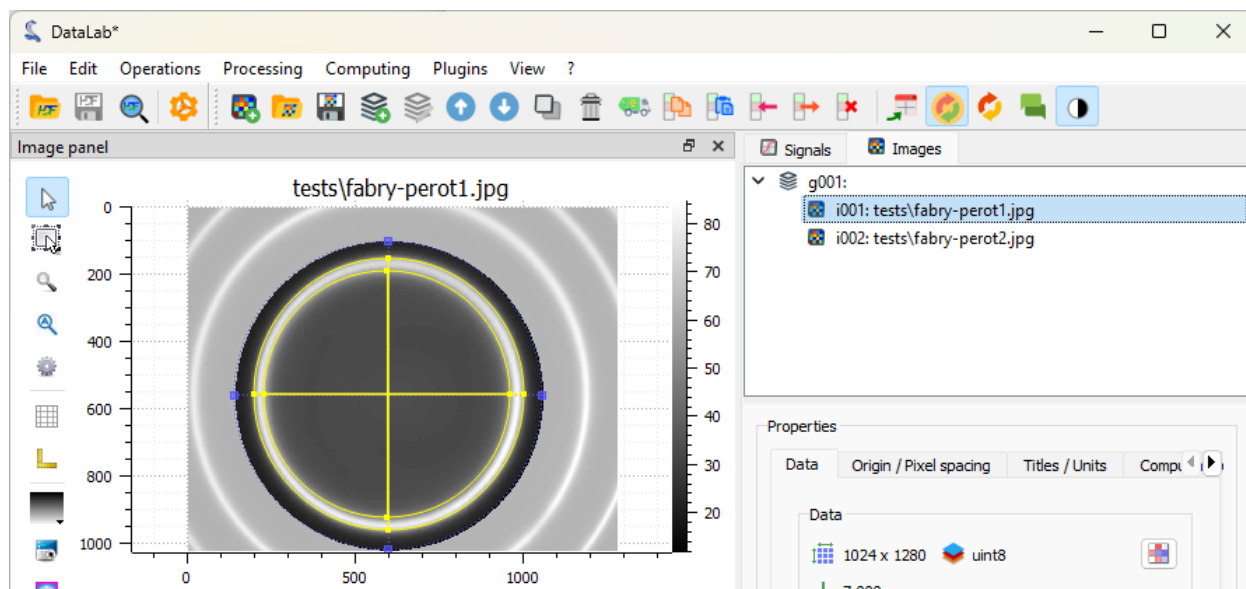


Fig. 50: The fitted circles are displayed on the image.

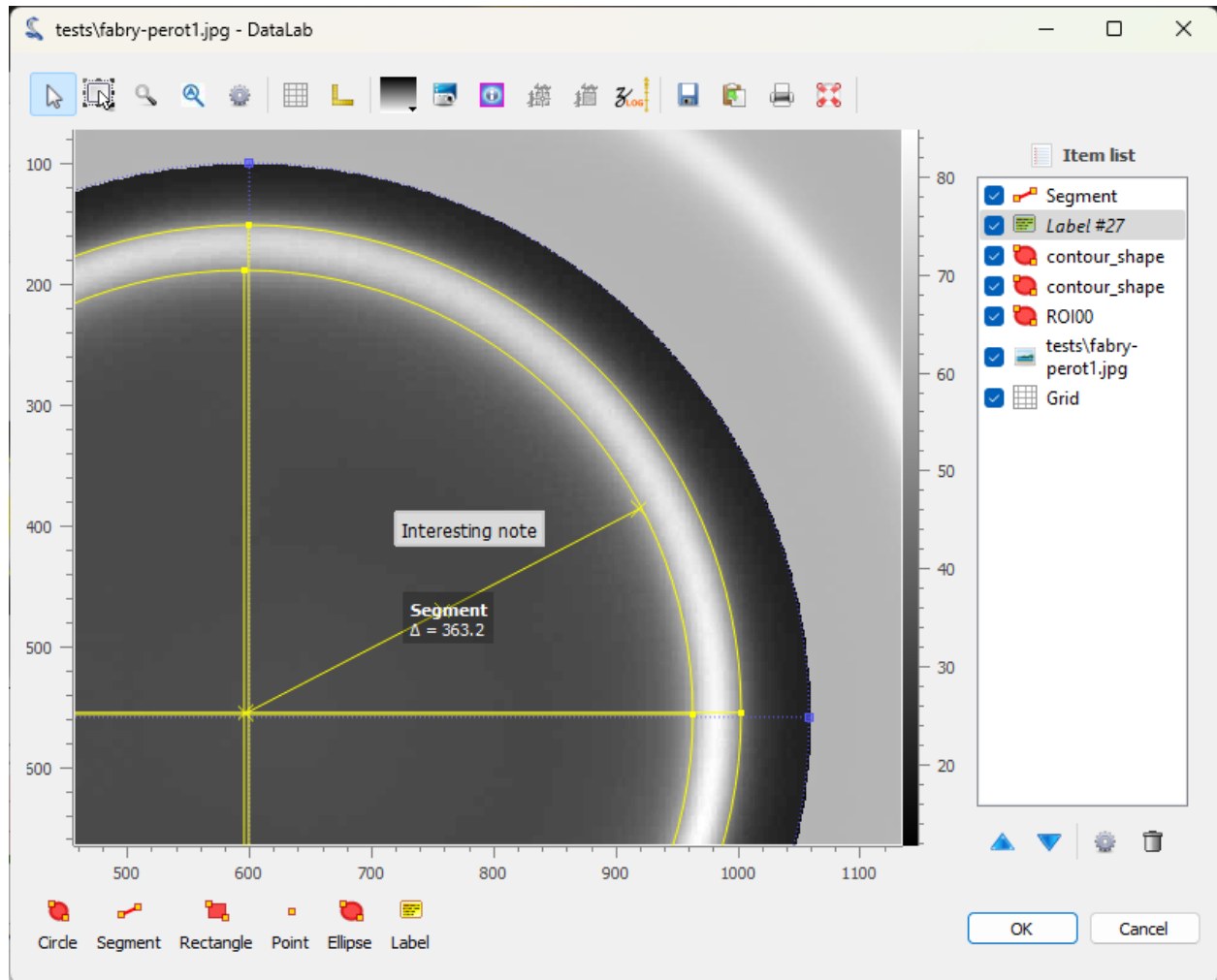


Fig. 51: The image is displayed in a separate window. The ROI and the fitted circles are also displayed. Annotations can be added to the image by clicking on the buttons at the bottom of the window. The annotations are stored in the metadata of the image, and together with the image data when the workspace is saved. Click on “OK” to close the window.

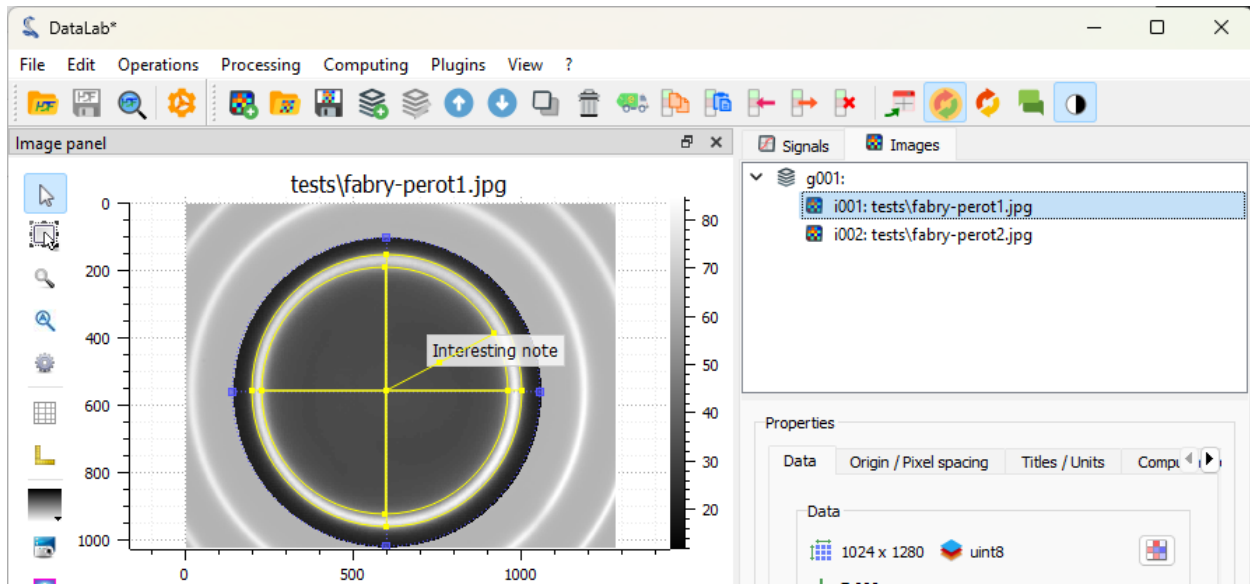


Fig. 52: The image is displayed in the main window, together with the annotations.

If you want to take a closer look at the metadata, you can open the “Metadata” dialog.

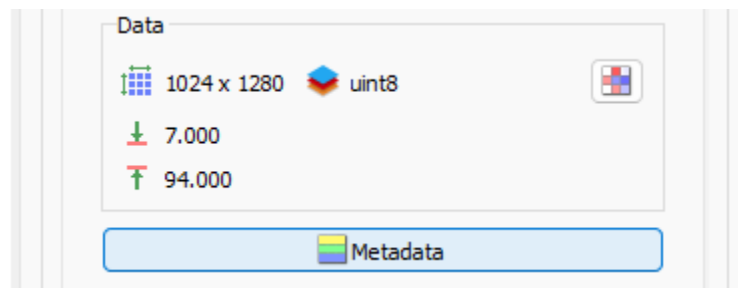



Fig. 53: The “Metadata” button is located below the image list.


Now, let’s delete the image metadata (including the annotations) to clean up the image.


If we want to define the exact same ROI on the second image, we can copy/paste the ROI from the first image to the second image, using the metadata.

To extract the intensity profile along the X axis, we have two options:

- Either select the “Line profile...” entry in the “Operations > Intensity profiles” menu.
- Or activate the “Cross section” tool  in the vertical toolbar on the left of the visualization panel.

Let’s try the first option, by selecting the “Line profile...” entry : that is the most straightforward way to extract a profile from an image, and it corresponds to the `compute_profile` method of DataLab’s API (so it can be used in a script, a plugin or a macro).

If you want to do some measurements on the profile, or add annotations, you can open the signal in a separate window, by clicking on the “View in a new window” entry in the “View” menu (or the  button in the toolbar).

Now, let’s try the second option for extracting the intensity profile along the X axis, by activating the “Cross section” tool  in the vertical toolbar on the left of the visualization panel (this tool is a `PlotPy` feature). Before being able to use it, we need to select the image in the visualization panel (otherwise the tool is grayed out). Then, we can click

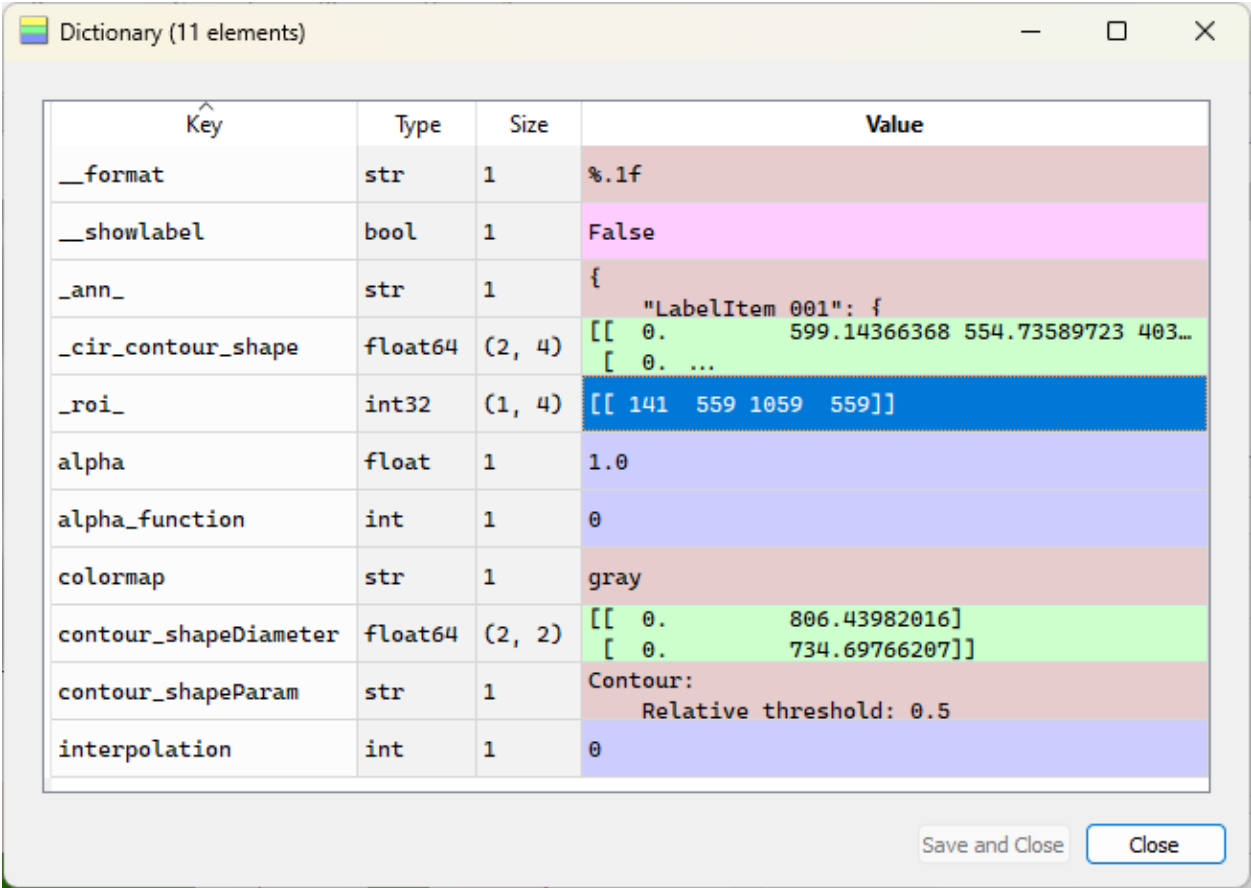


Fig. 54: The “Metadata” dialog opens. Among other information, it displays the annotations (in a JSON format), some style information (e.g. the colormap), and the ROI.

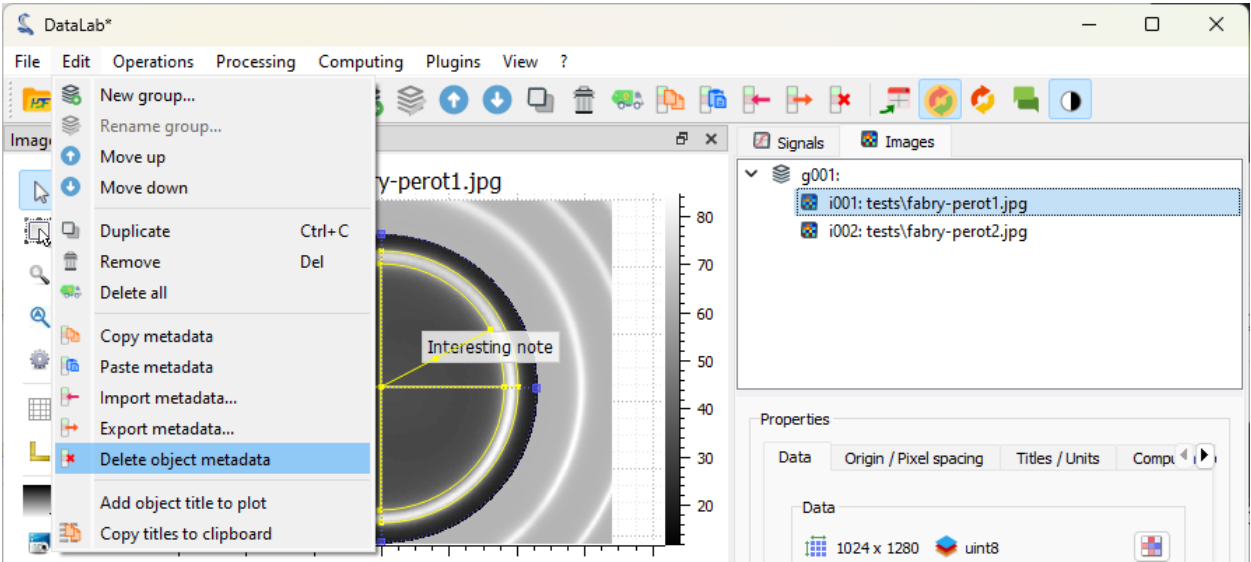


Fig. 55: Select the “Delete metadata” entry in the “Edit” menu, or the button in the toolbar.

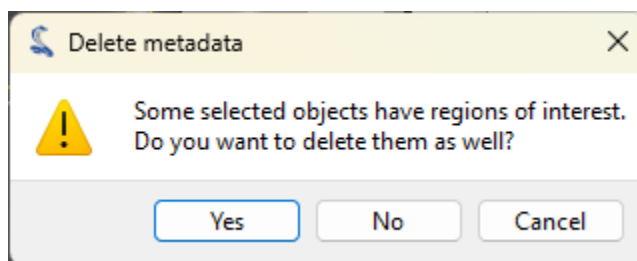


Fig. 56: The “Delete metadata” dialog opens. Click “No” to keep the ROI and delete the rest of the metadata.

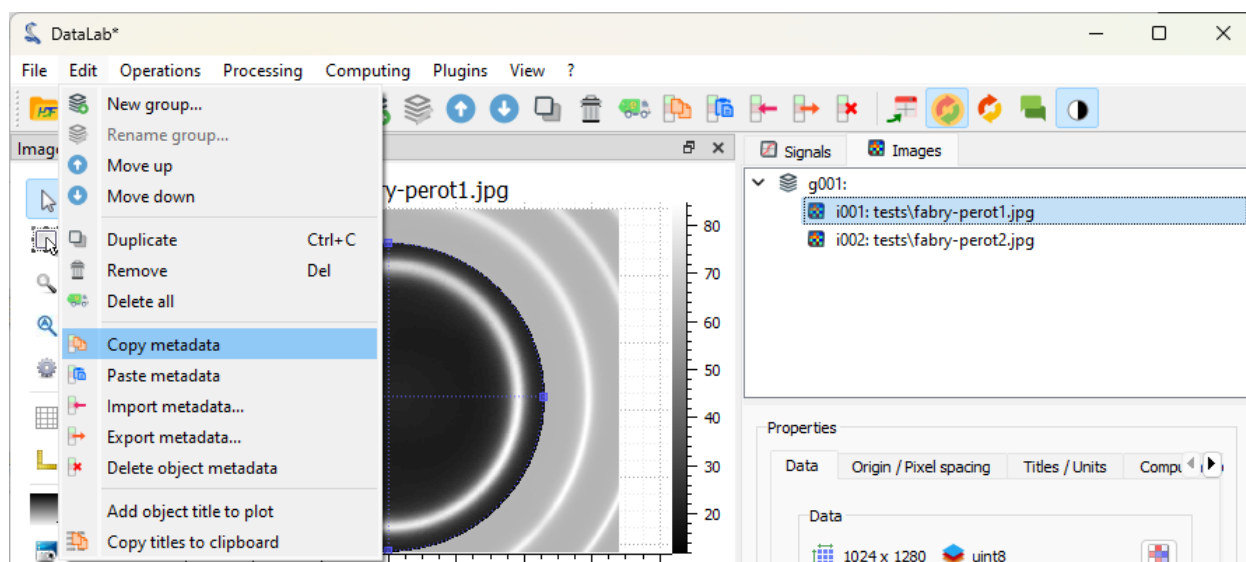


Fig. 57: Select the “Copy metadata” entry in the “Edit” menu, or the button in the toolbar.

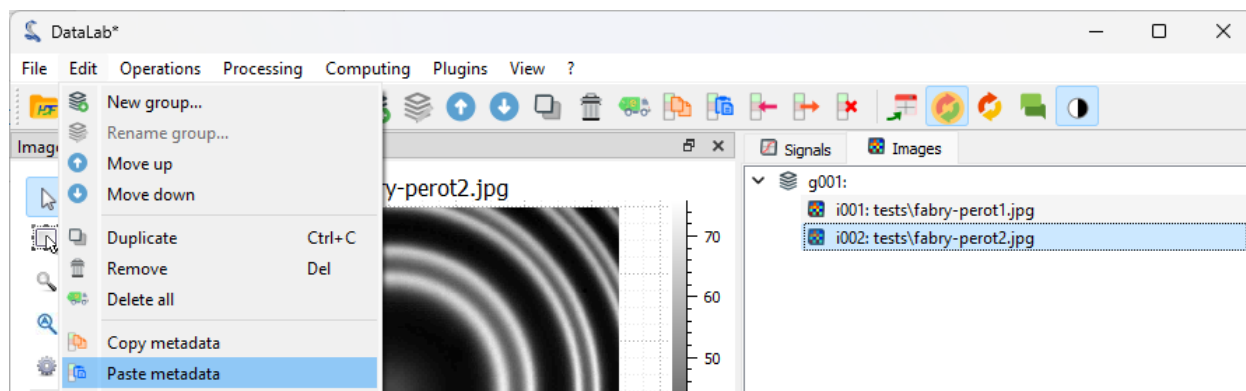


Fig. 58: Select the second image in the “Images” panel, then select the “Paste metadata” entry in the “Edit” menu, or the button in the toolbar.

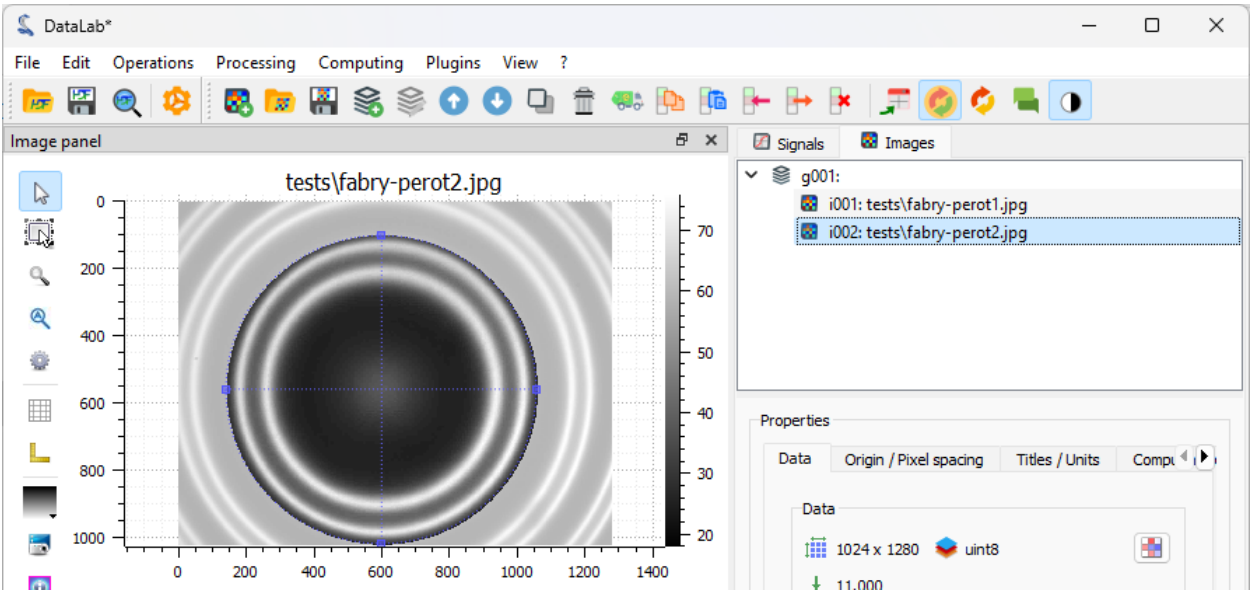


Fig. 59: The ROI is added to the second image.

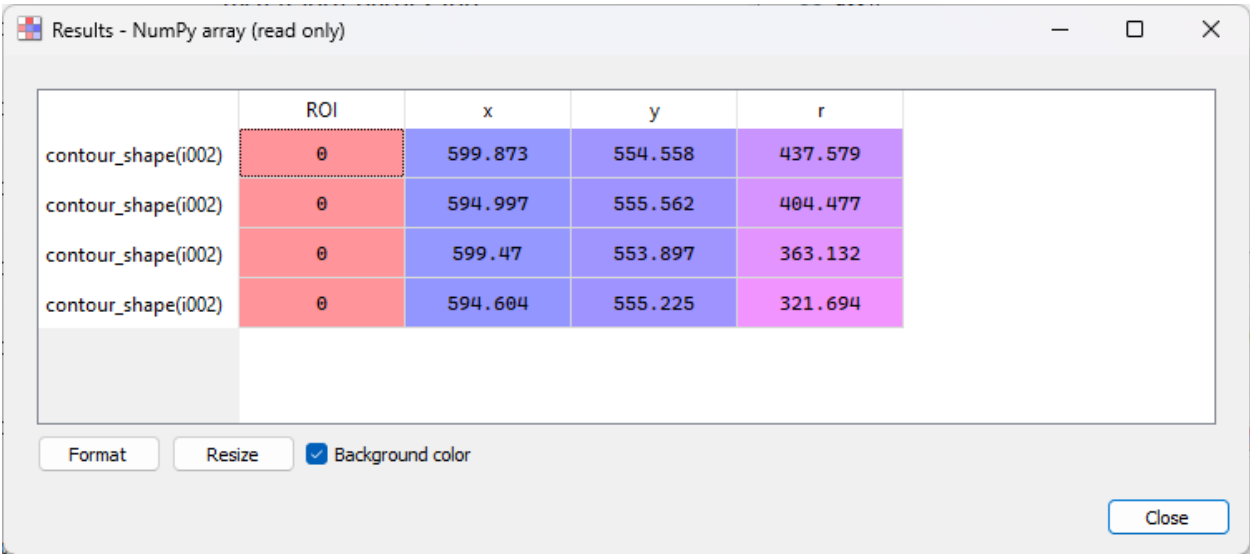


Fig. 60: Select the “Contour detection” tool in the “Analysis” menu, with the same parameters as before (shape “Circle”). On this image, there are two fringes, so four circles are fitted. The “Results” dialog opens, and displays the fitted circle parameters. Click “OK”.



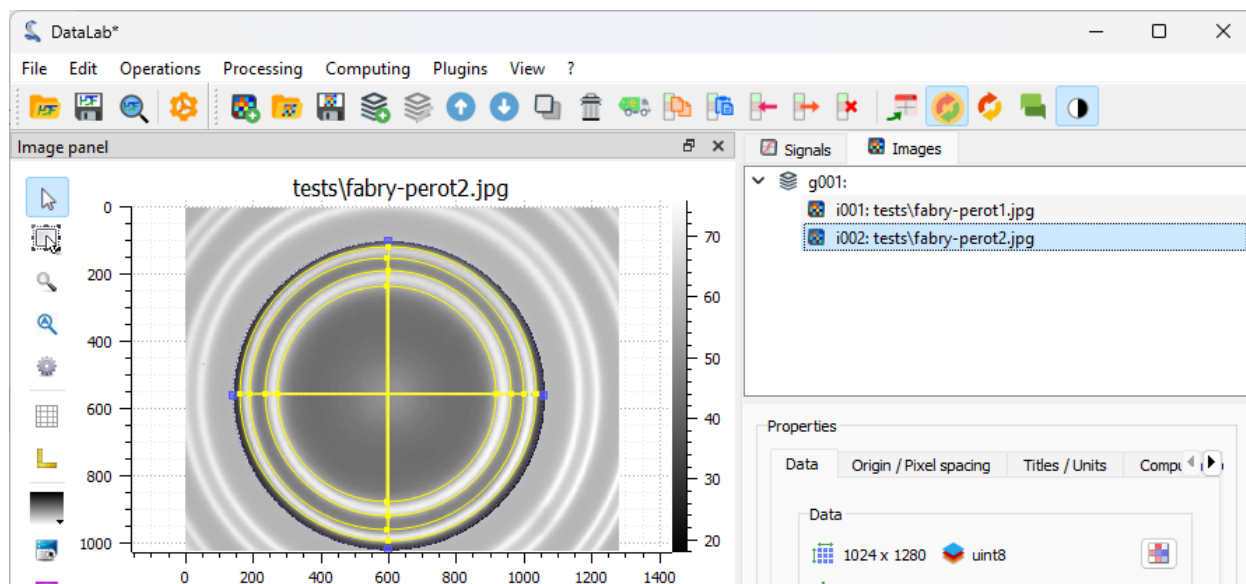


Fig. 61: The fitted circles are displayed on the image.

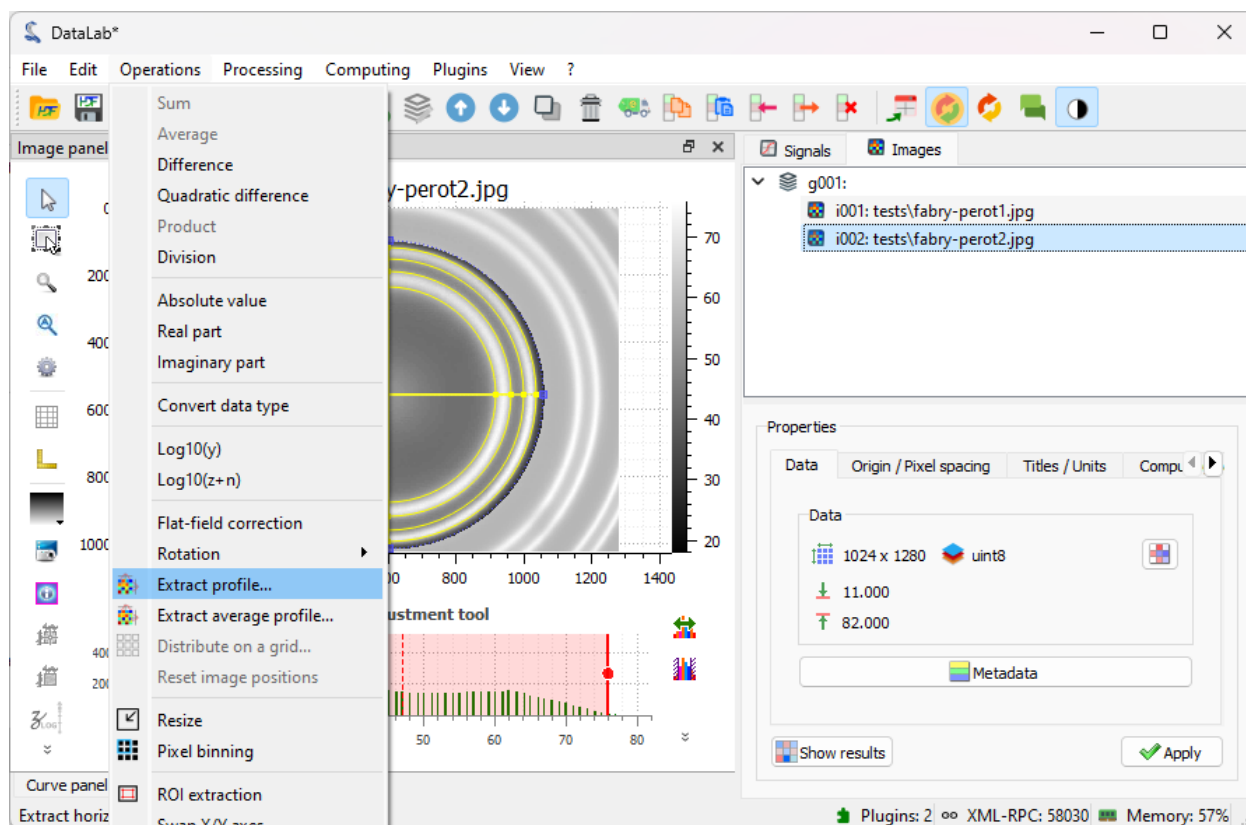


Fig. 62: Select the “Line profile...” entry in the “Operations” menu.

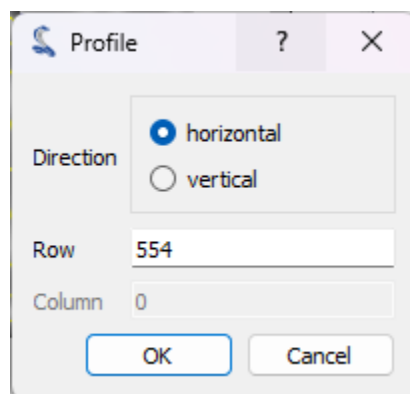


Fig. 63: The “Profile” dialog opens. Enter the row of the horizontal profile (or the column of the vertical profile) in the dialog box that opens. Click “OK”.

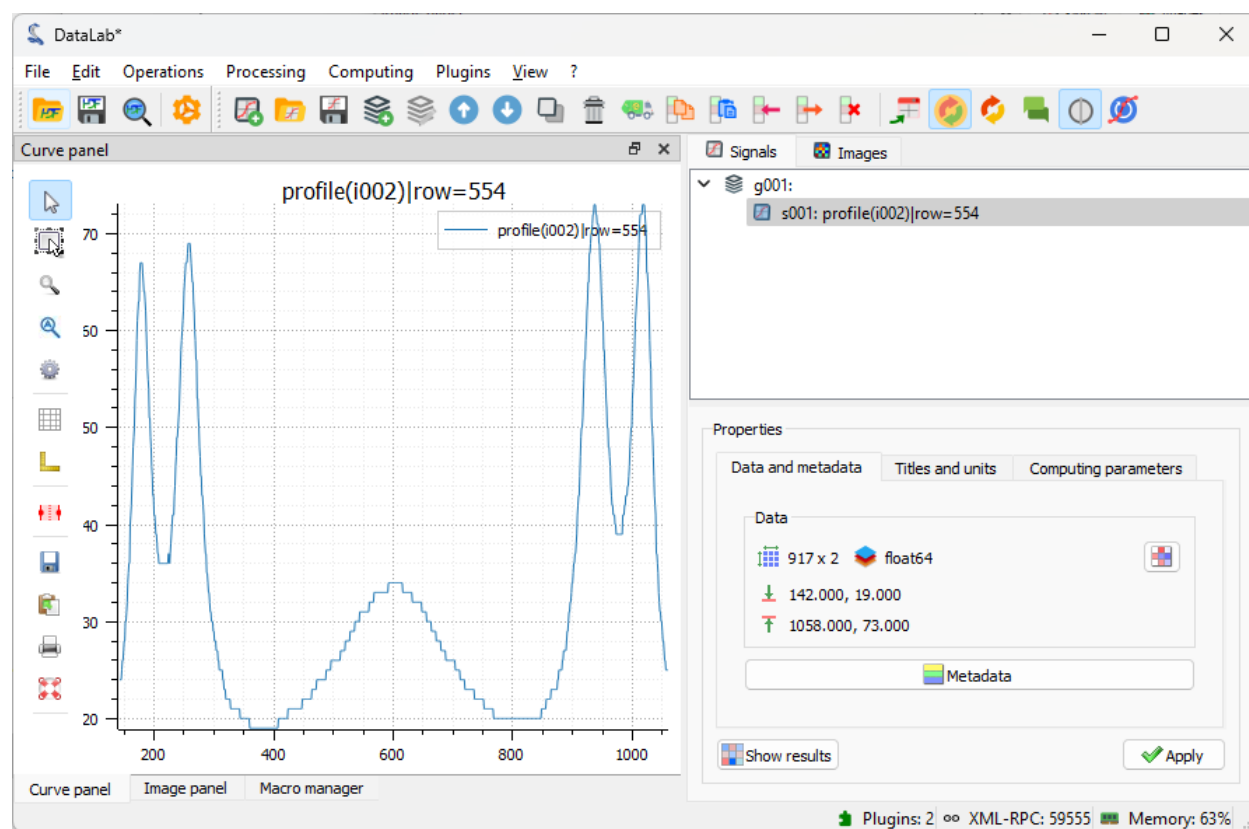


Fig. 64: The intensity profile is added to the “Signals” panel, and DataLab switches to this panel to display the profile.

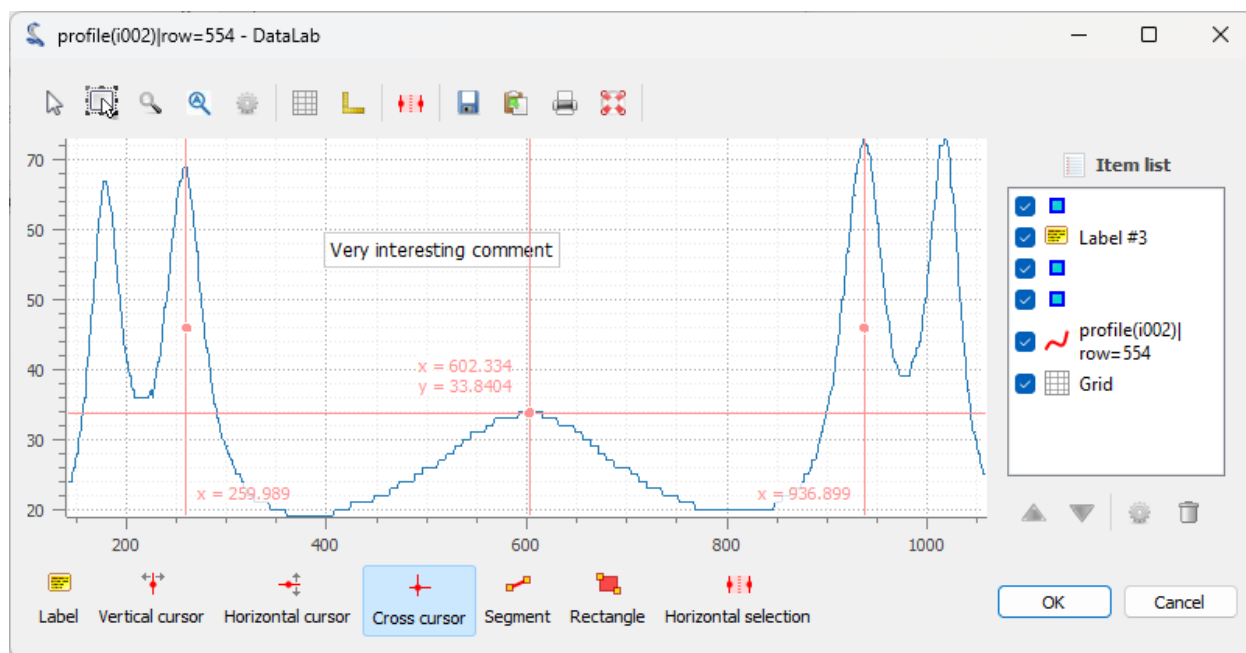


Fig. 65: The signal is displayed in a separate window. Here, we added vertical cursors and a very interesting text label. As for the images, the annotations are stored in the metadata of the signal, and together with the signal data when the workspace is saved. Click on “OK” to close the window.

on the image to display the intensity profile along the X axis. DataLab integrates a modified version of this tool, that allows to transfer the profile to the “Signals” panel for further processing.

Then, click on the “Process signal” button in the toolbar near the profile to transfer the profile to the “Signals” panel.

Finally, we can save the workspace to a file. The workspace contains all the images and signals that were loaded or processed in DataLab. It also contains the analysis results, the visualization settings (colormaps, contrast, etc.), the metadata, and the annotations.

If you want to load the workspace again, you can use the “File > Open HDF5 file...” (or the button in the toolbar) to load the whole workspace, or the “File > Browse HDF5 file...” (or the button in the toolbar) to load only a selection of data sets from the workspace.

## Measuring Laser Beam Size

This example shows how to measure the size of a laser beam along the propagation axis, using using DataLab:

- Load all the images in a folder
- Apply a threshold to the images
- Extract the intensity profile along an horizontal line
- Fit the intensity profile to a Gaussian function
- Compute the full width at half maximum (FWHM) of intensity profile
- Try another method: extract the radial intensity profile
- Compute the FWHM of the radial intensity profile
- Perform the same analysis on a stack of images and on the resulting profiles



Fig. 66: Switch back to the “Images” panel, and select the image *in the visualization panel* (otherwise the “Cross section” tool is grayed out). Select the “Cross section” tool in the vertical toolbar, and click on the image to display the intensity profiles along the X and Y axes.

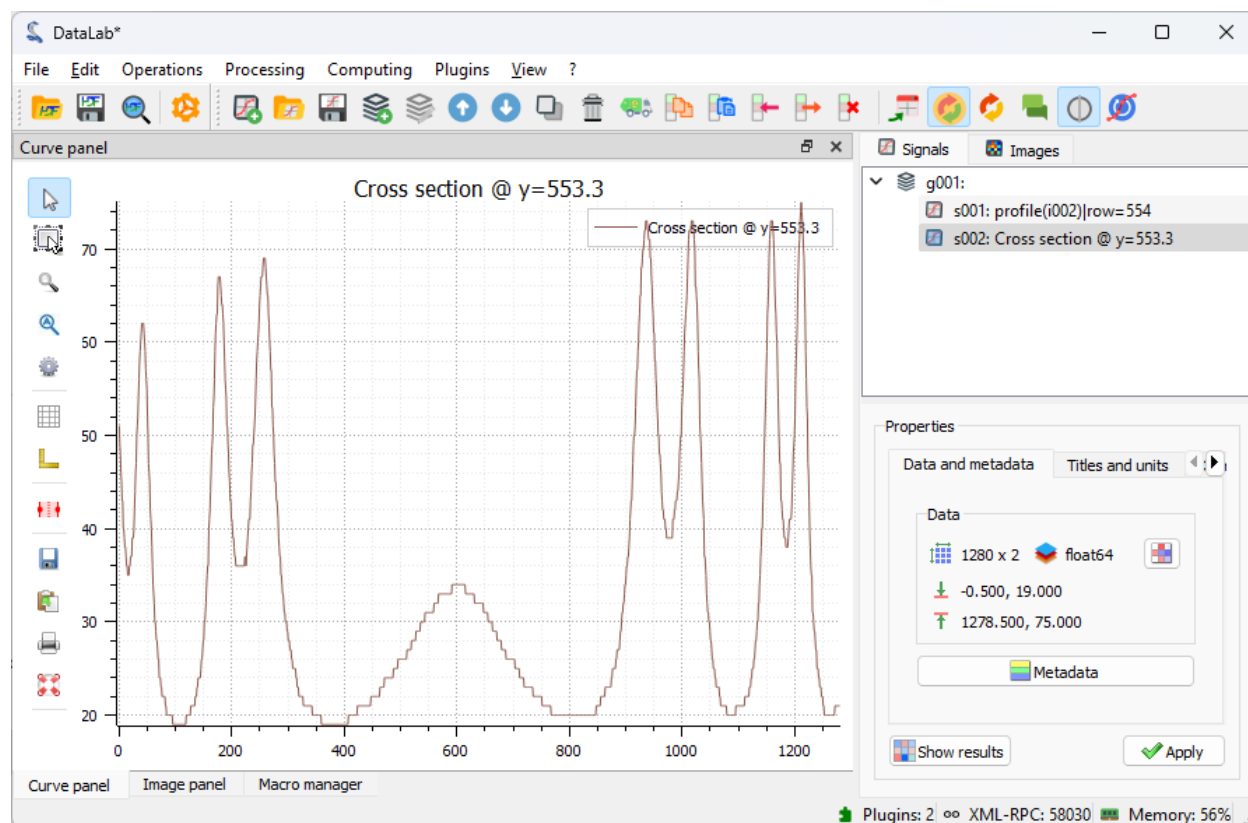


Fig. 67: The intensity profile is added to the “Signals” panel, and DataLab switches to this panel to display the profile.

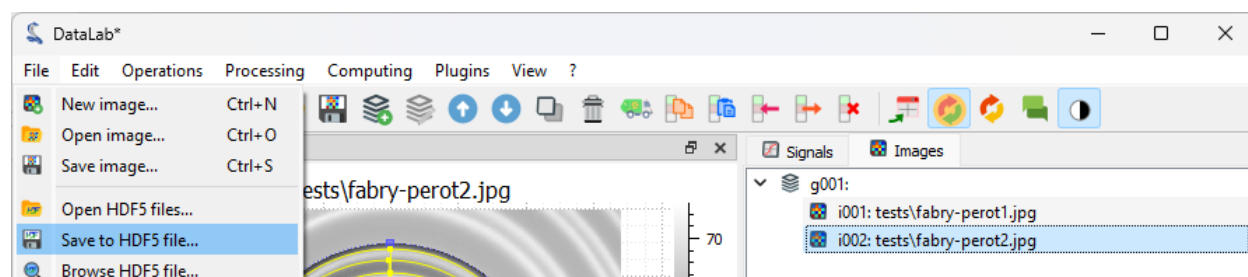


Fig. 68: Save the workspace to a file with “File > Save to HDF5 file...”, or the button in the toolbar.

- Plot the beam size as a function of the position along the propagation axis

First, we open DataLab and load the images:

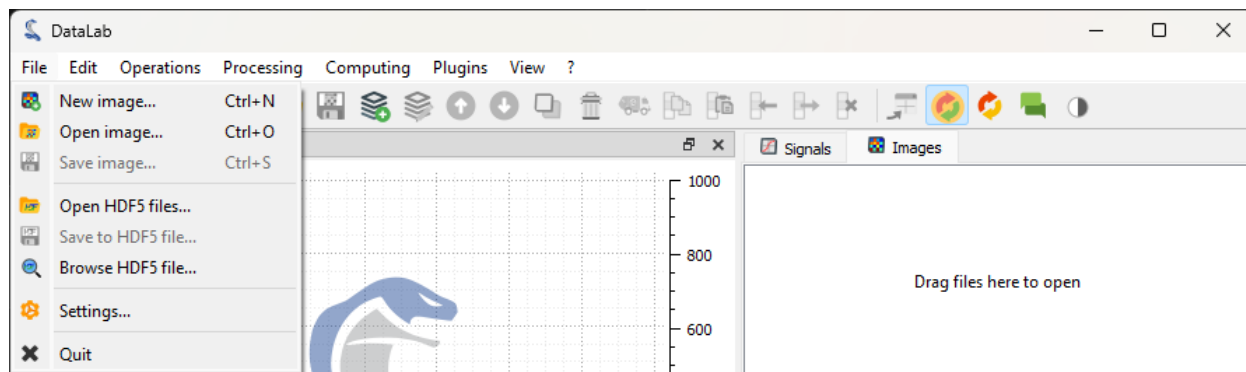


Fig. 69: Open the image files with “File > Open...”, or with the button in the toolbar, or by dragging and dropping the files into DataLab (on the panel on the right).

The selected images are loaded in the “Images” panel. The last image is displayed in the main window. On each image, we can zoom in and out by pressing the right mouse button and dragging the mouse up and down. We can also pan the image by pressing the middle mouse button and dragging the mouse.

**Note:** If we want to display the images side by side, we can select the “Distribute on a grid” entry in the “Operations” menu.

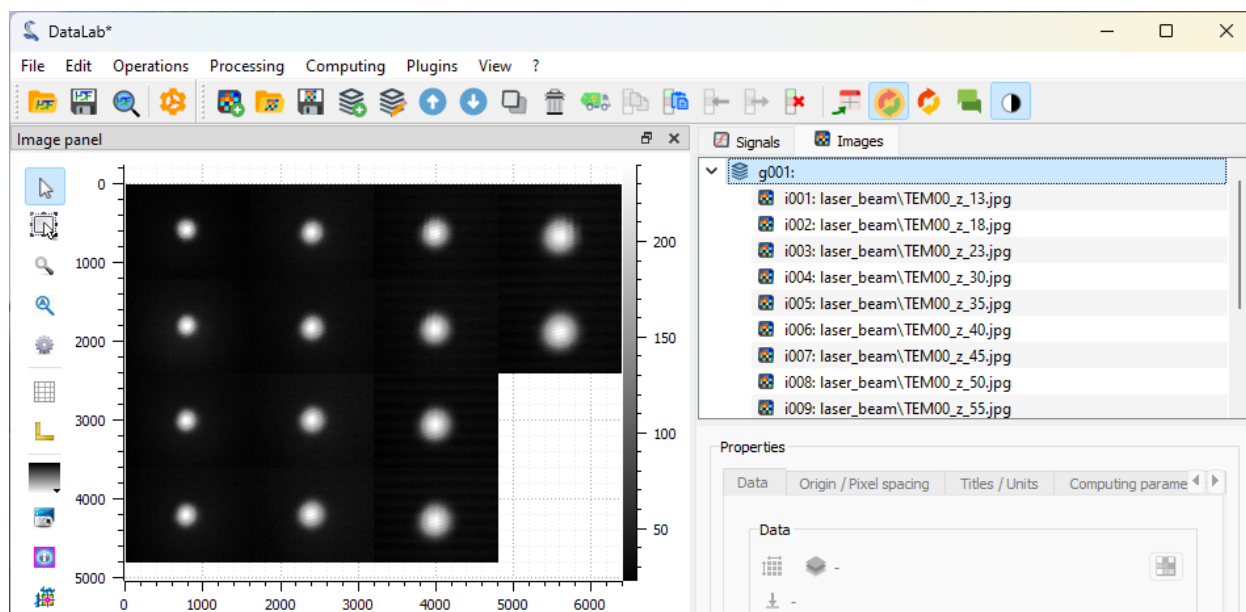


Fig. 72: Images distributed on a 4 rows grid

But, let's go back to the initial display by selecting the “Reset image positions” entry in the “Operations” menu.

If we select one of the images, we can see that there is background noise, so it might be useful to apply a threshold to the images.

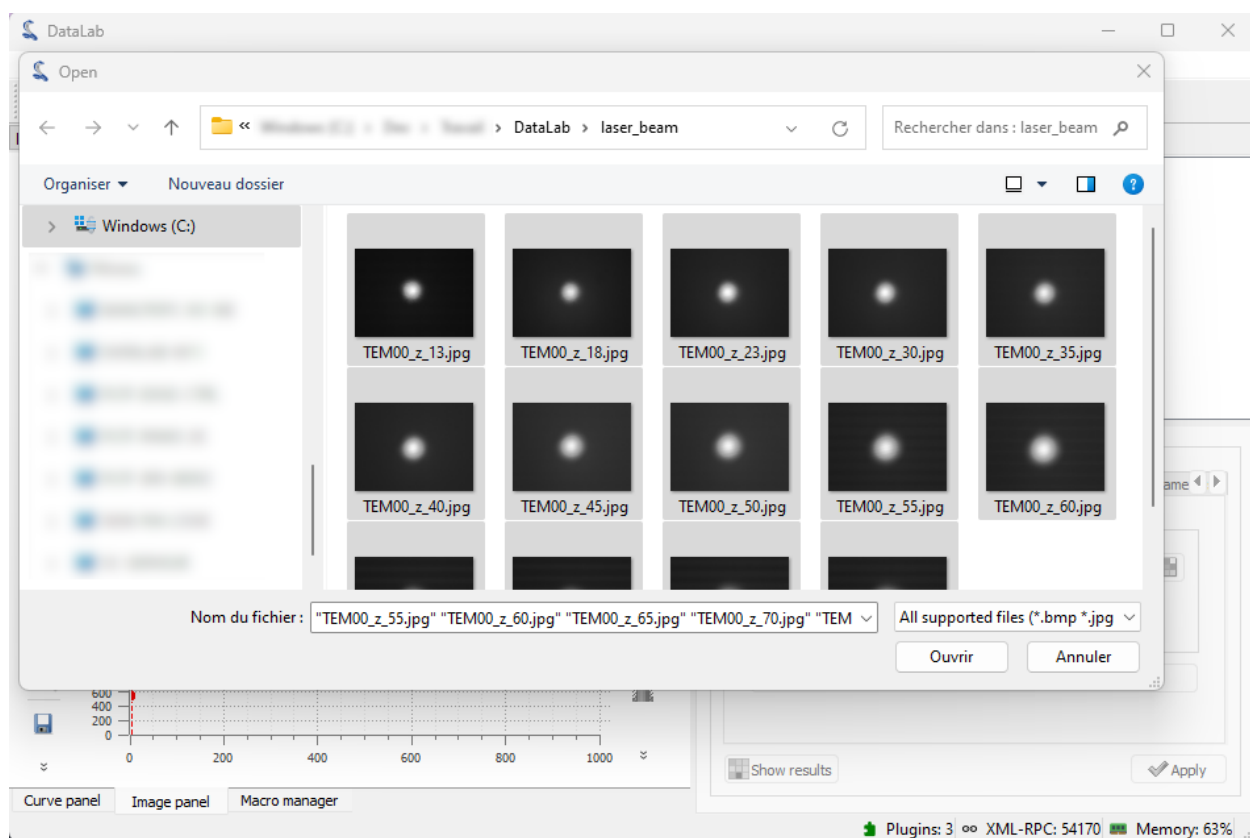


Fig. 70: Select the test images “TEM00\_z\_\*.jpg” and click “Open”.

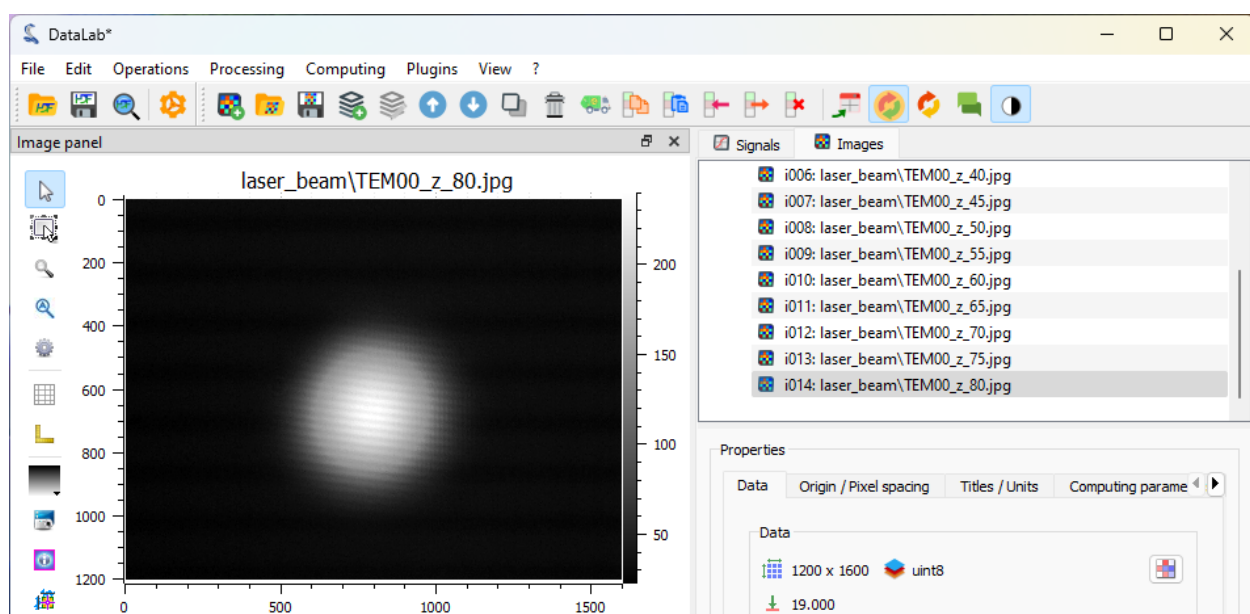


Fig. 71: Zoom in and out with the right mouse button. Pan the image with the middle mouse button.



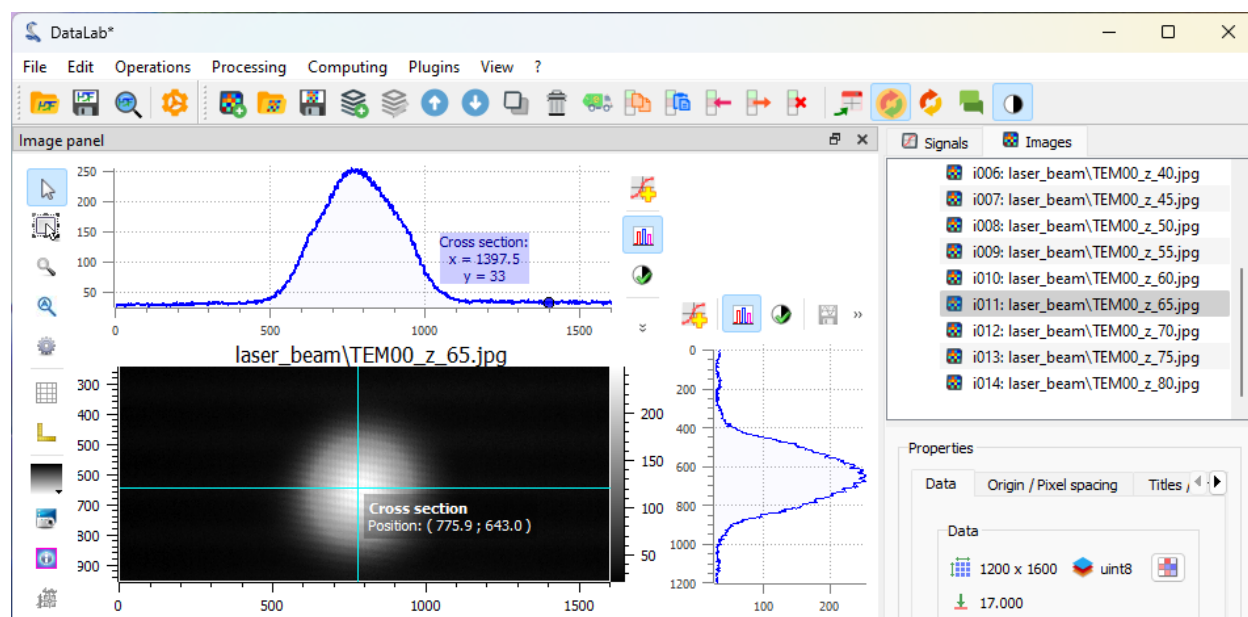



Fig. 73: Select one of the images in the “Images” panel, select the associated image in the visualization panel, and enable the “Cross section” tool  in the vertical toolbar on the left of the visualization panel (this tool is a [PlotPy](#) feature). On this figure, we can see that the background noise is around 30 lsb (to show the curve marker, we had to select the profile curve and right-click on it to display the context menu, and select “Markers > Bound to active item”).

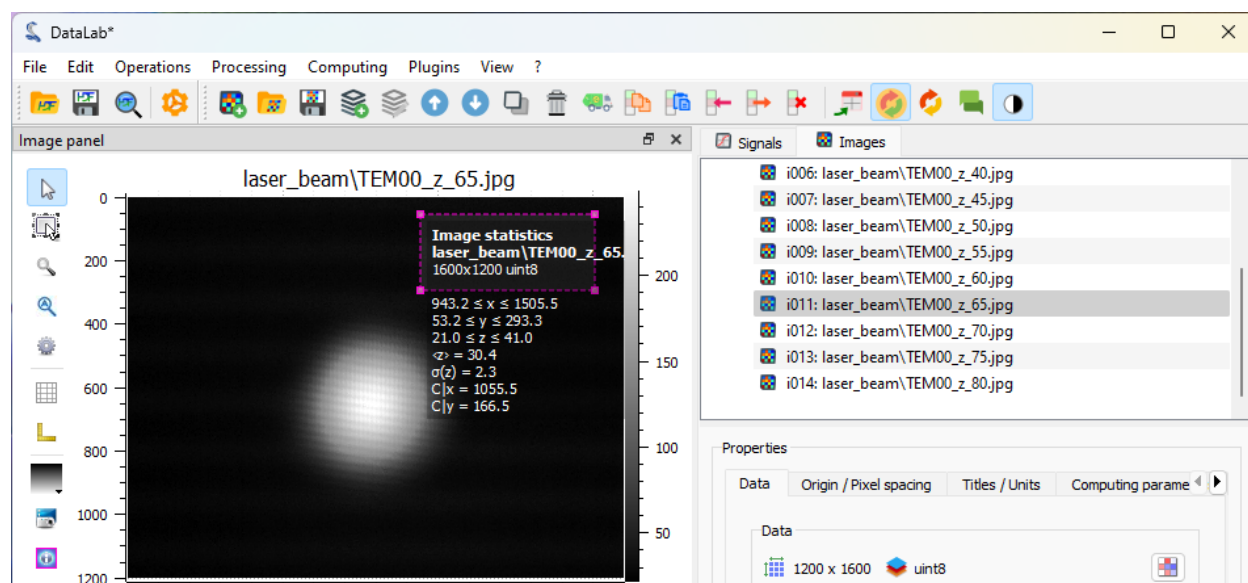



Fig. 74: Another way to measure the background noise is to use the “Image statistics” tool  in the vertical toolbar on the left of the visualization panel. It displays statistics on a the rectangular area defined by dragging the mouse on the image. This confirms that the background noise is around 30 lsb.



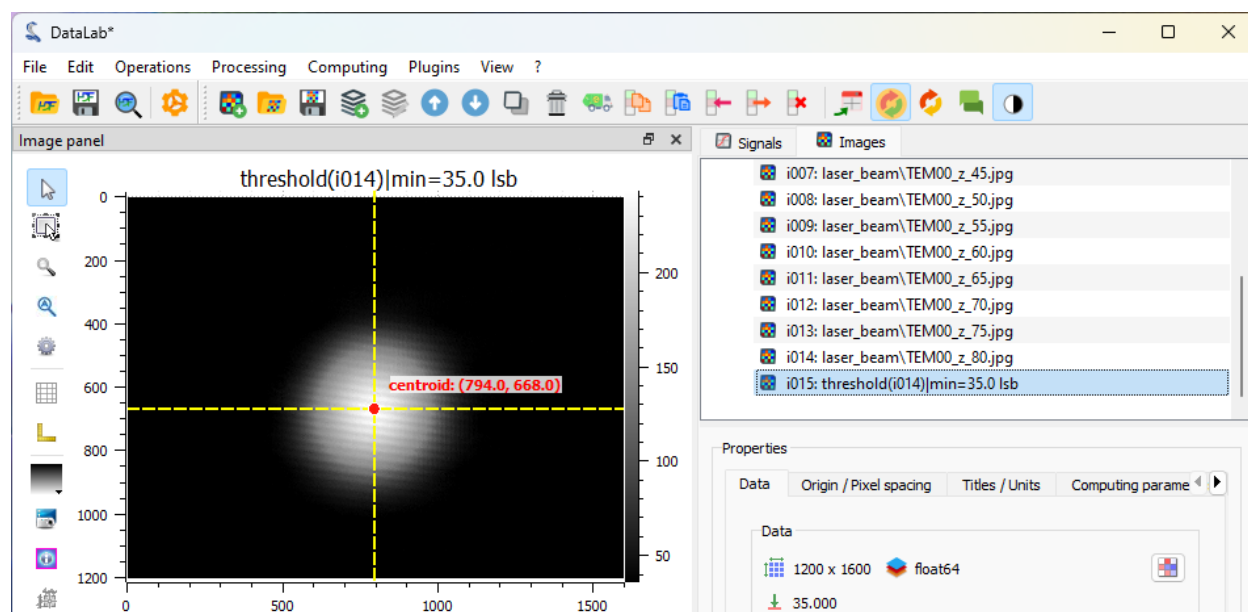


Fig. 75: After applying a threshold at 35 lsb (with “Processing > Thresholding...”), we can compute a more accurate position of the beam center using “Analysis > Centroid”.

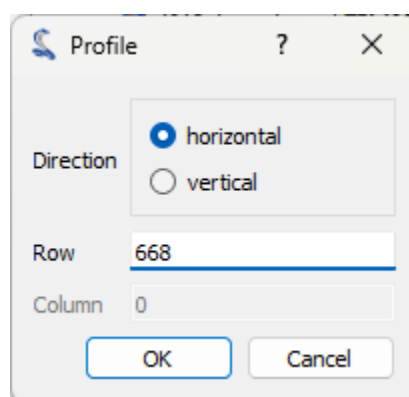


Fig. 76: Then we can extract a line profile along the horizontal axis with “Operations > Intensity profiles > Line profile”. We set the row position to the centroid position computed previously (i.e. 668).



Fig. 77: The intensity profile is displayed in the “Signals” panel. We can fit the profile to a Gaussian function with “Processing > Fitting > Gaussian fit”. Here we have selected both signals.

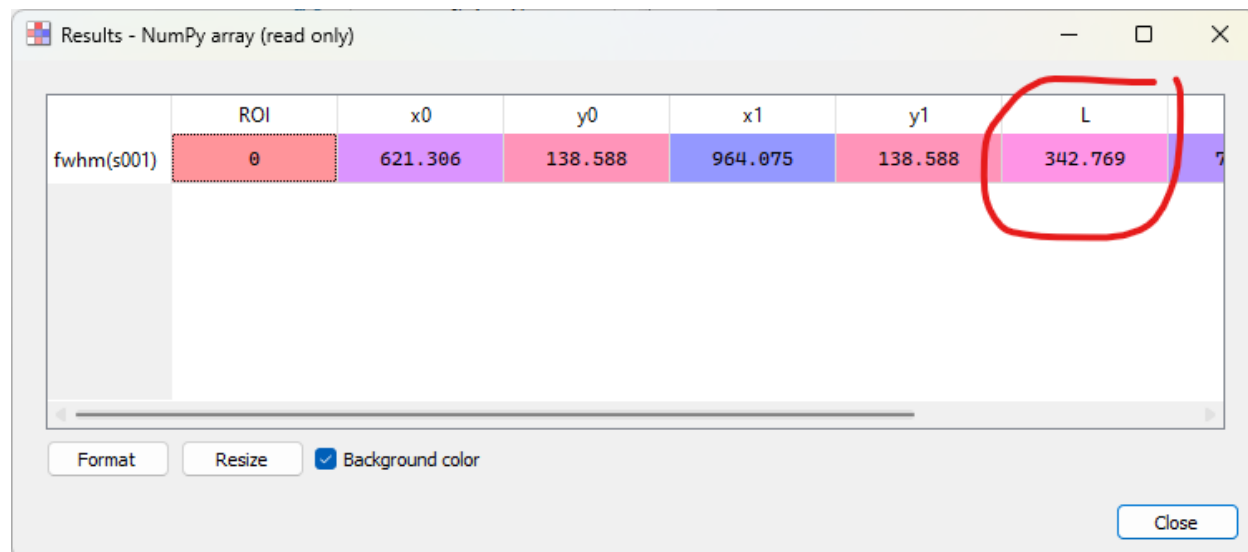


Fig. 78: If we go back to the first signal, the intensity profile, we can also directly compute the FWHM with “Analysis > Full width at half maximum”. The “Results” dialog displays a lot of information about the computation, including the FWHM value (that is the *L* column, “L” for “Length” because the result shape is a segment and FWHM is the length of the segment).

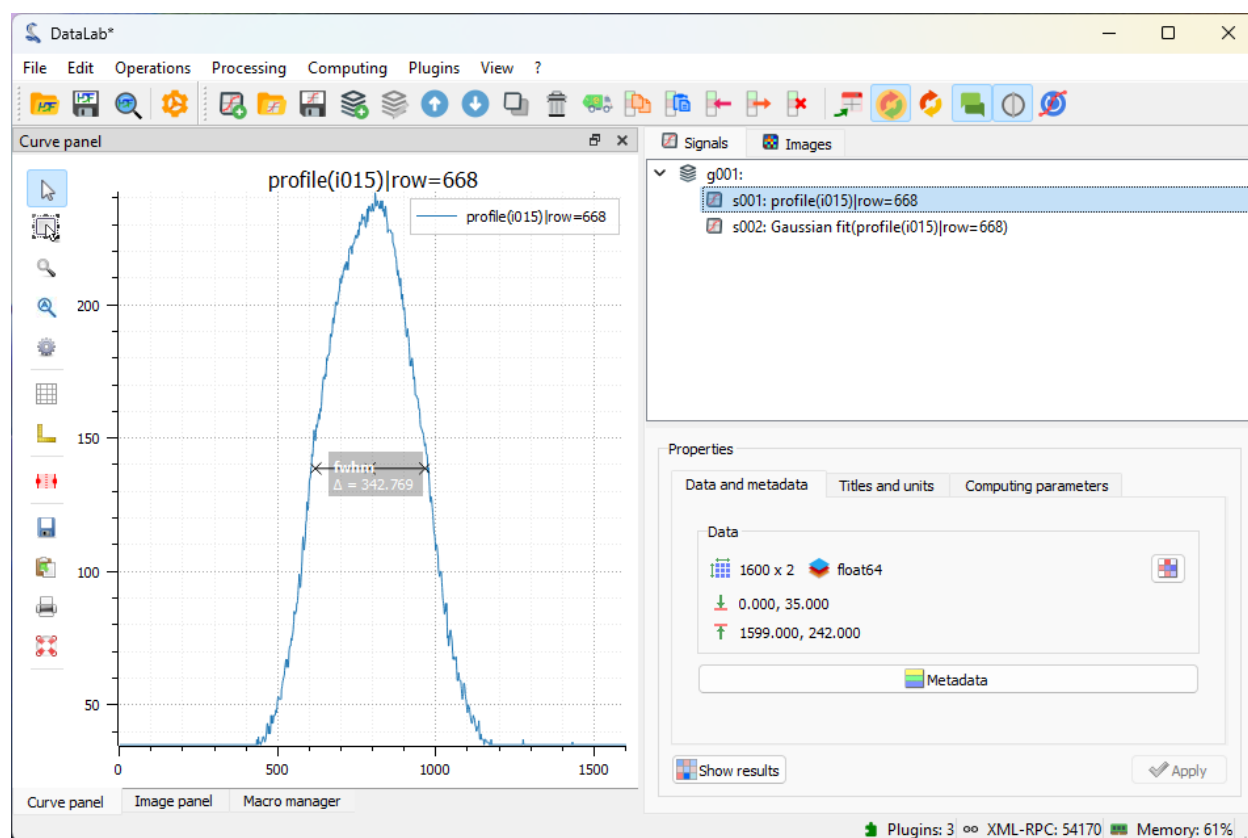


Fig. 79: The FWHM is also displayed on the curve, with an optional label (here, the title of this measurement has been displayed with “View > Show graphical object titles” or the button in the toolbar).

Now, let's try another method to measure the beam size.

From the “Images” panel, we can extract the radial intensity profile with “Operations > Intensity profiles > Radial profile”.

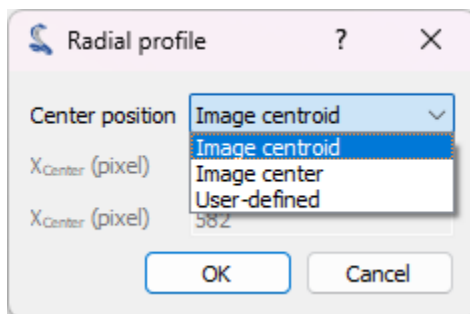


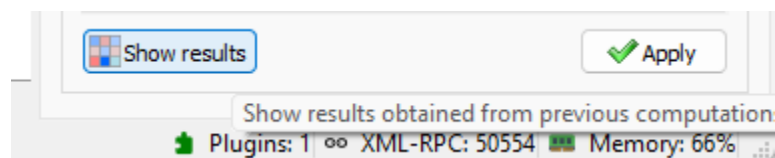
Fig. 80: The radial intensity profile may be computed around the centroid position, or around the center of the image, or around a user-defined position. Here we have selected the centroid position.

All these operations and computations that we have performed on a single image can be applied to all the images in the “Images” panel.

To do that, we begin by cleaning the “Signals” panel (with “Edit > Delete all” or the button in the toolbar). We also clean the intermediate results in the “Images” panel by selecting the images obtained during our prototyping and deleting them individually (with “Edit > Remove” or the button).

Then, we select all the images in the “Images” panel (individually, or by selecting the whole group “g001”).

**Note:** If you want to show the analysis results again, you can select the “Show results” entry in the “Analysis” menu, or the “Show results” button, below the image list:



Finally, we can save the workspace to a file. The workspace contains all the images and signals that were loaded or processed in DataLab. It also contains the analysis results, the visualization settings (colormaps, contrast, etc.), the metadata, and the annotations.

If you want to load the workspace again, you can use the “File > Open HDF5 file...” (or the button in the toolbar) to load the whole workspace, or the “File > Browse HDF5 file...” (or the button in the toolbar) to load only a selection of data sets from the workspace.

## Prototyping a custom processing pipeline

This example shows how to prototype a custom image processing pipeline using DataLab:

- Define a custom processing function
- Create a macro-command to apply the function to an image
- Use the same code from an external IDE (e.g. Spyder) or a Jupyter notebook
- Create a plugin to integrate the function in the DataLab GUI

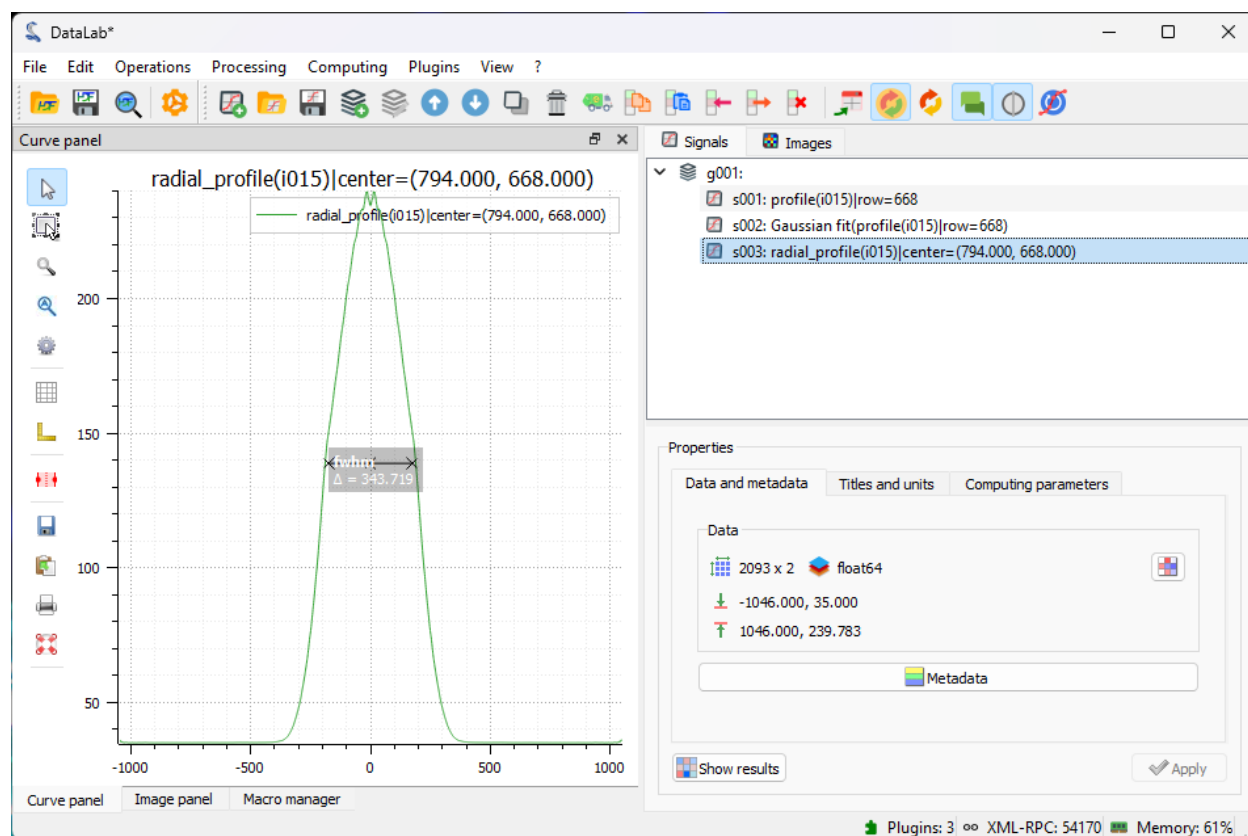


Fig. 81: The radial intensity profile is displayed in the “Signals” panel. It is smoother than the line profile, because it is computed from a larger number of pixels, thus averaging the noise.

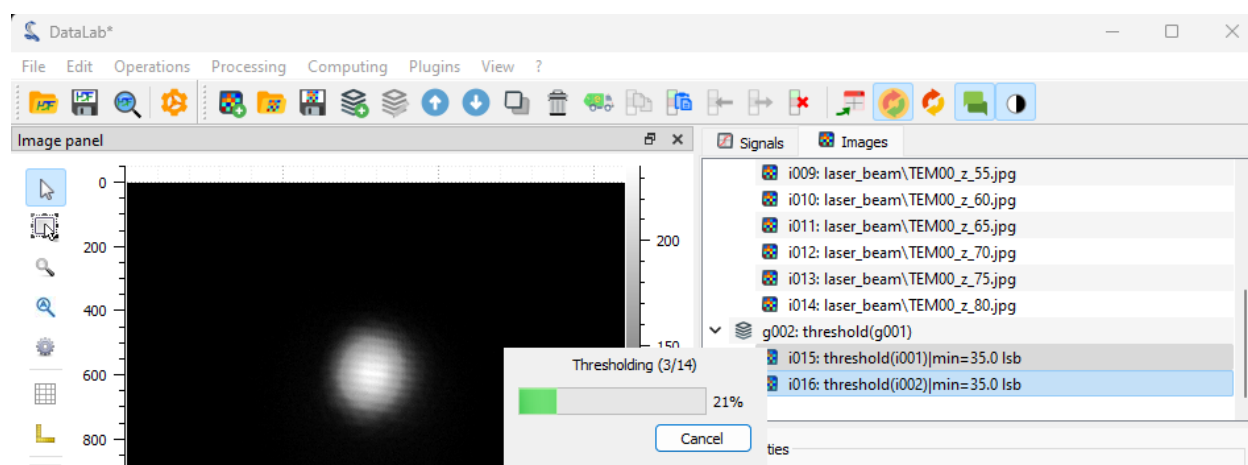


Fig. 82: We apply the threshold to all the images, and then we extract the radial intensity profile for all the images (after selecting the whole group “g002” - it should be automatically selected if you had selected “g001” before applying the threshold).

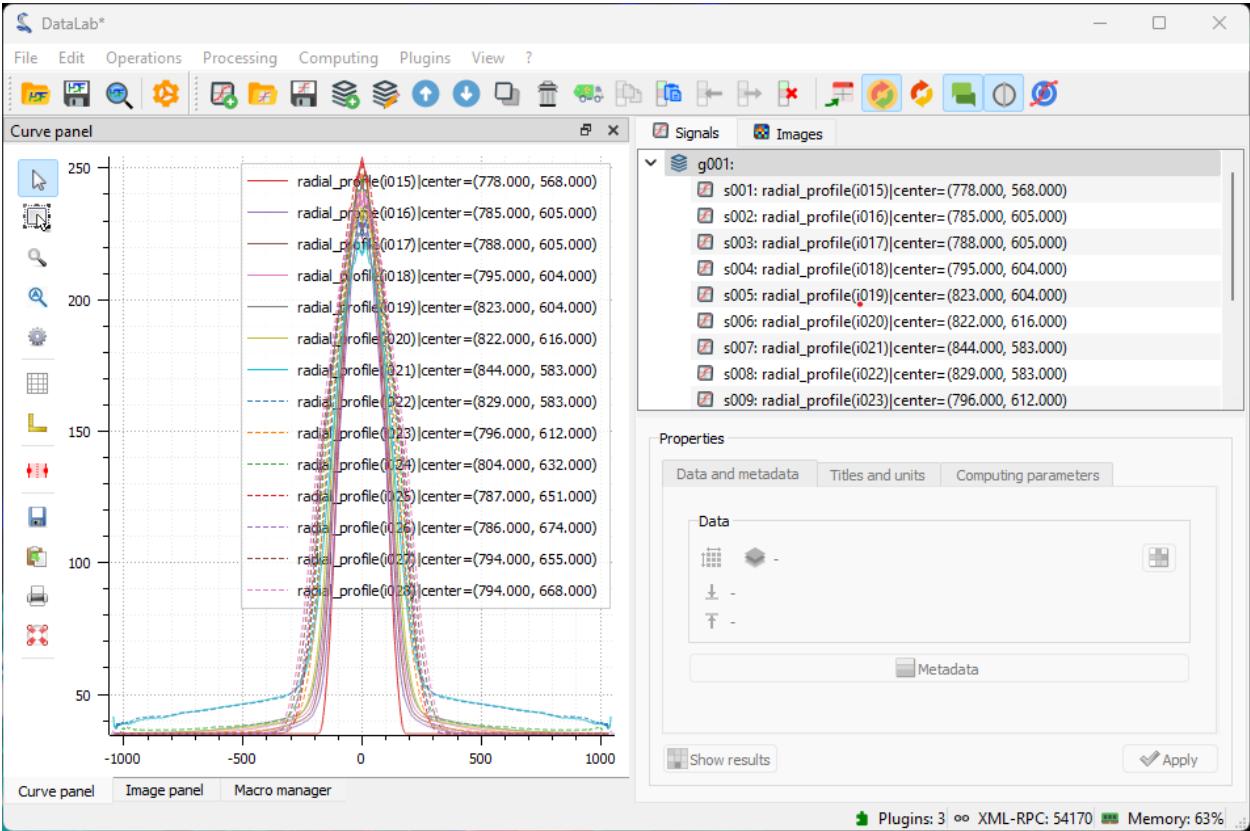


Fig. 83: The “Signals” panel now contains all the radial intensity profiles.

Results - NumPy array (read only)

	ROI	x0	y0	x1	y1	L	Xc	Yc
fwhm(s001)	0	-90.146	146.086	90.1454	146.086	180.291	-0.000307247	146.086
fwhm(s002)	0	-97.5273	134.95	97.5273	134.95	195.055	-2.84217e-14	134.95
fwhm(s003)	0	-102.844	144.563	102.845	144.563	205.689	0.000226296	144.563
fwhm(s004)	0	-108.246	140.839	108.246	140.839	216.492	-1.66472e-05	140.839
fwhm(s005)	0	-114.898	134.226	114.898	134.226	229.796	-1.0366e-05	134.226
fwhm(s006)	0	-120.149	138.229	120.147	138.229	240.295	-0.000828078	138.229
fwhm(s007)	0	-132.471	134.219	132.471	134.219	264.942	2.84217e-14	134.219
fwhm(s008)	0	-136.738	138.216	136.737	138.216	273.475	-0.000167016	138.216
fwhm(s009)	0	-136.727	139.425	136.727	139.425	273.454	-0.000201918	139.425

Format: Resize: ☒ Background color

Close

Fig. 84: We can compute the FWHM of all the radial intensity profiles: the “Results” dialog displays the FWHM values for all the profiles.

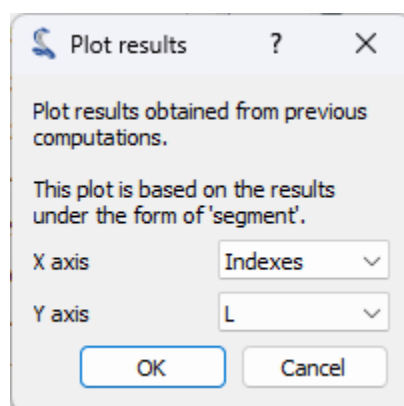


Fig. 85: Finally, we can plot the beam size as a function of the position along the propagation axis. To do that, we use the “Plot results” feature in the “Analysis” menu. This feature allows to plot result data sets by choosing the x and y axes among the result columns. Here, we choose the to plot the FWHM values ( $L$ ) as a function of the image index (*Indices*).

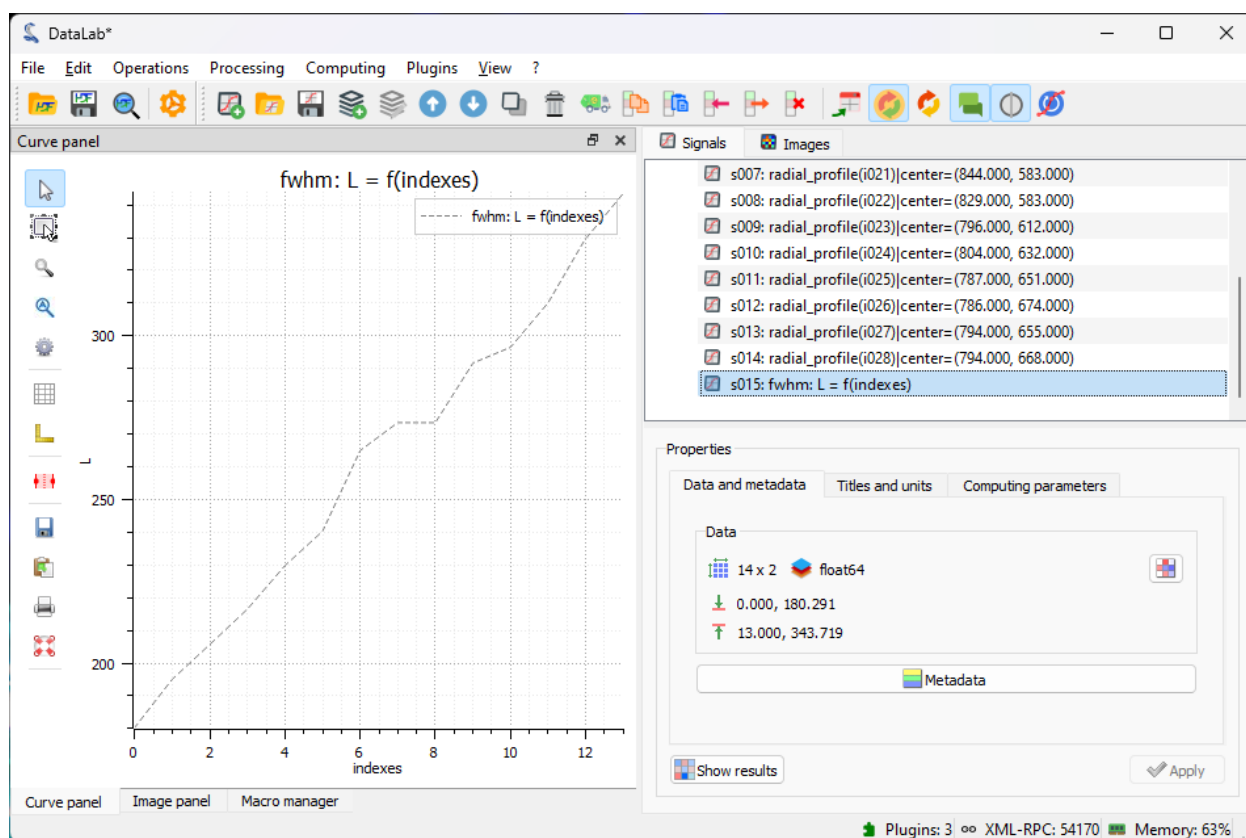


Fig. 86: The plot is displayed in the “Signals” panel and shows that the beam size increases with the position along the propagation axis (the position is here in arbitrary units, the image index).

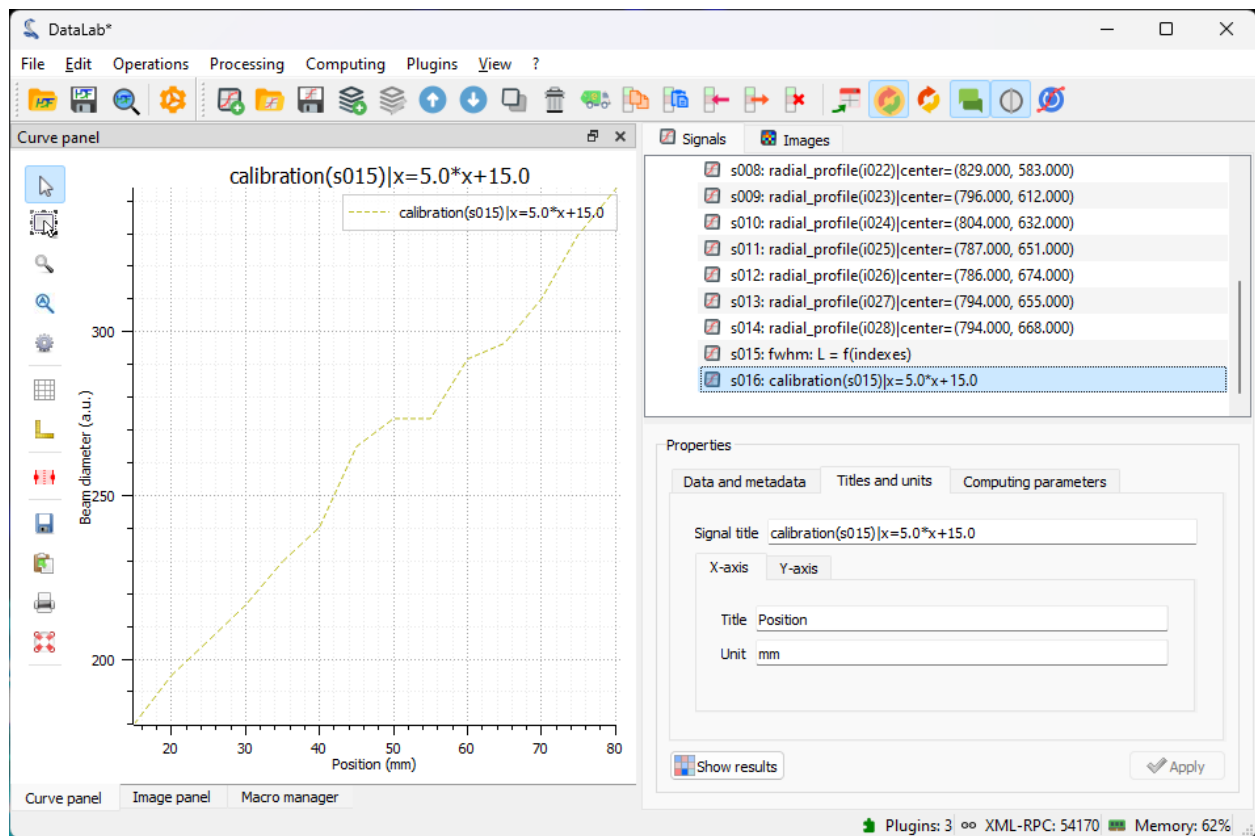


Fig. 87: We can also calibrate the X and Y axis using “Processing > Linear calibration”. Here we have set the X axis to the position in mm (and entered the title and unit in the “Properties” group box).



## Define a custom processing function

For illustrating the extensibility of DataLab, we will use a simple image processing function that is not available in the standard DataLab distribution, and that represents a typical use case for prototyping a custom processing pipeline.

The function that we will work on is a denoising filter that combines the ideas of averaging and edge detection. This filter will average the pixel values in the neighborhood, but with a twist: it will give less weight to pixels that are significantly different from the central pixel, assuming they might be part of an edge or noise.

Here is the code of the `weighted_average_denoise` function:

```
def weighted_average_denoise(data: np.ndarray) -> np.ndarray:
    """Apply a custom denoising filter to an image.

    This filter averages the pixels in a 5x5 neighborhood, but gives less weight
    to pixels that significantly differ from the central pixel.
    """

    def filter_func(values: np.ndarray) -> float:
        """Filter function"""
        central_pixel = values[len(values) // 2]
        differences = np.abs(values - central_pixel)
        weights = np.exp(-differences / np.mean(differences))
        return np.average(values, weights=weights)

    return spi.generic_filter(data, filter_func, size=5)
```

For testing our processing function, we will use a generated image from a DataLab plugin example (*plugins/examples/cdl\_example\_imageproc.py*). Before starting, make sure that the plugin is installed in DataLab (see the first steps of the tutorial *Detecting blobs on an image*).

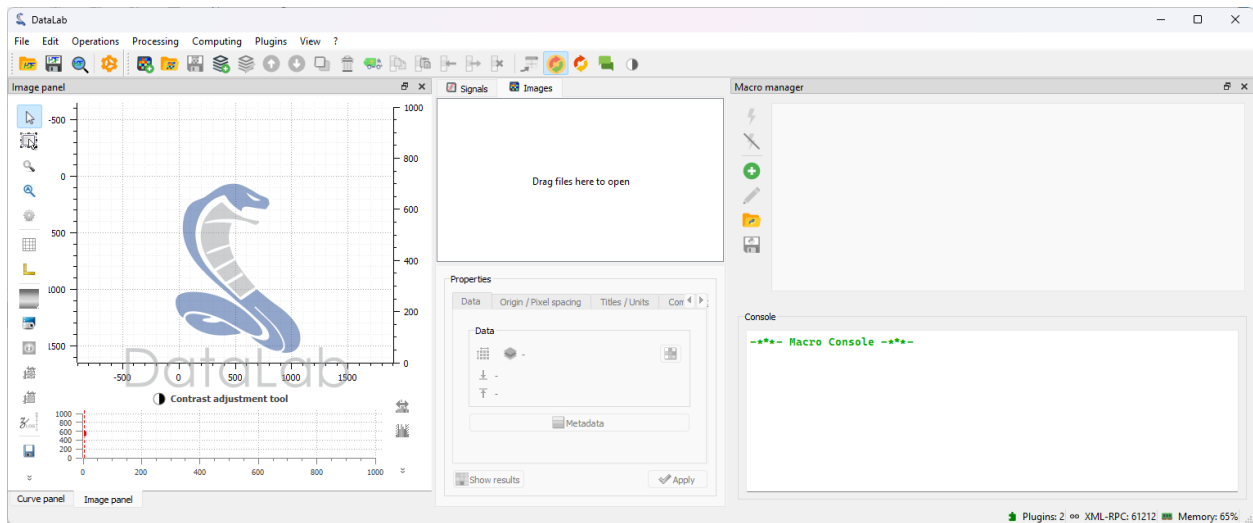


Fig. 88: To begin, we reorganize the window layout of DataLab to have the “Image Panel” on the left and the “Macro Panel” on the right.

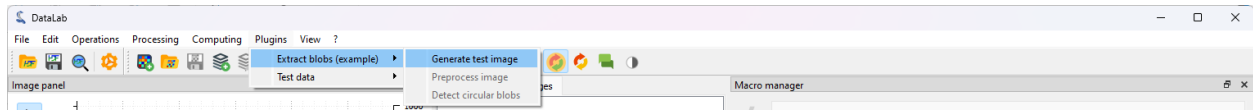


Fig. 89: We generate a new image using the “Plugins > Extract blobs (example) > Generate test image” menu.

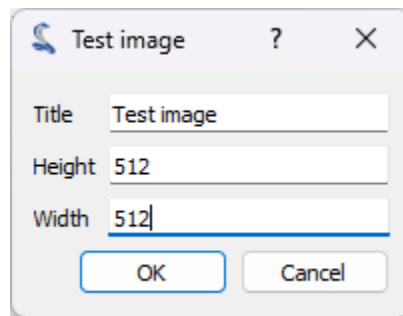


Fig. 90: We select a limited size for the image (e.g. 512x512 pixels) because our algorithm is quite slow, and click on “OK”.

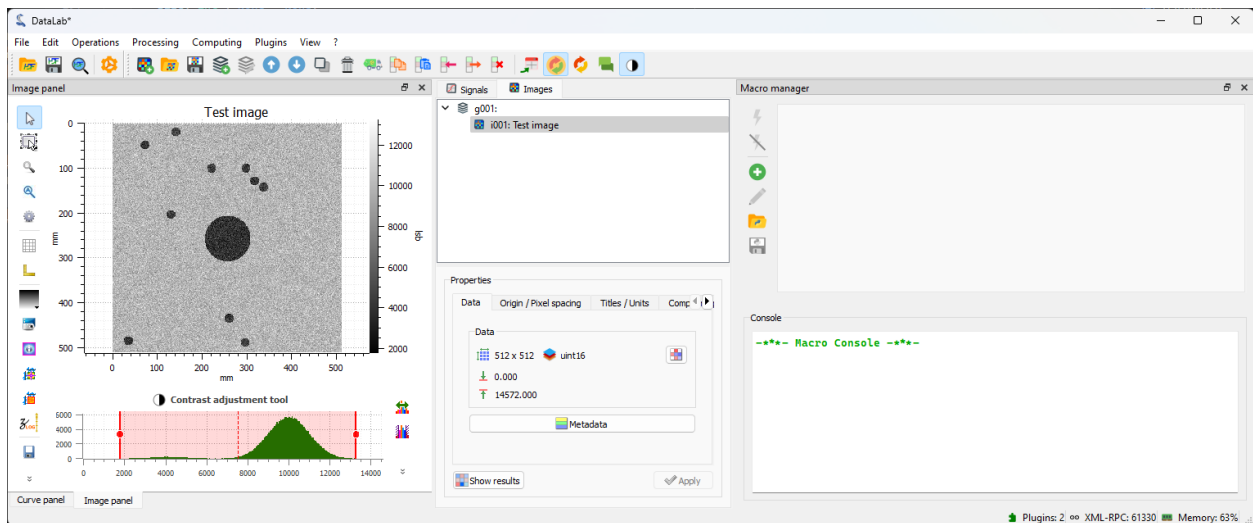


Fig. 91: We can now see the generated image in the “Image Panel”.

## Create a macro-command

Let's get back to our custom function. We can create a new macro-command that will apply the function to the current image. To do so, we open the "Macro Panel" and click on the "New macro" button.

DataLab creates a new macro-command which is not empty: it contains a sample code that shows how to create a new image and add it to the "Image Panel". We can remove this code and replace it with our own code:

```
# Import the necessary modules
import numpy as np
import scipy.ndimage as spi
from cdl.proxy import RemoteProxy

# Define our custom processing function
def weighted_average_denoise(values: np.ndarray) -> float:
    """Apply a custom denoising filter to an image.

    This filter averages the pixels in a 5x5 neighborhood, but gives less weight
    to pixels that significantly differ from the central pixel.
    """
    central_pixel = values[len(values) // 2]
    differences = np.abs(values - central_pixel)
    weights = np.exp(-differences / np.mean(differences))
    return np.average(values, weights=weights)

# Initialize the proxy to DataLab
proxy = RemoteProxy()

# Switch to the "Image Panel" and get the current image
proxy.set_current_panel("image")
image = proxy.get_object()
if image is None:
    # We raise an explicit error if there is no image to process
    raise RuntimeError("No image to process!")

# Get a copy of the image data, and apply the function to it
data = np.array(image.data, copy=True)
data = spi.generic_filter(data, weighted_average_denoise, size=5)

# Add new image to the panel
proxy.add_image("My custom filtered data", data)
```

In DataLab, macro-commands are simply Python scripts:

- Macros are part of DataLab's **workspace**, which means that they are saved and restored when exporting and importing to/from an HDF5 file.
- Macros are executed in a separate process, so we need to import the necessary modules and initialize the proxy to DataLab. The proxy is a special object that allows to communicate with DataLab.
- As a consequence, **when defining a plugin or when controlling DataLab from an external IDE, we can use exactly the same code as in the macro-command**. This is a very important point, because it means that we can prototype our processing pipeline in DataLab, and then use the same code in a plugin or in an external IDE to develop it further.

**Note:** The macro-command is executed in DataLab’s Python environment, so we can use the modules that are available in DataLab. However, we can also use our own modules, as long as they are installed in DataLab’s Python environment or in a Python distribution that is compatible with DataLab’s Python environment.

If your custom modules are not installed in DataLab’s Python environment, and if they are compatible with DataLab’s Python version, you can prepend the `sys.path` with the path to the Python distribution that contains your modules:

```
import sys
sys.path.insert(0, "/path/to/my/python/distribution")
```

This will allow you to import your modules in the macro-command and mix them with the modules that are available in DataLab.

**Warning:** If you use this method, make sure that your modules are compatible with DataLab’s Python version. Otherwise, you will get errors when importing them.

Now, let’s execute the macro-command by clicking on the “Run macro” button:

- The macro-command is executed in a separate process, so we can continue to work in DataLab while the macro-command is running. And, if the macro-command takes too long to execute, we can stop it by clicking on the “Stop macro” button.
- During the execution of the macro-command, we can see the progress in the “Macro Panel” window: the process standard output is displayed in the “Console” below the macro editor. We can see the following messages:
  - `---[...]-[# ==> Running 'Untitled 01' macro...]:` the macro-command starts
  - `Connecting to DataLab XML-RPC server...OK [...]:` the proxy is connected to DataLab
  - `---[...]-[# <== 'Untitled 01' macro has finished]:` the macro-command ends

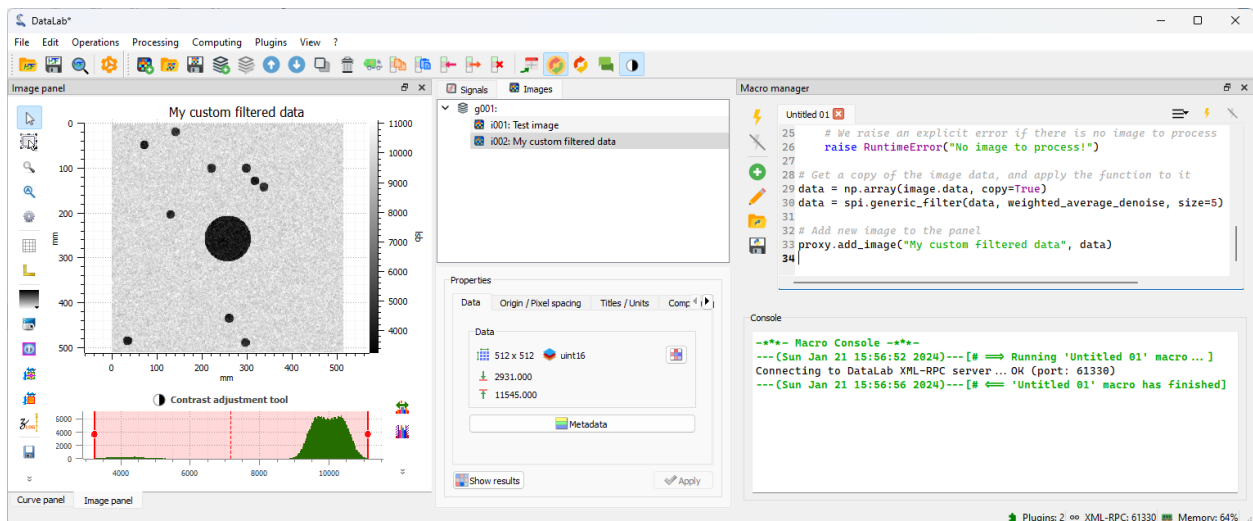


Fig. 92: When the macro-command has finished, we can see the new image in the “Image Panel”. Our filter has been applied to the image, and we can see that the noise has been reduced.

## Prototyping with an external IDE

Now that we have a working prototype of our processing pipeline, we can use the same code in an external IDE to develop it further.

For example, we can use the Spyder IDE to debug our code. To do so, we need to install Spyder but not necessarily in DataLab's Python environment (in the case of the stand-alone version of DataLab, it wouldn't be possible anyway).

The only requirement is to install a DataLab client in Spyder's Python environment:

- If you use the stand-alone version of DataLab or if you want or need to keep DataLab and Spyder in separate Python environments, you can install the [DataLab Simple Client](#) (`cdl-client`) using the `pip` package manager:

```
pip install cdl-client
```

Or you may also install the [DataLab Python package](#) (`cdl`) which includes the client (but also other modules, so we don't recommend this method if you don't need all DataLab's features in this Python environment):

```
pip install cdl
```

- If you use the DataLab Python package, you may run Spyder in the same Python environment as DataLab, so you don't need to install the client: it is already available in the main DataLab package (the `cdl` package).

Once the client is installed, we can start Spyder and create a new Python script:

```

1  # -*- coding: utf-8 -*-
2  """
3  Example of remote control of DataLab current session,
4  from a Python script running outside DataLab (e.g. in Spyder)
5
6  Created on Fri May 12 12:28:56 2023
7
8  @author: p.raybaut
9  """
10
11 # %% Importing necessary modules
12
13 import numpy as np
14 import scipy.ndimage as spi
15 from cdlclient import SimpleRemoteProxy
16
17 # %% Connecting to DataLab current session
18
19 proxy = SimpleRemoteProxy()
20 proxy.connect()
21
22 # %% Executing commands in DataLab (...)
23
24
25 # Define our custom processing function
26 def weighted_average_denoise(data: np.ndarray) -> np.ndarray:
27     """Apply a custom denoising filter to an image.
28
29     This filter averages the pixels in a 5x5 neighborhood, but gives less weight
30     to pixels that significantly differ from the central pixel.
```

(continues on next page)

(continued from previous page)

```

31 """
32
33 def filter_func(values: np.ndarray) -> float:
34     """Filter function"""
35     central_pixel = values[len(values) // 2]
36     differences = np.abs(values - central_pixel)
37     weights = np.exp(-differences / np.mean(differences))
38     return np.average(values, weights=weights)
39
40     return spi.generic_filter(data, filter_func, size=5)
41
42
43 # Switch to the "Image Panel" and get the current image
44 proxy.set_current_panel("image")
45 image = proxy.get_object()
46 if image is None:
47     # We raise an explicit error if there is no image to process
48     raise RuntimeError("No image to process!")
49
50 # Get a copy of the image data, and apply the function to it
51 data = np.array(image.data, copy=True)
52 data = weighted_average_denoise(data)
53
54 # Add new image to the panel
55 proxy.add_image("Filtered using Spyder", data)

```

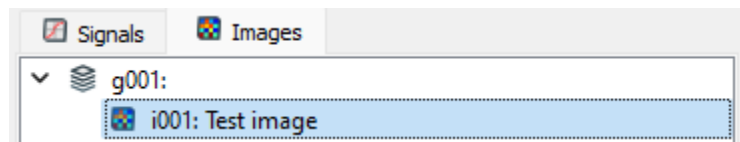


Fig. 93: We go back to DataLab and select the first image in the “Image Panel”.

## Prototyping with a Jupyter notebook

We can also use a Jupyter notebook to prototype our processing pipeline. To do so, we need to install Jupyter but not necessarily in DataLab’s Python environment (in the case of the stand-alone version of DataLab, it wouldn’t be possible anyway).

The only requirement is to install a DataLab client in Jupyter’s Python environment (see the previous section for more details: that is exactly the same procedure as for Spyder or any other IDE like Visual Studio Code, for example).

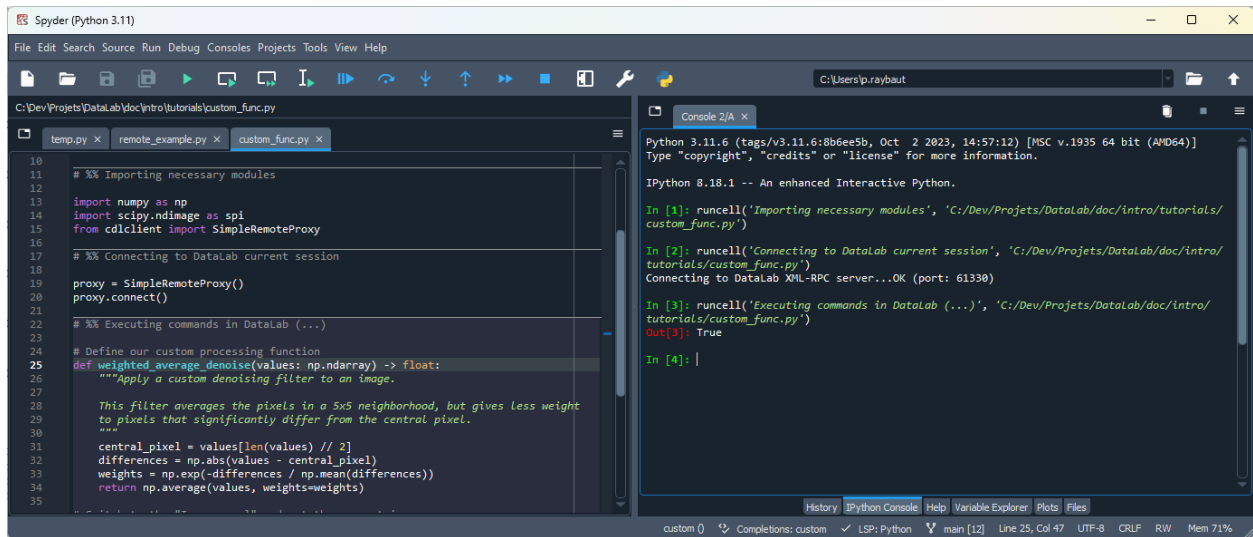


Fig. 94: Then, we execute the script in Spyder, step-by-step (using the defined cells), and we can see the result in DataLab.

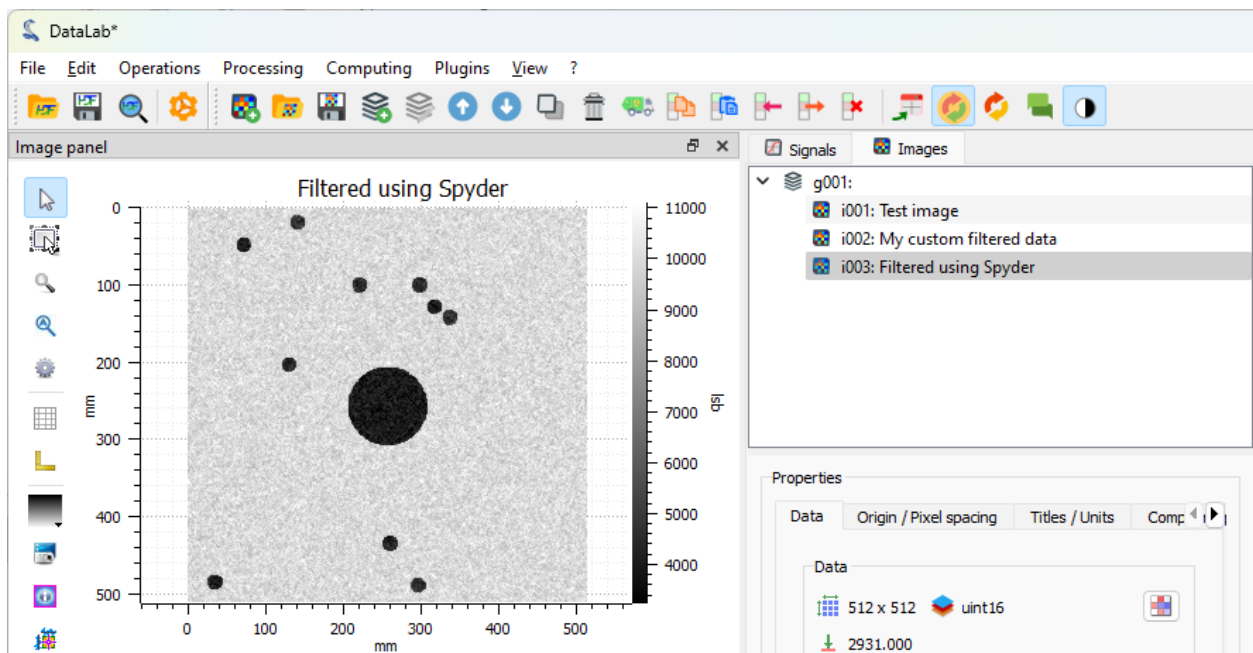


Fig. 95: We can see in DataLab that a new image has been added to the “Image Panel”. This image is the result of the execution of the script in Spyder. Here we have used the script without any modification, but we could have modified it to test new ideas, and then use the modified script in DataLab.

## DataLab custom function example

This is part of the DataLab's custom function tutorial which aims at illustrating the extensibility of DataLab (macros, plugins, and control from an external IDE or a Jupyter notebook).

The only requirement is to install the *DataLab Simple Client* package (using `pip install cdlclient`, for example).

```
[2]: import numpy as np
import scipy.ndimage as spi
from cdlclient import SimpleRemoteProxy

# Define our custom processing function
def weighted_average_denoise(values: np.ndarray) -> float:
    """Apply a custom denoising filter to an image.

    This filter averages the pixels in a 5x5 neighborhood, but gives less weight
    to pixels that significantly differ from the central pixel.
    """
    central_pixel = values[len(values) // 2]
    differences = np.abs(values - central_pixel)
    weights = np.exp(-differences / np.mean(differences))
    return np.average(values, weights=weights)
```

Connecting to DataLab current session:

```
[3]: proxy = SimpleRemoteProxy()
proxy.connect()

Connecting to DataLab XML-RPC server...OK (port: 61330)
```

Switch to the "Image panel" and get the current image

```
[5]: proxy.set_current_panel("image")
image = proxy.get_object()
if image is None:
    # We raise an explicit error if there is no image to process
    raise RuntimeError("No image to process!")
```

Get a copy of the image data, apply the function to it, and add new image to the panel

```
[6]: data = np.array(image.data, copy=True)
data = spi.generic_filter(data, weighted_average_denoise, size=5)
proxy.add_image("Filtered using a Jupyter notebook", data)
```

Fig. 96: Once the client is installed, we can start Jupyter and create a new notebook.

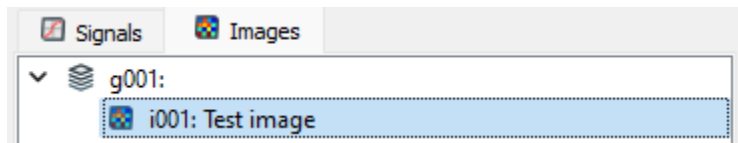


Fig. 97: We go back to DataLab and select the first image in the "Image Panel".

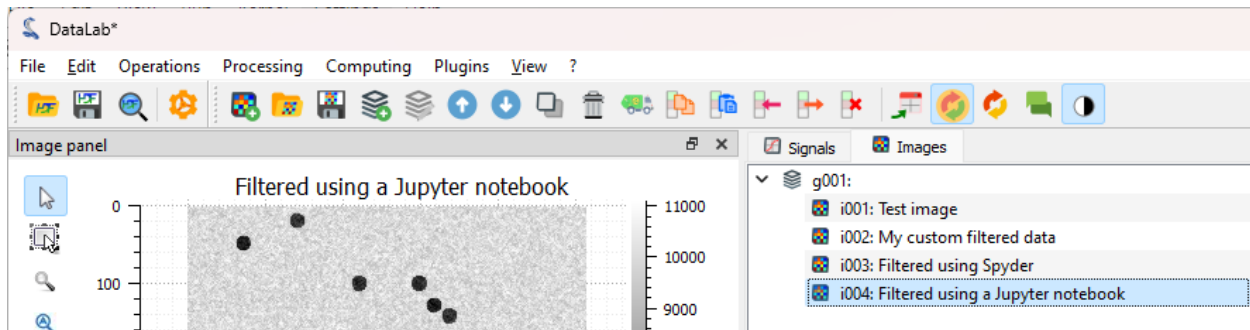


Fig. 98: Then, we execute the notebook in Jupyter, step-by-step (using the defined cells), and we can see the result in DataLab. Once again, we can see in DataLab that a new image has been added to the "Image Panel". This image is the result of the execution of the notebook in Jupyter. As for the script in Spyder, we could have modified the notebook to test new ideas, and then use the modified notebook in DataLab.



## Creating a plugin

Now that we have a working prototype of our processing pipeline, we can create a plugin to integrate it in DataLab's GUI. To do so, we need to create a new Python module that will contain the plugin code. We can use the same code as in the macro-command, but we need to make some changes.

### See also:

The plugin system is described in the *Plugins* section.

Apart from integrating the feature to DataLab's GUI which is more convenient for the user, the advantage of creating a plugin is that we can take benefit of the DataLab infrastructure, if we encapsulate our processing function in a certain way (see below):

- Our function will be executed in a separate process, so we can interrupt it if it takes too long to execute.
- Warnings and errors will be handled by DataLab, so we don't need to handle them ourselves.

The most significant change is that we need to define a function that will be operating on DataLab's native image objects (*cdl.obj.ImageObj*), instead of operating on NumPy arrays. So we need to find a way to call our custom function *weighted\_average\_denoise* with a *cdl.obj.ImageObj* as input and output. To avoid writing a lot of boilerplate code, we can use the function wrapper provided by DataLab: *cdl.computation.image.Wrap11Func*.

Besides we need to define a class that describes our plugin, which must inherit from *cdl.plugins.PluginBase* and name the Python script that contains the plugin code with a name that starts with *cdl\_* (e.g. *cdl\_custom\_func.py*), so that DataLab can discover it at startup.

Moreover, inside the plugin code, we want to add an entry in the "Plugins" menu, so that the user can access our plugin from the GUI.

Here is the plugin code:

```

1  # -*- coding: utf-8 -*-
2
3  """
4  Custom denoising filter plugin
5  =====
6
7  This is a simple example of a DataLab image processing plugin.
8
9  It is part of the DataLab custom function tutorial.
10
11  .. note::
12
13     This plugin is not installed by default. To install it, copy this file to
14     your DataLab plugins directory (see `DataLab documentation
15     <https://datalab-platform.com/en/features/general/plugins.html>`).
16  """
17
18  import numpy as np
19  import scipy.ndimage as spi
20
21  import cdl.computation.image as cpi
22  import cdl.obj
23  import cdl.param
24  import cdl.plugins
25

```

(continues on next page)

(continued from previous page)

```

26
27 def weighted_average_denoise(data: np.ndarray) -> np.ndarray:
28     """Apply a custom denoising filter to an image.
29
30     This filter averages the pixels in a 5x5 neighborhood, but gives less weight
31 to pixels that significantly differ from the central pixel.
32     """
33
34     def filter_func(values: np.ndarray) -> float:
35         """Filter function"""
36         central_pixel = values[len(values) // 2]
37         differences = np.abs(values - central_pixel)
38         weights = np.exp(-differences / np.mean(differences))
39         return np.average(values, weights=weights)
40
41     return spi.generic_filter(data, filter_func, size=5)
42
43
44 class CustomFilters(cdl.plugins.PluginBase):
45     """DataLab Custom Filters Plugin"""
46
47     PLUGIN_INFO = cdl.plugins.PluginInfo(
48         name="My custom filters",
49         version="1.0.0",
50         description="This is an example plugin",
51     )
52
53     def create_actions(self) -> None:
54         """Create actions"""
55         acth = self.imagepanel.acthandler
56         proc = self.imagepanel.processor
57         with acth.new_menu(self.PLUGIN_INFO.name):
58             for name, func in (("Weighted average denoise", weighted_average_denoise),):
59                 # Wrap function to handle ``ImageObj`` objects instead of NumPy arrays
60                 wrapped_func = cpi.Wrap11Func(func)
61                 acth.new_action(
62                     name, triggered=lambda: proc.compute_11(wrapped_func, title=name)
63                 )

```

To test it, we have to add the plugin script to one of the plugin directories that are discovered by DataLab at startup (see the *Plugins* section for more details, or the *Detecting blobs on an image* for an example).

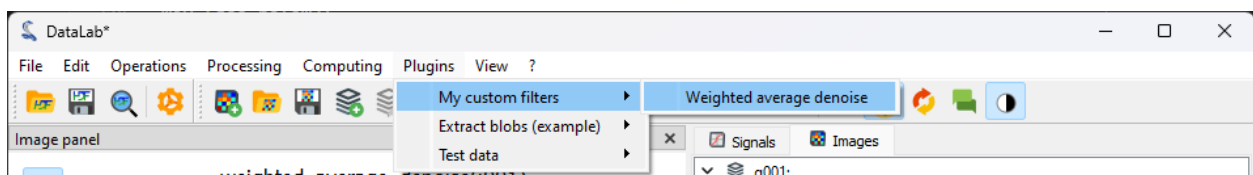


Fig. 99: We restart DataLab and we can see that the plugin has been loaded.

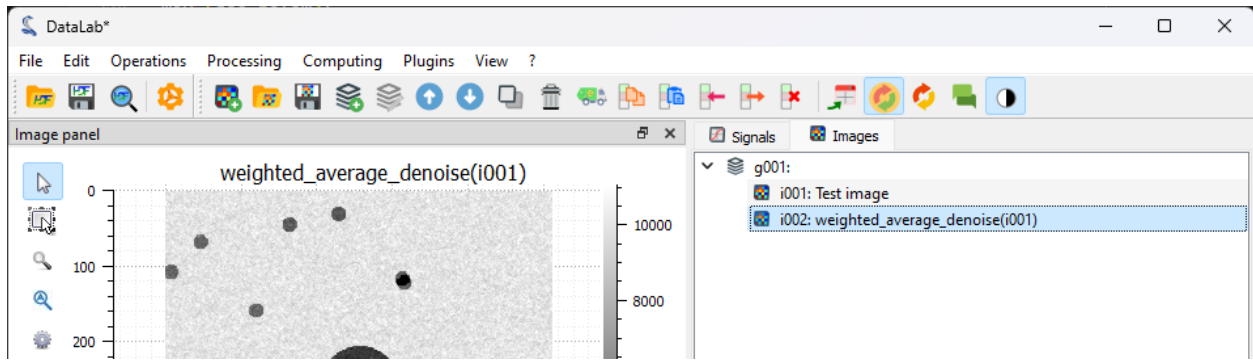
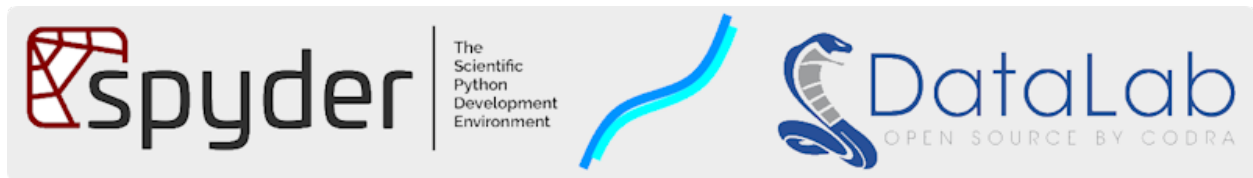


Fig. 100: We generate again our test image using (see the first steps of the tutorial), and we process it using the plugin: “Plugins > My custom filters > Weighted average denoise”.

### DataLab and Spyder: a perfect match

This tutorial shows how to use [Spyder](#) to work with DataLab through an example, using fake algorithms and data that represent an hypothetical research/technical work. The goal is to illustrate how to use DataLab to test your algorithms with some data, and how to debug them if necessary.

The example is quite simple, but it illustrates the basic concepts of working with DataLab and [Spyder](#).



**Note:** DataLab and [Spyder](#) are **complementary** tools. While [Spyder](#) is a powerful development environment with interactive scientific computing capabilities, DataLab is a versatile data analysis tool that may be used to perform a wide range of tasks, from simple data visualization to complex data analysis and processing. In other words, [Spyder](#) is a **development** tool, while DataLab is a **data analysis** tool. You can use [Spyder](#) to develop algorithms and then use DataLab to analyze data with those algorithms.

### Basic concepts

In the context of your research or technical work, we assume that you are developing a software to process data (signals or images): this software may either be a stand-alone application or a library that you will use in other applications, or even a simple script that you will run from the command line. In any case, you will need to follow a development process that will include the following steps:

0. Prototype the algorithm in a development environment, such as [Spyder](#).
1. Develop the algorithm in a development environment, such as [Spyder](#).
2. Test the algorithm with some data.
3. Debug the algorithm if necessary.
4. Repeat steps 2 and 3 until the algorithm works as expected.
5. Use the algorithm in your application.

---

**Note:** DataLab can help you with step 0 because it provides all the processing primitives that you need to prototype your algorithm: you can load data, visualize it, and perform basic processing operations. We won't cover this step in the following paragraphs because the DataLab documentation already provides a lot of information about it.

---

In this tutorial, we will see how to use DataLab to perform steps 2 and 3. We assume that you already have prototyped (preferably in DataLab!) and developed your algorithm in [Spyder](#). Now, you want to test it with some data, but without quitting [Spyder](#) because you may need to do some changes to your algorithm and re-test it. Besides, your workflow is already set up in [Spyder](#) and you don't want to change it.

---

**Note:** In this tutorial, we assume that you have already installed DataLab and that you have started it. If you haven't done it yet, please refer to the [Installation](#) section of the documentation.

Besides, we assume that you have already installed [Spyder](#) and that you have started it. If you haven't done it yet, please refer to the [Spyder](#) documentation. **Note that you don't need to install DataLab in the same environment as Spyder.** that's the whole point of DataLab, it is a stand-alone application that can be used from any environment. For this tutorial, you only need to install the DataLab Simple Client (`pip install cdlclient`) in the same environment as [Spyder](#).

---

## Testing your algorithm with DataLab

Let's assume that you have developed algorithms in the `my_work` module of your project. You have already prototyped them in DataLab, and you have developed them in [Spyder](#) by writing functions that take some data as input and return some processed data as output. Now, you want to test these algorithms with some data.

To test these algorithms, you have written two functions in the `my_work` module:

- `test_my_1d_algorithm`: this function returns some 1D data that will allow you to validate your first algorithm which works on 1D data.
- `test_my_2d_algorithm`: this function returns some 2D data that will allow you to validate your second algorithm which works on 2D data.

You can now use DataLab to visualize the data returned by these functions directly from [Spyder](#):

- First, you need to start both DataLab and [Spyder](#).
- Remember that DataLab is a stand-alone application that can be used from any environment, so you don't need to install it in the same environment as [Spyder](#) because the connection between these two applications is done through a communication protocol.

Here is how to do it:

```
# %% Connecting to DataLab current session

from cdlclient import SimpleRemoteProxy

proxy = SimpleRemoteProxy()
proxy.connect()

# %% Visualizing 1D data from my work

from my_work import test_my_1d_algorithm
```

(continues on next page)

(continued from previous page)

```

x, y = test_my_1d_algorithm() # Here is all my research/technical work!
proxy.add_signal("My 1D result data", x, y) # Let's visualize it in DataLab
proxy.compute_wiener() # Denoise the signal using the Wiener filter

# %% Visualizing 2D data from my work

from my_work import test_my_2d_algorithm

z = test_my_2d_algorithm()[1] # Here is all my research/technical work!
proxy.add_image("My 2D result data", z) # Let's visualize it in DataLab

```

If we execute the first two cells, we will see the following output in the Spyder console:

```

Python 3.11.5 (tags/v3.11.5:cce6ba9, Aug 24 2023, 14:38:34) [MSC v.1936 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

```

```

IPython 8.15.0 -- An enhanced Interactive Python.

```

```

In [1]: runcell('Connecting to DataLab current session', 'my_work_test_with_cdl.py')
Connecting to DataLab XML-RPC server...OK (port: 54577)

```

```

In [2]: runcell('Visualizing 1D data from my work', 'my_work_test_with_cdl.py')
Out[2]: True

```

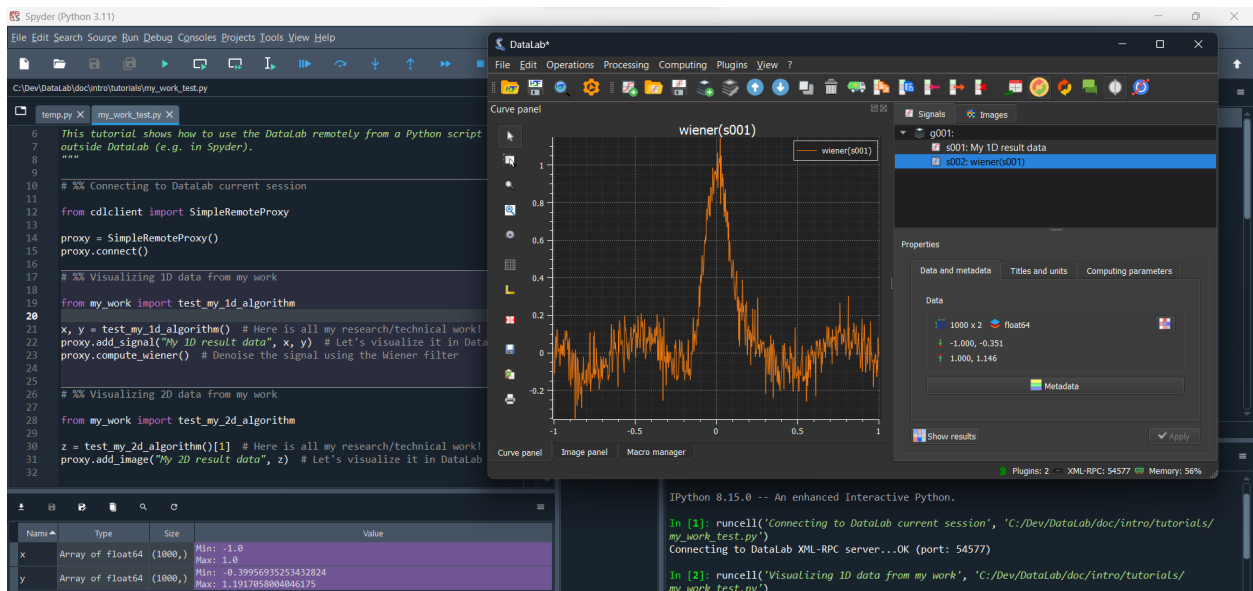


Fig. 101: On this screenshot, we can see the result of evaluating the first two cells: the first cell connects to DataLab, and the second cell visualizes the 1D data returned by the `test_my_1d_algorithm` function.

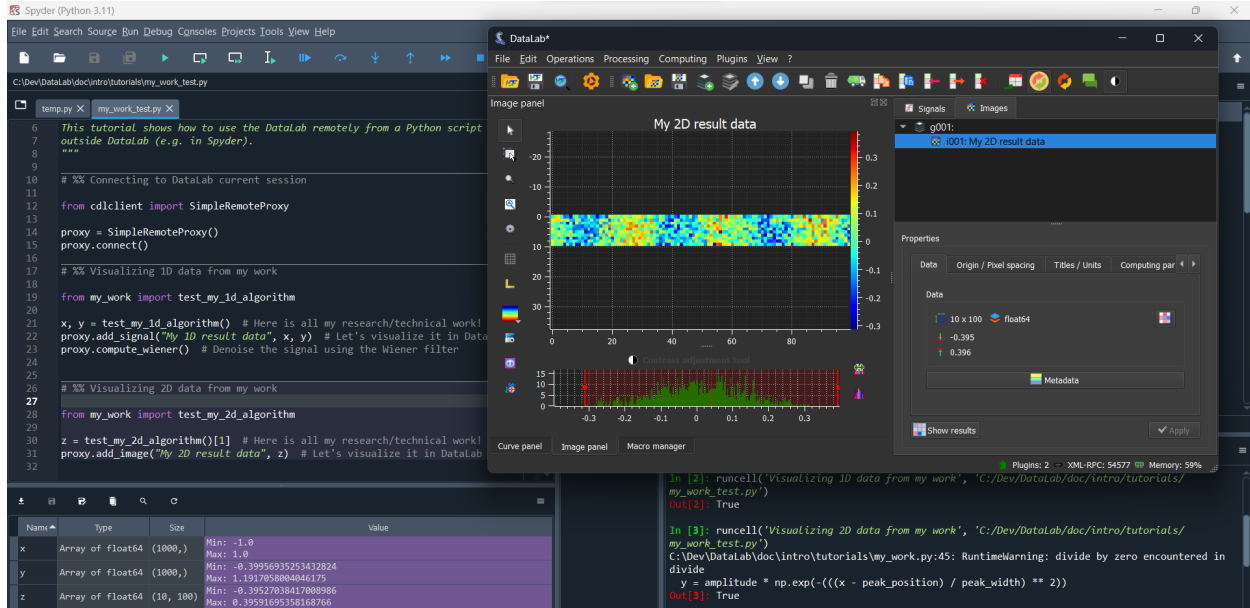


Fig. 102: On this screenshot, we can see the result of evaluating the third cell: the `test_my_2d_algorithm` function returns a 2D array, and we can visualize it directly in DataLab.

## Debugging your algorithm with DataLab

Now that you have tested your algorithms with some data, you may want to debug them if necessary. To do so, you can combine the [Spyder](#) debugging capabilities with DataLab.

Here is the code of the fake algorithm that we want to debug, in which we have introduced an optional `debug_with_datalab` parameter that - if set to `True` - will create a proxy object allowing to visualize the data step-by-step in DataLab:

```
def generate_2d_data(
    num_lines: int = 10,
    noise_std: float = 0.1,
    x_min: float = -1,
    x_max: float = 1,
    num_points: int = 100,
    debug_with_datalab: bool = False,
) -> tuple[np.ndarray, np.ndarray]:
    """Generate 2D data that can be used in the tutorials to represent some results.

    Args:
        num_lines: Number of lines in the generated data, by default 10.
        noise_std: Standard deviation of the noise, by default 0.1.
        x_min: Minimum value of the x axis, by default -1.
        x_max: Maximum value of the x axis, by default 1.
        num_points: Number of points in the generated data, by default 100.
        debug_with_datalab: Whether to use the DataLab to debug the function,
            by default False.

    Returns:
        Generated data (x, y; where y is a 2D array and x is a 1D array).
```

(continues on next page)

(continued from previous page)

```

"""
proxy = None
if debug_with_datalab:
    proxy = SimpleRemoteProxy()
    proxy.connect()
z = np.zeros((num_lines, num_points))
for i in range(num_lines):
    amplitude = 0.1 * i**2
    peak_position = 0.5 * i**2
    peak_width = 0.1 * i**2
    x, y = generate_1d_data(
        amplitude,
        peak_position,
        peak_width,
        relaxation_oscillations=True,
        noise=True,
        noise_std=noise_std,
        x_min=x_min,
        x_max=x_max,
        num_points=num_points,
    )
    z[i] = y
    if proxy is not None:
        proxy.add_signal(f"Line {i}", x, y)
return x, z

```

The corresponding `test_my_2d_algorithm` function also has an optional `debug_with_datalab` parameter that is simply passed to the `generate_2d_data` function.

Now, we can use [Spyder](#) to debug the `test_my_2d_algorithm` function:

```

# %% Debugging my work with DataLab

from my_work import generate_2d_data

x, z = generate_2d_data(debug_with_datalab=True)

```

In this simple example, the algorithm is just iterating 10 times and generating a 1D array at each iteration. Each 1D array is then stacked in a 2D array that is returned by the `generate_2d_data` function. With the `debug_with_datalab` parameter set to `True`, we can visualize each 1D array in DataLab: that way, we can check that the algorithm is working as expected.

---

**Note:** If we had executed the script using [Spyder](#) debugger and set a breakpoint in the `generate_2d_data` function, we would have seen the generated 1D arrays in DataLab at each iteration: since DataLab is executed in a separate process, we would have been able to manipulate the data in DataLab while the algorithm is paused in [Spyder](#).

---

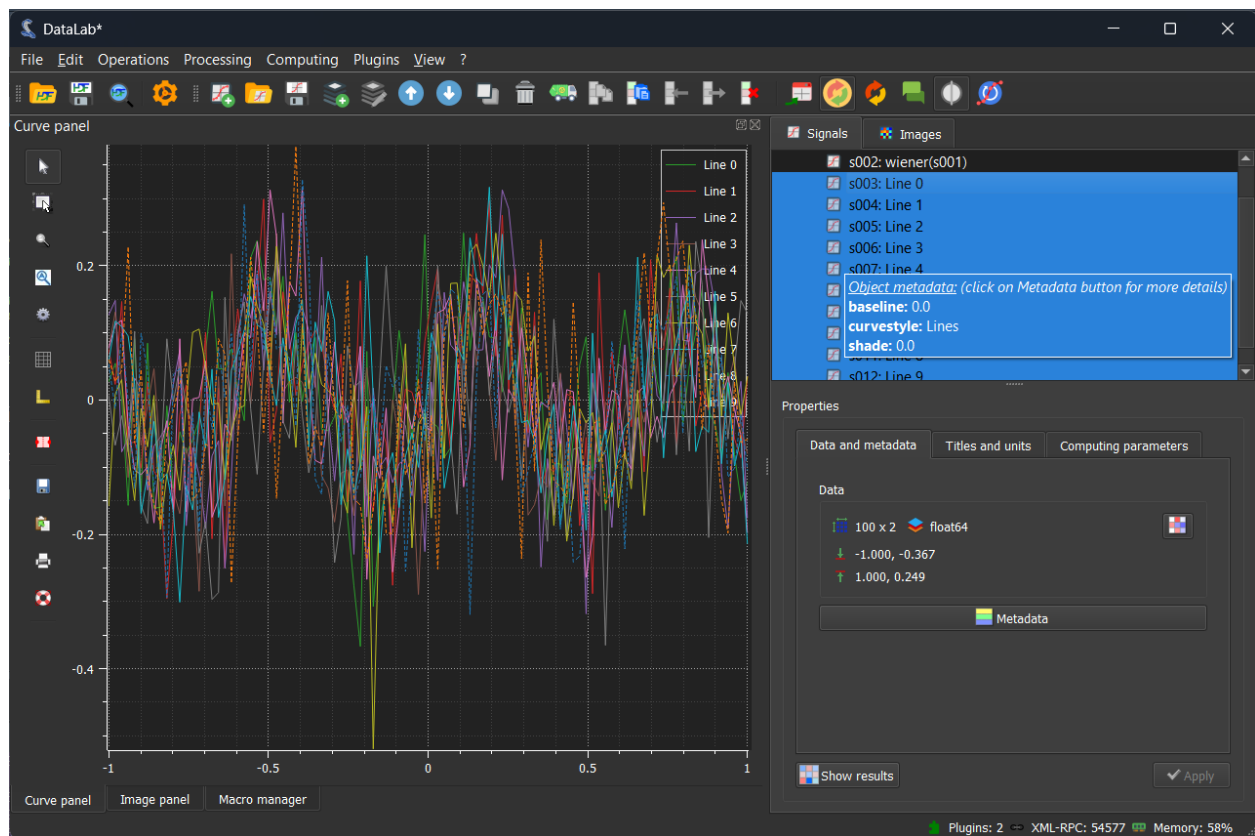


Fig. 103: On this screenshot, we can see the result of evaluating the first cell: the `test_my_2d_algorithm` function is called with the `debug_with_dataLAB` parameter set to `True`: 10 1D arrays are generated and visualized in DataLab.



## FEATURES

The following sections describe the features of DataLab, the open-source scientific data analysis and visualization platform.

First, the **validation** section explains why and how DataLab is validated - that is another singular and fundamental feature of DataLab.

Then, the **general features** section describes the general features of DataLab, which concern both the signal and image processing panels.

The **signal processing** and **image processing** sections describe the features specific to the signal and image processing panels, respectively.

---

**Note:** Before jumping into the details of other parts of the documentation, it is recommended to start with the [Workspace](#) page, which describes the basic concepts of DataLab.

---

**See also:**

For a synthetic overview of the features of DataLab, please refer to the [Key features](#) page.

### 2.1 Validation

DataLab is a platform for scientific data analysis and visualization. It may be used in a variety of scientific disciplines, including biology, physics, and astronomy. The common ground for all these disciplines is the need to validate the results of computational analysis against ground-truth data. This is a critical step in the scientific method, and it is essential for reproducibility and trust in the results. This is what we call **technical validation**.

DataLab is also used in industrial applications, where the validation of the results is essential for guaranteeing the quality of the process but a wider range of validation is required to ensure that a maximum of use cases are covered from a functional point of view. This is what we call **functional validation**.

Thus, DataLab validation may be categorized into two types:

- **Technical validation:** ensures that the results of computational analysis are accurate and reliable.
- **Functional validation:** checks that the software behaves as expected in a variety of use cases (classic automated unit test suite).

### 2.1.1 Technical Validation

DataLab technical validation is based on two key concepts: **ground-truth data** and **analytical validation**.

#### Ground-truth data

Ground-truth data is data that is known to be correct. It is used to validate the results of computational analysis.

In DataLab, ground-truth data may be obtained from a variety of sources, including:

- Experimental data
- Simulated data
- Synthetic data
- Data from a trusted source

#### Analytical validation

Analytical validation is the process of comparing the results of computational analysis to ground-truth data. This is done to ensure that the results are accurate and reliable.

In DataLab, analytical validation is implemented using a variety of techniques, including:

- Cross-validation with an analytical model (from a trusted source, e.g. [SciPy](#) or [NumPy](#))
- Statistical analysis
- Visual inspection
- Expert review

#### Scope

The scope of technical validation in DataLab includes all compute functions that operate on DataLab's signal and image objects (i.e. `cdl.obj.SignalObj` and `cdl.obj.ImageObj`).

This includes functions for (all functions are named `compute_<function_name>`):

- Signal processing (`cdl.computation.signal`)
- Image processing (`cdl.computation.image`)

#### Implementation

The tests are implemented using the [pytest](#) framework.

When writing a new technical validation test, the following rules should be followed regarding the test function:

- The test function should be named:
  - `test_signal_<function_name>` for signal compute functions
  - `test_image_<function_name>` for image compute functions

---

**Note:** The `signal` or `image` prefix is used to indicate the type of object that the function operates on. It may be omitted if the function operates exclusively on one type of object (e.g. `test_adjust_gamma` is the test function for the `compute_adjust_gamma` function, which operates on images).

---

- The test function should be marked with the `@pytest.mark.validation` decorator.

Following those rules ensures that:

- The tests are easily identified as technical validation tests.
- The tests can be executed separately using the command line interface (see [Run technical validation tests](#)).
- The tests are automatically discovered for synthesizing the validation status of the compute functions (see [Validation Status of DataLab](#)).

## Executing tests

In DataLab, technical validation tests are disseminated in the test suite of the project, but they can also be executed separately using the command line interface.

### See also:

See paragraph [Run technical validation tests](#) for more information on how to run technical validation tests.

## 2.1.2 Functional validation

### Strategy

DataLab functional validation is based a classic test strategy, with a strong emphasis on automated testing. Apart from one or two manual tests (e.g. load test), all tests are automated (more than 99% of the tests are automated).

Writing tests follows the TDD (Test-Driven Development) principle:

- When *a new feature is developed*, the developer writes the tests first. The tests are then executed to ensure that they fail. The developer then implements the feature, and the tests are executed again to ensure that they pass.
- When *a bug is reported*, the developer writes a test that reproduces the bug. The test is executed to ensure that it fails. The developer then fixes the bug, and the test is executed again to ensure that it passes.

Depending on the abstraction level, unit tests and/or application tests are written. When writing both types of tests, the developer starts with the unit tests and then writes the application tests.

### Types of tests

The functional validation of DataLab is based on two main types of tests:

- **Unit tests** (test scripts named `*_unit_test.py`): Test individual functions or methods. All unit tests are automated.
- **Application tests** (test scripts named `*_app_test.py`): Test the interaction between components (integration tests), or the application as a whole. All application tests are automated.

## Implementation

The tests are implemented using the `pytest` framework. Many existing tests may be derived from to create new tests.

## Executing tests

To execute the tests, the developer uses the command line interface. See section *Run complete test suite* for more information on how to run functional validation tests.

## 2.1.3 Validation Status of DataLab

### Functional validation

In DataLab, functional validation is based on a classic test strategy (see *Functional validation*).

Test coverage is around 90%, with more than 200 tests.

### Technical validation

This paragraph provides the validation status of compute functions in DataLab (this is what we call technical validation, see *Technical Validation*).

---

**Note:** This is a work in progress: the tables below are updated continuously as new functions are validated or test code is adapted (the tables are generated from the test code). Some functions are already validated but do not appear in the list below yet, while others are still in the validation process.

---

**Warning:** The validation status must not be confused with the test coverage. The validation status indicates whether the function has been validated against ground-truth data or analytical models. The test coverage indicates the percentage of the code that is executed by the test suite, but it does not necessarily take into account the correctness of the results (DataLab's test coverage is around 90%).

Table 1: Validation Statistics

Category	Signal	Image	Total
Number of compute functions	66	96	162
Number of validated compute functions	56	84	140
Percentage of validated compute functions	84%	87%	86%

## Signal Compute Functions

The table below shows the validation status of signal compute functions in DataLab. It is automatically generated from the source code.

Table 2: Validation status of signal compute functions

Compute function	Description	Test function
<code>compute_XY_mode</code>	Simulate the X-Y mode of an oscilloscope	<code>test_signal_XY_mode</code>
<code>compute_abs</code>	Compute absolute value with <code>numpy.absolute</code>	<code>test_signal_abs</code>
<code>compute_addition</code>	Add <b>dst</b> and <b>src</b> signals and return <b>dst</b> signal modified in place	<code>test_signal_addition</code>
<code>compute_addition_constant</code>	Add <b>dst</b> and a constant value and return a the new result signal object	<code>test_signal_addition_constant</code>
<code>compute_allan_deviation</code>	Compute Allan deviation with <code>cdl.algorithms.signal.allan_deviation()</code>	<code>test_signal_allan_deviation</code>
<code>compute_allan_variance</code>	Compute Allan variance with <code>cdl.algorithms.signal.allan_variance()</code>	<code>test_signal_allan_variance</code>
<code>compute_arithmetic</code>	Perform arithmetic operation on two signals	<code>test_signal_arithmetic</code>
<code>compute_astype</code>	Convert data type with <code>numpy.astype()</code>	<code>test_signal_astype</code>
<code>compute_bandwidth_3db</code>	Compute bandwidth at -3 dB with <code>cdl.algorithms.signal.bandwidth()</code>	<code>test_signal_bandwidth_3db</code>
<code>compute_calibration</code>	Compute linear calibration	<code>test_signal_calibration</code>
<code>compute_cartesian2polar</code>	Convert cartesian coordinates to polar coordinates	<code>test_signal_cartesian2polar</code>
<code>compute_clip</code>	Compute maximum data clipping with <code>numpy.clip()</code>	<code>test_signal_clip</code>
<code>compute_contrast</code>	Compute contrast with <code>cdl.algorithms.signal.contrast()</code>	<code>test_signal_contrast</code>
<code>compute_convolution</code>	Compute convolution of two signals	<code>test_signal_convolution</code>
<code>compute_derivative</code>	Compute derivative with <code>numpy.gradient()</code>	<code>test_signal_derivative</code>
<code>compute_detrending</code>	Detrend data with <code>scipy.signal.detrend()</code>	<code>test_signal_detrending</code>
<code>compute_difference</code>	Compute difference between two signals	<code>test_signal_difference</code>
<code>compute_difference_constant</code>	Subtract a constant value from a signal	<code>test_signal_difference_constant</code>
<code>compute_division</code>	Compute division between two signals	<code>test_signal_division</code>
<code>compute_division_constant</code>	Divide a signal by a constant value	<code>test_signal_division_constant</code>
<code>compute_dynamic_parameters</code>	Compute Dynamic parameters	<code>test_dynamic_parameters</code>
<code>compute_exp</code>	Compute exponential with <code>numpy.exp</code>	<code>test_signal_exp</code>
<code>compute_fft</code>	Compute FFT with <code>cdl.algorithms.signal.fft1d()</code>	<code>test_signal_fft</code>
<code>compute_filter</code>	Compute frequency filter (low-pass, high-pass, band-pass, band-stop)	N/A
<code>compute_full_width_at_y</code>		<code>test_signal_full_width_at_y</code>
<code>compute_fw1e2</code>	Compute FW at $1/e^2$ with <code>cdl.algorithms.signal.fw1e2()</code>	<code>test_signal_fw1e2</code>
<code>compute_fwhm</code>	Compute FWHM with <code>cdl.algorithms.signal.fwhm()</code>	<code>test_signal_fwhm</code>

continues on next page

Table 2 – continued from previous page

Compute function	Description	Test function
<code>compute_gaussian_filter</code>	Compute gaussian filter with <code>scipy.ndimage.gaussian_filter()</code>	<code>test_signal_gaussian_filter</code>
<code>compute_hadamard_variance</code>	Compute Hadamard variance	N/A
<code>compute_histogram</code>	Compute histogram with <code>numpy.histogram()</code>	N/A
<code>compute_iff</code>	Compute iFFT with <code>cdl.algorithms.signal.iff1d()</code>	<code>test_signal_iff</code>
<code>compute_im</code>	Compute imaginary part with <code>numpy.imag()</code>	<code>test_signal_im</code>
<code>compute_integral</code>	Compute integral with <code>scipy.integrate.cumulative_trapezoid()</code>	<code>test_signal_integral</code>
<code>compute_interpolation</code>	Interpolate data with <code>cdl.algorithms.signal.interpolate()</code>	N/A
<code>compute_inverse</code>	Compute inverse with <code>numpy.invert</code>	<code>test_signal_inverse</code>
<code>compute_log10</code>	Compute Log10 with <code>numpy.log10</code>	<code>test_signal_log10</code>
<code>compute_magnitude_spectrum</code>	Compute magnitude spectrum	<code>test_signal_magnitude_spectrum</code>
<code>compute_modified_allan_var</code>	Compute Modified Allan variance	N/A
<code>compute_moving_average</code>	Compute moving average with <code>scipy.ndimage.uniform_filter()</code>	<code>test_signal_moving_average</code>
<code>compute_moving_median</code>	Compute moving median with <code>scipy.ndimage.median_filter()</code>	<code>test_signal_moving_median</code>
<code>compute_normalize</code>	Normalize data with <code>cdl.algorithms.signal.normalize()</code>	<code>test_signal_normalize</code>
<code>compute_offset_correction</code>	Correct offset: subtract the mean value of the signal in the specified range	<code>test_signal_offset_correction</code>
<code>compute_overlapping_allan</code>	Compute Overlapping Allan variance	N/A
<code>compute_peak_detection</code>	Peak detection with <code>cdl.algorithms.signal.peak_indices()</code>	N/A
<code>compute_phase_spectrum</code>	Compute phase spectrum	<code>test_signal_phase_spectrum</code>
<code>compute_polar2cartesian</code>	Convert polar coordinates to cartesian coordinates	<code>test_signal_polar2cartesian</code>
<code>compute_power</code>	Compute power with <code>numpy.power</code>	<code>test_signal_power</code>
<code>compute_product</code>	Multiply <b>dst</b> and <b>src</b> signals and return <b>dst</b> signal modified in place	<code>test_signal_product</code>
<code>compute_product_constant</code>	Multiply <b>dst</b> by a constant value and return the new result signal object	<code>test_signal_product_constant</code>
<code>compute_psd</code>	Compute power spectral density	<code>test_signal_psd</code>
<code>compute_quadratic_differer</code>	Compute quadratic difference between two signals	<code>test_signal_quadratic_difference</code>
<code>compute_re</code>	Compute real part with <code>numpy.real()</code>	<code>test_signal_re</code>
<code>compute_resampling</code>	Resample data with <code>cdl.algorithms.signal.interpolate()</code>	<code>test_signal_resampling</code>
<code>compute_reverse_x</code>	Reverse x-axis	<code>test_signal_reverse_x</code>
<code>compute_sampling_rate_peri</code>	Compute sampling rate and period	<code>test_signal_sampling_rate_period</code>
<code>compute_sqrt</code>	Compute square root with <code>numpy.sqrt</code>	<code>test_signal_sqrt</code>
<code>compute_stats</code>	Compute statistics on a signal	<code>test_signal_stats_unit</code>
<code>compute_swap_axes</code>	Swap axes	<code>test_signal_swap_axes</code>
<code>compute_time_deviation</code>	Compute Time Deviation (TDEV)	N/A
<code>compute_total_variance</code>	Compute Total variance	N/A

continues on next page

Table 2 – continued from previous page

Compute function	Description	Test function
<code>compute_wiener</code>	Compute Wiener filter with <code>scipy.signal.wiener()</code>	<code>test_signal_wiener</code>
<code>compute_windowing</code>	Compute windowing (available methods: hamming, hanning, bartlett, blackman)	N/A
<code>compute_x_at_minmax</code>		<code>test_signal_x_at_minmax</code>
<code>compute_x_at_y</code>		<code>test_signal_x_at_y</code>
<code>compute_y_at_x</code>		<code>test_signal_y_at_x</code>
<code>compute_zero_padding</code>	Compute zero padding with <code>cdl.algorithms.signal.zero_padding()</code>	<code>test_signal_zero_padding</code>

## Image Compute Functions

The table below shows the validation status of image compute functions in DataLab. It is automatically generated from the source code.

Table 3: Validation status of image compute functions

Compute function	Description	Test function
<code>compute_abs</code>	Compute absolute value with <code>numpy.absolute</code>	<code>test_image_abs</code>
<code>compute_addition</code>	Add <b>dst</b> and <b>src</b> images and return <b>dst</b> image modified in place	<code>test_image_addition</code>
<code>compute_addition_constant</code>	Add <b>dst</b> and a constant value and return the new result image object	<code>test_image_addition_constant</code>
<code>compute_arithmetic</code>	Compute arithmetic operation on two images	<code>test_image_arithmetic</code>
<code>compute_astype</code>	Convert image data type with <code>cdl.algorithms.datatypes.clip_astype()</code>	<code>test_image_astype</code>
<code>compute_average_profile</code>	Compute horizontal or vertical average profile	<code>test_average_profile</code>
<code>compute_binning</code>	Binning function on data with <code>cdl.algorithms.image.binning()</code>	<code>test_binning</code>
<code>compute_butterworth</code>	Compute Butterworth filter with <code>skimage.filters.butterworth()</code>	<code>test_butterworth</code>
<code>compute_calibration</code>	Compute linear calibration	<code>test_image_calibration</code>
<code>compute_centroid</code>	Compute centroid	<code>test_image_centroid</code>
<code>compute_clip</code>	Apply clipping with <code>numpy.clip()</code>	<code>test_image_clip</code>
<code>compute_difference</code>	Compute difference between two images	<code>test_image_difference</code>
<code>compute_difference_constant</code>	Subtract a constant value from an image and return the new result image object	<code>test_image_difference_constant</code>
<code>compute_division</code>	Compute division between two images	<code>test_image_division</code>
<code>compute_division_constant</code>	Divide an image by a constant value and return the new result image object	<code>test_image_division_constant</code>
<code>compute_enclosing_circle</code>	Compute minimum enclosing circle	N/A
<code>compute_exp</code>	Compute exponential with <code>numpy.exp</code>	<code>test_image_exp</code>
<code>compute_fft</code>	Compute FFT with <code>cdl.algorithms.image.fft2d()</code>	<code>test_image_fft</code>

continues on next page



Table 3 – continued from previous page

Compute function	Description	Test function
<code>compute_flatfield</code>	Compute flat field correction with <code>cdl.algorithms.image.flatfield()</code>	N/A
<code>compute_fliph</code>	Flip data horizontally with <code>numpy.fliplr()</code>	<code>test_image_fliph</code>
<code>compute_flipv</code>	Flip data vertically with <code>numpy.flipud()</code>	<code>test_image_flipv</code>
<code>compute_gaussian_filter</code>	Compute gaussian filter with <code>scipy.ndimage.gaussian_filter()</code>	<code>test_image_gaussian_filter</code>
<code>compute_histogram</code>	Compute histogram of the image data, with <code>numpy.histogram()</code>	N/A
<code>compute_hough_circle_peaks</code>	Compute Hough circles	N/A
<code>compute_iff</code>	Compute inverse FFT with <code>cdl.algorithms.image.iff2d()</code>	<code>test_image_iff</code>
<code>compute_im</code>	Compute imaginary part with <code>numpy.imag()</code>	<code>test_image_im</code>
<code>compute_inverse</code>	Compute the inverse of an image and return the new result image object	<code>test_image_inverse</code>
<code>compute_line_profile</code>	Compute horizontal or vertical profile	<code>test_line_profile</code>
<code>compute_log10</code>	Compute log10 with <code>numpy.log10</code>	<code>test_image_log10</code>
<code>compute_logp1</code>	Compute log10(z+n) with <code>numpy.log10</code>	<code>test_image_logp1</code>
<code>compute_magnitude_spectrum</code>	Compute magnitude spectrum	<code>test_image_magnitude_spectrum</code>
<code>compute_moving_average</code>	Compute moving average with <code>scipy.ndimage.uniform_filter()</code>	<code>test_image_moving_average</code>
<code>compute_moving_median</code>	Compute moving median with <code>scipy.ndimage.median_filter()</code>	<code>test_image_moving_median</code>
<code>compute_normalize</code>		<code>test_image_normalize</code>
<code>compute_offset_correction</code>	Apply offset correction	<code>test_image_offset_correction</code>
<code>compute_phase_spectrum</code>	Compute phase spectrum	<code>test_image_phase_spectrum</code>
<code>compute_product</code>	Multiply <b>dst</b> and <b>src</b> images and return <b>dst</b> image modified in place	<code>test_image_product</code>
<code>compute_product_constant</code>	Multiply <b>dst</b> by a constant value and return the new result image object	<code>test_image_product_constant</code>
<code>compute_psd</code>	Compute power spectral density	<code>test_image_psd</code>
<code>compute_quadratic_differer</code>	Compute quadratic difference between two images	<code>test_image_quadratic_difference</code>
<code>compute_radial_profile</code>	Compute radial profile around the centroid	N/A
<code>compute_re</code>	Compute real part with <code>numpy.real()</code>	<code>test_image_re</code>
<code>compute_resize</code>	Zooming function with <code>scipy.ndimage.zoom()</code>	N/A
<code>compute_rotate</code>	Rotate data with <code>scipy.ndimage.rotate()</code>	<code>test_image_rotate</code>
<code>compute_rotate270</code>	Rotate data 270° with <code>numpy.rot90()</code>	<code>test_image_rotate270</code>
<code>compute_rotate90</code>	Rotate data 90° with <code>numpy.rot90()</code>	<code>test_image_rotate90</code>
<code>compute_segment_profile</code>	Compute segment profile	<code>test_segment_profile</code>
<code>compute_stats</code>	Compute statistics on an image	<code>test_image_stats_unit</code>
<code>compute_swap_axes</code>	Swap image axes with <code>numpy.transpose()</code>	<code>test_image_swap_axes</code>
<code>compute_wiener</code>	Compute Wiener filter with <code>scipy.signal.wiener()</code>	<code>test_image_wiener</code>
<code>compute_zero_padding</code>		<code>test_image_zero_padding</code>

continues on next page



Table 3 – continued from previous page

Compute function	Description	Test function
<code>compute_blob_dog</code>	Compute blobs using Difference of Gaussian method	N/A
<code>compute_blob_doh</code>	Compute blobs using Determinant of Hessian method	N/A
<code>compute_blob_log</code>	Compute blobs using Laplacian of Gaussian method	N/A
<code>compute_blob_opencv</code>	Compute blobs using OpenCV	N/A
<code>compute_contour_shape</code>	Compute contour shape fit	N/A
<code>compute_peak_detection</code>	Compute 2D peak detection	N/A
<code>compute_canny</code>	Compute Canny filter with <code>skimage.feature.canny()</code>	<code>test_canny</code>
<code>compute_farid</code>	Compute Farid filter with <code>skimage.filters.farid()</code>	<code>test_farid</code>
<code>compute_farid_h</code>	Compute horizontal Farid filter with <code>skimage.filters.farid_h()</code>	<code>test_farid_h</code>
<code>compute_farid_v</code>	Compute vertical Farid filter with <code>skimage.filters.farid_v()</code>	<code>test_farid_v</code>
<code>compute_laplace</code>	Compute Laplace filter with <code>skimage.filters.laplace()</code>	<code>test_laplace</code>
<code>compute_prewitt</code>	Compute Prewitt filter with <code>skimage.filters.prewitt()</code>	<code>test_prewitt</code>
<code>compute_prewitt_h</code>	Compute horizontal Prewitt filter with <code>skimage.filters.prewitt_h()</code>	<code>test_prewitt_h</code>
<code>compute_prewitt_v</code>	Compute vertical Prewitt filter with <code>skimage.filters.prewitt_v()</code>	<code>test_prewitt_v</code>
<code>compute_roberts</code>	Compute Roberts filter with <code>skimage.filters.roberts()</code>	<code>test_roberts</code>
<code>compute_scharr</code>	Compute Scharr filter with <code>skimage.filters.scharr()</code>	<code>test_scharr</code>
<code>compute_scharr_h</code>	Compute horizontal Scharr filter with <code>skimage.filters.scharr_h()</code>	<code>test_scharr_h</code>
<code>compute_scharr_v</code>	Compute vertical Scharr filter with <code>skimage.filters.scharr_v()</code>	<code>test_scharr_v</code>
<code>compute_sobel</code>	Compute Sobel filter with <code>skimage.filters.sobel()</code>	<code>test_sobel</code>
<code>compute_sobel_h</code>	Compute horizontal Sobel filter with <code>skimage.filters.sobel_h()</code>	<code>test_sobel_h</code>
<code>compute_sobel_v</code>	Compute vertical Sobel filter with <code>skimage.filters.sobel_v()</code>	<code>test_sobel_v</code>
<code>compute_adjust_gamma</code>	Gamma correction with <code>skimage.exposure.adjust_gamma()</code>	<code>test_adjust_gamma</code>
<code>compute_adjust_log</code>	Compute log correction with <code>skimage.exposure.adjust_log()</code>	<code>test_adjust_log</code>
<code>compute_adjust_sigmoid</code>	Compute sigmoid correction with <code>skimage.exposure.adjust_sigmoid()</code>	<code>test_adjust_sigmoid</code>
<code>compute_equalize_adapthist</code>	Adaptive histogram equalization	<code>test_equalize_adapthist</code>
<code>compute_equalize_hist</code>	Histogram equalization with <code>skimage.exposure.equalize_hist()</code>	<code>test_equalize_hist</code>
<code>compute_rescale_intensity</code>	Rescale image intensity levels	<code>test_rescale_intensity</code>

continues on next page

Table 3 – continued from previous page

Compute function	Description	Test function
<code>compute_black_tophat</code>	Compute Black Top-Hat with <code>skimage.morphology.black_tophat()</code>	<code>test_black_tophat</code>
<code>compute_closing</code>	Compute morphological closing with <code>skimage.morphology.closing()</code>	<code>test_closing</code>
<code>compute_dilation</code>	Compute Dilation with <code>skimage.morphology.dilation()</code>	<code>test_dilation</code>
<code>compute_erosion</code>	Compute Erosion with <code>skimage.morphology.erosion()</code>	<code>test_erosion</code>
<code>compute_opening</code>	Compute morphological opening with <code>skimage.morphology.opening()</code>	<code>test_opening</code>
<code>compute_white_tophat</code>	Compute White Top-Hat with <code>skimage.morphology.white_tophat()</code>	<code>test_white_tophat</code>
<code>compute_denoise_bilateral</code>	Compute bilateral filter denoising	<code>test_denoise_bilateral</code>
<code>compute_denoise_tophat</code>	Denoise using White Top-Hat	<code>test_denoise_tophat</code>
<code>compute_denoise_tv</code>	Compute Total Variation denoising	<code>test_denoise_tv</code>
<code>compute_denoise_wavelet</code>	Compute Wavelet denoising	<code>test_denoise_wavelet</code>
<code>compute_threshold</code>	Compute the threshold, using one of the available algorithms	<code>test_threshold</code>
<code>compute_threshold_isodata</code>	Compute the threshold using the Isodata algorithm with default parameters	<code>test_threshold_isodata</code>
<code>compute_threshold_li</code>	Compute the threshold using the Li algorithm with default parameters	<code>test_threshold_li</code>
<code>compute_threshold_mean</code>	Compute the threshold using the Mean algorithm	<code>test_threshold_mean</code>
<code>compute_threshold_minimum</code>	Compute the threshold using the Minimum algorithm with default parameters	<code>test_threshold_minimum</code>
<code>compute_threshold_otsu</code>	Compute the threshold using the Otsu algorithm with default parameters	<code>test_threshold_otsu</code>
<code>compute_threshold_triangle</code>	Compute the threshold using the Triangle algorithm with default parameters	<code>test_threshold_triangle</code>
<code>compute_threshold_yen</code>	Compute the threshold using the Yen algorithm with default parameters	<code>test_threshold_yen</code>

## 2.2 General features

This section describes the general features of DataLab, which concern both the signal and image processing panels.

### 2.2.1 Workspace

#### Basic concepts

Working with DataLab is very easy. The user interface is intuitive and self-explanatory. The main window is divided into two main areas:

- The left area shows the list of data sets which are currently loaded in DataLab, distributed over two tabs: **Signals** and **Images**. The user can switch between the two tabs by clicking on the corresponding tab: this switches the main window to the corresponding panel, as well as the menu and toolbar contents. Below the list of data sets, a **Properties** view shows information about the currently selected data set.

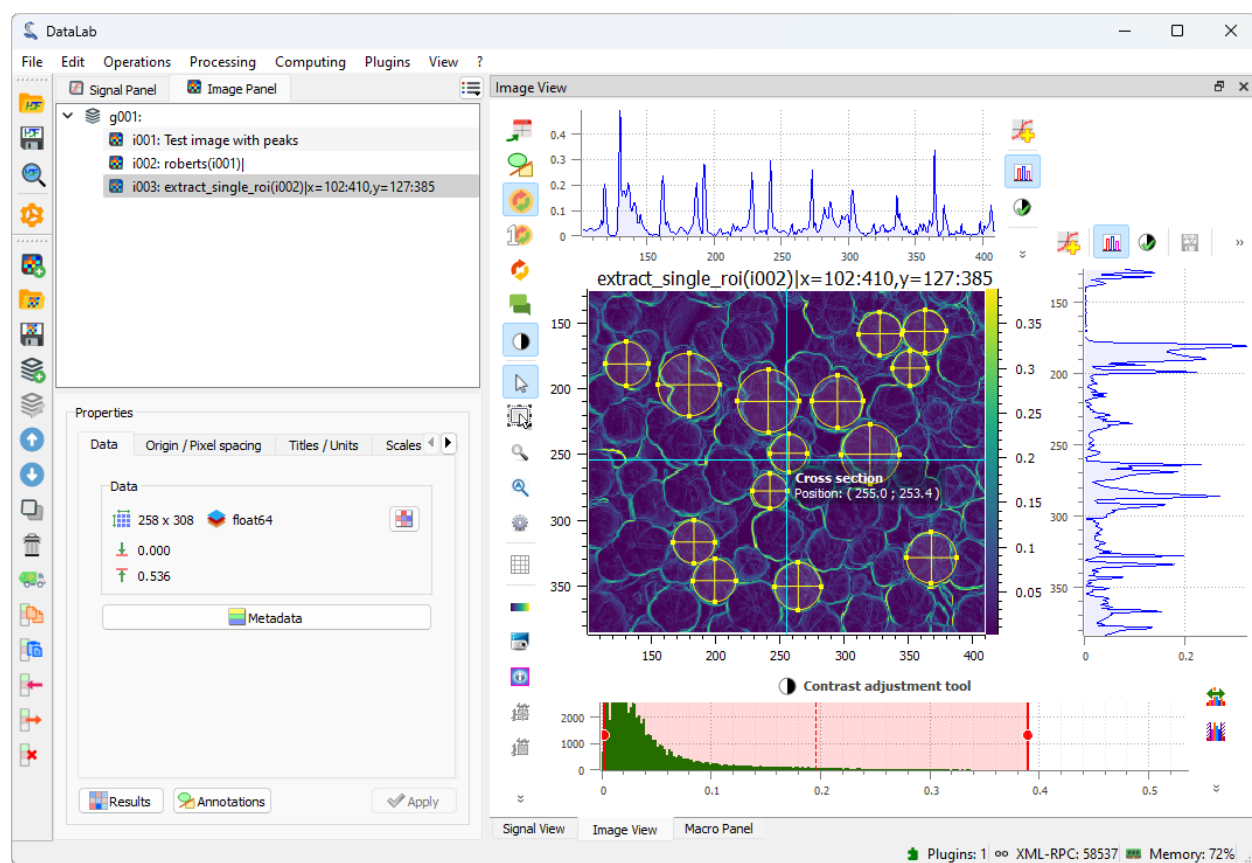


Fig. 1: DataLab main window

- The right area shows the visualization of the currently selected data set. The visualization is updated automatically when the user selects a new data set in the list of data sets.

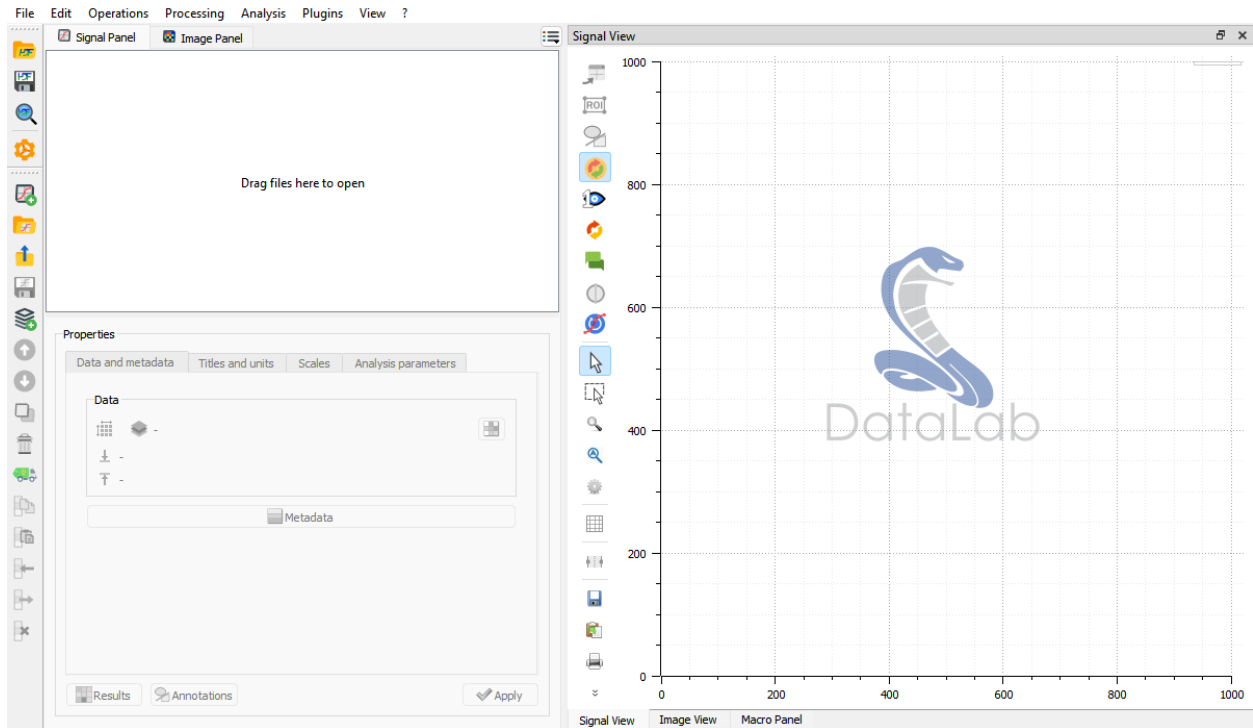


Fig. 2: DataLab main window, at startup.

## Internal data model and workspace

DataLab has its own internal data model, in which data sets are organized around a tree structure. Each panel in the main window corresponds to a branch of the tree. Each data set shown in the panels corresponds to a leaf of the tree. Inside the data set, the data is organized in an object-oriented way, with a set of attributes and methods. The data model is described in more details in the API section (see [cdl.obj](#)).

For each data set (1D signal or 2D image), not only the data itself is stored, but also a set of metadata, which describes the data or the way it has to be displayed. The metadata is stored in a dictionary, which is accessible through the metadata attribute of the data set (and may also be browsed in the **Properties** view, with the **Metadata** button).

The DataLab **Workspace** is defined as the collection of all data sets which are currently loaded in DataLab, in both the **Signals** and **Images** panels.

## Loading and saving the workspace

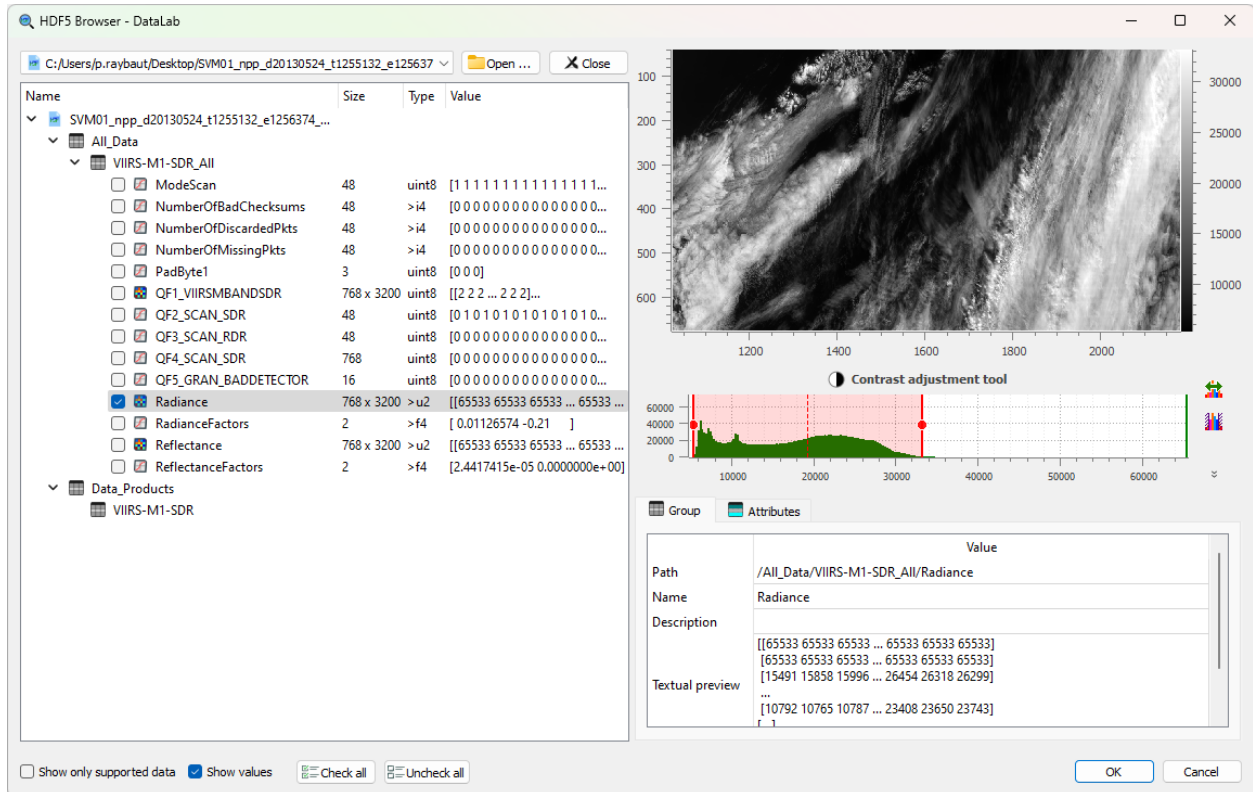
The following actions are available to manage the workspace from the **File** menu:

- **Open HDF5 file:** load a workspace from an HDF5 file.
- **Save to HDF5 file:** save the current workspace to an HDF5 file.
- **Browse HDF5 file:** open the [HDF5 Browser](#) to explore the content of an HDF5 file and import data sets into the workspace.

**Note:** Data sets may also be saved or loaded individually, using data formats such as *.txt* or *.npy* for 1D signals (see [Open signal](#) for the list of supported formats), or *.tiff* or *.dcm* for 2D images (see [Open image](#) for the list of supported formats).

## 2.2.2 HDF5 Browser

The “HDF5 Browser” is a modal dialog box allowing to import almost any 1D and 2D data into DataLab workspace (and eventually metadata).



Compatible curve or image data are displayed in a hierarchical view on the left panel, as well as other scalar data (scalar values are just shown for context purpose and may not be imported into DataLab workspace).

The right panel displays the selected curve or image data. It also shows information on “Group” (path, description, etc.) and “Attributes” (names and values).

**The HDF5 browser is fairly simple to use:**

- On the left panel, select the curve or image data you want to import
- Selected data is plotted on the right panel
- Click on “Check all” if you want to import all compatible data
- Then validate by clicking on “OK”

**Note:** The HDF5 browser may be used to explore multiple HDF5 files at once, thus allowing to import data from different files into the same DataLab workspace.

## 2.2.3 Macros

### Overview

There are many ways to extend DataLab with new functionality (see [Plugins](#) or [Remote controlling](#)). The easiest way to do so is by using macros. Macros are small Python scripts that can be executed from the “Macro Panel” in DataLab.



Fig. 3: The Macro Panel in DataLab.

Macros can be used to automate repetitive tasks, or to create new functionality. As the plugin and remote control system, macros rely on the DataLab high-level API to interact with the application. This means that you can reuse the same code snippets in macros, plugins, and remote control scripts.

**Warning:** DataLab handles macros as Python scripts. This means that you can use the full power of Python to create your macros. Even though this is a powerful feature, it also means that you should be careful when running macros from unknown sources, as they can potentially harm your system.

**See also:**

The DataLab high-level API is documented in the [API](#) section. The plugin system is documented in the [Plugins](#) section, and the remote control system is documented in the [Remote controlling](#) section.

**Main features**

The Macro Panel is a simple interface to:

- Create new macros, using the “New macro” button.
- Rename existing macros, using the “Rename macro” button.
- Import/export macros from/to files, using the “Import macro” and “Export macro” buttons.
- Execute macros, using the “Run macro” button.
- Stop the execution of a macro, using the “Stop macro” button.

Macros are embedded in the DataLab workspace, so they are saved together with the rest of the data (i.e. with signals and images) when exporting the workspace to a HDF5 file. This means that you can share your macros with other users simply by sharing the workspace file.

---

**Note:** Macro are executed in a separate process, so they won’t block the main DataLab application. This means that you can continue working with DataLab while a macro is running and that *you can stop a macro at any time* using the button.

---

**Example**

For a detailed example of how to create a macro, see the [Prototyping a custom processing pipeline](#) tutorial.

**2.2.4 Remote controlling**

DataLab may be controlled remotely using the [XML-RPC](#) protocol which is natively supported by Python (and many other languages). Remote controlling allows to access DataLab main features from a separate process.

---

**Note:** If you are looking for a lightweight alternative solution to remote control DataLab (i.e. without having to install the whole DataLab package and its dependencies on your environment), please have a look at the [DataLab Simple Client](#) package (*pip install cdlclient*).

---

**From an IDE**

DataLab may be controlled remotely from an IDE (e.g. [Spyder](#) or any other IDE, or even a Jupyter Notebook) that runs a Python script. It allows to connect to a running DataLab instance, adds a signal and an image, and then runs calculations. This feature is exposed by the *RemoteProxy* class that is provided in module `cdl.proxy`.

## From a third-party application

DataLab may also be controlled remotely from a third-party application, for the same purpose.

If the third-party application is written in Python 3, it may directly use the *RemoteProxy* class as mentioned above. From another language, it is also achievable, but it requires to implement a XML-RPC client in this language using the same methods of proxy server as in the *RemoteProxy* class.

Data (signals and images) may also be exchanged between DataLab and the remote client application, in both directions.

The remote client application may be written in any language that supports XML-RPC. For example, it is possible to write a remote client application in Python, Java, C++, C#, etc. The remote client application may be a graphical application or a command line application.

The remote client application may be run on the same computer as DataLab or on a different computer. In the latter case, the remote client application must know the IP address of the computer running DataLab.

The remote client application may be run before or after DataLab. In the latter case, the remote client application must try to connect to DataLab until it succeeds.

## Supported features

Supported features are the following:

- Switch to signal or image panel
- Remove all signals and images
- Save current session to a HDF5 file
- Open HDF5 files into current session
- Browse HDF5 file
- Open a signal or an image from file
- Add a signal
- Add an image
- Get object list
- Run calculation with parameters

---

**Note:** The signal and image objects are described on this section: [Internal data model](#).

---

Some examples are provided to help implementing such a communication between your application and DataLab:

- See module: `cdl.tests.remoteclient_app`
- See module: `cdl.tests.remoteclient_unit`



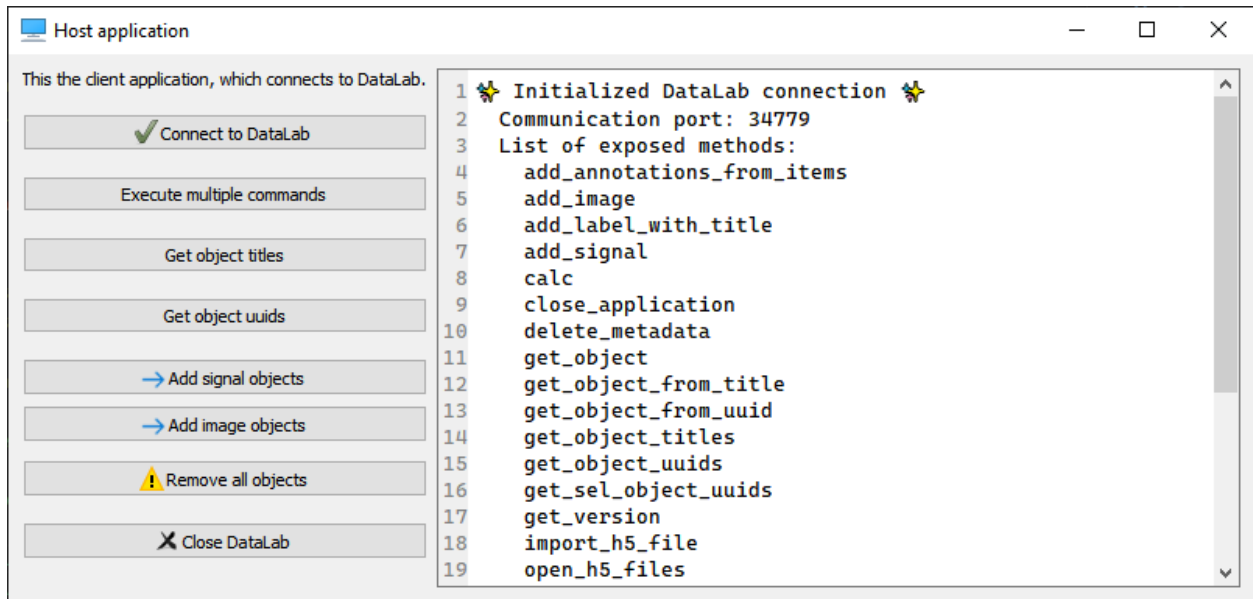


Fig. 4: Screenshot of remote client application test (`cdl.tests.remoteclient_app`)

## Examples

When using Python 3, you may directly use the *RemoteProxy* class as in examples cited above or below.

Here is an example in Python 3 of a script that connects to a running DataLab instance, adds a signal and an image, and then runs calculations (the cell structure of the script make it convenient to be used in *Spyder* IDE):

```

"""
Example of remote control of DataLab current session,
from a Python script running outside DataLab (e.g. in Spyder)

Created on Fri May 12 12:28:56 2023

@author: p.raybaut
"""

# %% Importing necessary modules

# NumPy for numerical array computations:
import numpy as np

# DataLab remote control client:
from cdlclient import SimpleRemoteProxy as RemoteProxy

# %% Connecting to DataLab current session

proxy = RemoteProxy()

# %% Executing commands in DataLab (...)

z = np.random.rand(20, 20)

```

(continues on next page)

(continued from previous page)

```

proxy.add_image("toto", z)

# %% Executing commands in DataLab (...)

proxy.toggle_auto_refresh(False) # Turning off auto-refresh
x = np.array([1.0, 2.0, 3.0])
y = np.array([4.0, 5.0, -1.0])
proxy.add_signal("toto", x, y)

# %% Executing commands in DataLab (...)

proxy.compute_derivative()
proxy.toggle_auto_refresh(True) # Turning on auto-refresh

# %% Executing commands in DataLab (...)

proxy.set_current_panel("image")

# %% Executing a lot of commands without refreshing DataLab

z = np.random.rand(400, 400)
proxy.add_image("foobar", z)
with proxy.context_no_refresh():
    for _idx in range(100):
        proxy.compute_fft()

```

Here is a Python 2.7 reimplementaion of this class:

```

# Copyright (c) DataLab Platform Developers, BSD 3-Clause license, see LICENSE file.

"""
DataLab remote controlling class for Python 2.7
"""

import io
import os
import os.path as osp
import socket
import sys

import ConfigParser as cp
import numpy as np
from guidata.userconfig import get_config_dir
from xmlrpclib import Binary, ServerProxy

def array_to_rpcbinary(data):
    """Convert NumPy array to XML-RPC Binary object, with shape and dtype"""
    dbytes = io.BytesIO()
    np.save(dbytes, data, allow_pickle=False)
    return Binary(dbytes.getvalue())

```

(continues on next page)

(continued from previous page)

```

def get_cdl_xmlrpc_port():
    """Return DataLab current XML-RPC port"""
    if sys.platform == "win32" and "HOME" in os.environ:
        os.environ.pop("HOME") # Avoid getting old WinPython settings dir
    fname = osp.join(get_config_dir(), ".DataLab", "DataLab.ini")
    ini = cp.ConfigParser()
    ini.read(fname)
    try:
        return ini.get("main", "rpc_server_port")
    except (cp.NoSectionError, cp.NoOptionError):
        raise ConnectionRefusedError("DataLab has not yet been executed")

class RemoteClient(object):
    """Object representing a proxy/client to DataLab XML-RPC server"""

    def __init__(self):
        self.port = None
        self.serverproxy = None

    def connect(self, port=None):
        """Connect to DataLab XML-RPC server"""
        if port is None:
            port = get_cdl_xmlrpc_port()
        self.port = port
        url = "http://127.0.0.1:" + port
        self.serverproxy = ServerProxy(url, allow_none=True)
        try:
            self.get_version()
        except socket.error:
            raise ConnectionRefusedError("DataLab is currently not running")

    def get_version(self):
        """Return DataLab public version"""
        return self.serverproxy.get_version()

    def close_application(self):
        """Close DataLab application"""
        self.serverproxy.close_application()

    def raise_window(self):
        """Raise DataLab window"""
        self.serverproxy.raise_window()

    def get_current_panel(self):
        """Return current panel"""
        return self.serverproxy.get_current_panel()

    def set_current_panel(self, panel):
        """Switch to panel"""
        self.serverproxy.set_current_panel(panel)

```

(continues on next page)

(continued from previous page)

```

def reset_all(self):
    """Reset all application data"""
    self.serverproxy.reset_all()

def toggle_auto_refresh(self, state):
    """Toggle auto refresh state"""
    self.serverproxy.toggle_auto_refresh(state)

def toggle_show_titles(self, state):
    """Toggle show titles state"""
    self.serverproxy.toggle_show_titles(state)

def save_to_h5_file(self, filename):
    """Save to a DataLab HDF5 file"""
    self.serverproxy.save_to_h5_file(filename)

def open_h5_files(self, h5files, import_all, reset_all):
    """Open a DataLab HDF5 file or import from any other HDF5 file"""
    self.serverproxy.open_h5_files(h5files, import_all, reset_all)

def import_h5_file(self, filename, reset_all):
    """Open DataLab HDF5 browser to Import HDF5 file"""
    self.serverproxy.import_h5_file(filename, reset_all)

def load_from_files(self, filenames):
    """Open objects from files in current panel (signals/images)"""
    self.serverproxy.load_from_files(filenames)

def add_signal(
    self,
    title,
    xdata,
    ydata,
    xunit=None,
    yunit=None,
    xlabel=None,
    ylabel=None,
    group_id="",
    set_current=True,
):
    """Add signal data to DataLab"""
    xbinary = array_to_rpcbinary(xdata)
    ybinary = array_to_rpcbinary(ydata)
    p = self.serverproxy
    return p.add_signal(
        title, xbinary, ybinary, xunit, yunit, xlabel, ylabel, group_id, set_current
    )

def add_image(
    self,
    title,

```

(continues on next page)

(continued from previous page)

```

        data,
        xunit=None,
        yunit=None,
        zunit=None,
        xlabel=None,
        ylabel=None,
        zlabel=None,
        group_id="",
        set_current=True,
    ):
        """Add image data to DataLab"""
        zbinary = array_to_rpcbinary(data)
        p = self.serverproxy
        return p.add_image(
            title,
            zbinary,
            xunit,
            yunit,
            zunit,
            xlabel,
            ylabel,
            zlabel,
            group_id,
            set_current,
        )

    def get_object_titles(self, panel=None):
        """Get object (signal/image) list for current panel"""
        return self.serverproxy.get_object_titles(panel)

    def get_object(self, nb_id_title=None, panel=None):
        """Get object (signal/image) by number, id or title"""
        return self.serverproxy.get_object(nb_id_title, panel)

    def get_object_uuids(self, panel=None, group=None):
        """Get object (signal/image) list for current panel"""
        return self.serverproxy.get_object_uuids(panel, group)

def test_remote_client():
    """DataLab Remote Client test"""
    cdl = RemoteClient()
    cdl.connect()
    data = np.array([[3, 4, 5], [7, 8, 0]], dtype=np.uint16)
    cdl.add_image("toto", data)

if __name__ == "__main__":
    test_remote_client()

```

## Connection dialog

The DataLab package also provides a connection dialog that may be used to connect to a running DataLab instance. It is exposed by the `cdl.widgets.connection.ConnectionDialog` class.

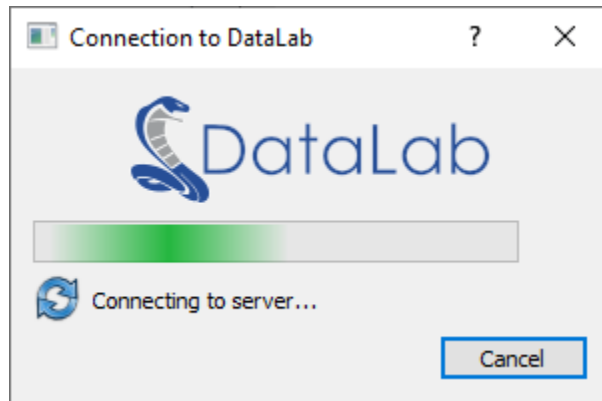


Fig. 5: Screenshot of connection dialog (`cdl.widgets.connection.ConnectionDialog`)

Example of use:

```
# Copyright (c) DataLab Platform Developers, BSD 3-Clause license, see LICENSE file.

"""
DataLab Remote client connection dialog example
"""

# guitest: show,skip

from guidata.qthelpers import qt_app_context
from qtpy import QtWidgets as QW

from cdl.proxy import RemoteProxy
from cdl.widgets.connection import ConnectionDialog

def test_dialog():
    """Test connection dialog"""
    proxy = RemoteProxy(autoconnect=False)
    with qt_app_context():
        dlg = ConnectionDialog(proxy.connect)
        if dlg.exec():
            QW.QMessageBox.information(None, "Connection", "Successfully connected")
        else:
            QW.QMessageBox.critical(None, "Connection", "Connection failed")

if __name__ == "__main__":
    test_dialog()
```

## Public API: remote client

### `class cdl.core.remote.RemoteClient`

Object representing a proxy/client to DataLab XML-RPC server. This object is used to call DataLab functions from a Python script.

### Examples

Here is a simple example of how to use RemoteClient in a Python script or in a Jupyter notebook:

```
>>> from cdl.core.remote import RemoteClient
>>> proxy = RemoteClient()
>>> proxy.connect()
Connecting to DataLab XML-RPC server...OK (port: 28867)
>>> proxy.get_version()
'1.0.0'
>>> proxy.add_signal("toto", np.array([1., 2., 3.]), np.array([4., 5., -1.]))
True
>>> proxy.get_object_titles()
['toto']
>>> proxy["toto"]
<cdl.core.model.signal.SignalObj at 0x7f7f1c0b4a90>
>>> proxy[1]
<cdl.core.model.signal.SignalObj at 0x7f7f1c0b4a90>
>>> proxy[1].data
array([1., 2., 3.]
```

**connect**(*port: str | None = None, timeout: float | None = None, retries: int | None = None*) → *None*

Try to connect to DataLab XML-RPC server.

#### Parameters

- **port** – XML-RPC port to connect to. If not specified, the port is automatically retrieved from DataLab configuration.
- **timeout** – Timeout in seconds. Defaults to 5.0.
- **retries** – Number of retries. Defaults to 10.

#### Raises

- **ConnectionRefusedError** – Unable to connect to DataLab
- **ValueError** – Invalid timeout (must be >= 0.0)
- **ValueError** – Invalid number of retries (must be >= 1)

**disconnect**() → *None*

Disconnect from DataLab XML-RPC server.

**is\_connected**() → *bool*

Return True if connected to DataLab XML-RPC server.

**get\_method\_list**() → *list[str]*

Return list of available methods.

```
add_signal(title: str, xdata: ndarray, ydata: ndarray, xunit: str | None = None, yunit: str | None = None,  
           xlabel: str | None = None, ylabel: str | None = None, group_id: str = "", set_current: bool =  
           True) → bool
```

Add signal data to DataLab.

#### Parameters

- **title** – Signal title
- **xdata** – X data
- **ydata** – Y data
- **xunit** – X unit. Defaults to *None*
- **yunit** – Y unit. Defaults to *None*
- **xlabel** – X label. Defaults to *None*
- **ylabel** – Y label. Defaults to *None*
- **group\_id** – group id in which to add the signal. Defaults to ""
- **set\_current** – if *True*, set the added signal as current

#### Returns

*True* if signal was added successfully, *False* otherwise

#### Raises

- **ValueError** – Invalid xdata dtype
- **ValueError** – Invalid ydata dtype

```
add_image(title: str, data: ndarray, xunit: str | None = None, yunit: str | None = None, zunit: str | None =  
           None, xlabel: str | None = None, ylabel: str | None = None, zlabel: str | None = None, group_id:  
           str = "", set_current: bool = True) → bool
```

Add image data to DataLab.

#### Parameters

- **title** – Image title
- **data** – Image data
- **xunit** – X unit. Defaults to *None*
- **yunit** – Y unit. Defaults to *None*
- **zunit** – Z unit. Defaults to *None*
- **xlabel** – X label. Defaults to *None*
- **ylabel** – Y label. Defaults to *None*
- **zlabel** – Z label. Defaults to *None*
- **group\_id** – group id in which to add the image. Defaults to ""
- **set\_current** – if *True*, set the added image as current

#### Returns

*True* if image was added successfully, *False* otherwise

#### Raises

- **ValueError** – Invalid data dtype



**add\_object**(*obj*: *SignalObj* | *ImageObj*, *group\_id*: *str* = "", *set\_current*: *bool* = *True*) → *None*

Add object to DataLab.

**Parameters**

- **obj** – Signal or image object
- **group\_id** – group id in which to add the object. Defaults to ""
- **set\_current** – if True, set the added object as current

**calc**(*name*: *str*, *param*: *DataSet* | *None* = *None*) → *None*

Call compute function name in current panel's processor.

**Parameters**

- **name** – Compute function name
- **param** – Compute function parameter. Defaults to None.

**Raises**

**ValueError** – unknown function

**get\_object**(*nb\_id\_title*: *int* | *str* | *None* = *None*, *panel*: *str* | *None* = *None*) → *SignalObj* | *ImageObj*

Get object (signal/image) from index.

**Parameters**

- **nb\_id\_title** – Object number, or object id, or object title. Defaults to None (current object).
- **panel** – Panel name. Defaults to None (current panel).

**Returns**

Object

**Raises**

**KeyError** – if object not found

**get\_object\_shapes**(*nb\_id\_title*: *int* | *str* | *None* = *None*, *panel*: *str* | *None* = *None*) → *list*

Get plot item shapes associated to object (signal/image).

**Parameters**

- **nb\_id\_title** – Object number, or object id, or object title. Defaults to None (current object).
- **panel** – Panel name. Defaults to None (current panel).

**Returns**

List of plot item shapes

**add\_annotations\_from\_items**(*items*: *list*, *refresh\_plot*: *bool* = *True*, *panel*: *str* | *None* = *None*) → *None*

Add object annotations (annotation plot items).

**Parameters**

- **items** – annotation plot items
- **refresh\_plot** – refresh plot. Defaults to True.
- **panel** – panel name (valid values: "signal", "image"). If None, current panel is used.

**add\_group**(title: *str*, panel: *str* | *None* = *None*, select: *bool* = *False*) → *None*

Add group to DataLab.

**Parameters**

- **title** – Group title
- **panel** – Panel name (valid values: “signal”, “image”). Defaults to *None*.
- **select** – Select the group after creation. Defaults to *False*.

**add\_label\_with\_title**(title: *str* | *None* = *None*, panel: *str* | *None* = *None*) → *None*

Add a label with object title on the associated plot

**Parameters**

- **title** – Label title. Defaults to *None*. If *None*, the title is the object title.
- **panel** – panel name (valid values: “signal”, “image”). If *None*, current panel is used.

**close\_application**() → *None*

Close DataLab application

**context\_no\_refresh**() → *Generator*[*None*, *None*, *None*]

Return a context manager to temporarily disable auto refresh.

**Returns**

Context manager

**Example**

```
>>> with proxy.context_no_refresh():
...     proxy.add_image("image1", data1)
...     proxy.compute_fft()
...     proxy.compute_wiener()
...     proxy.compute_ifft()
...     # Auto refresh is disabled during the above operations
```

**delete\_metadata**(refresh\_plot: *bool* = *True*, keep\_roi: *bool* = *False*) → *None*

Delete metadata of selected objects

**Parameters**

- **refresh\_plot** – Refresh plot. Defaults to *True*.
- **keep\_roi** – Keep ROI. Defaults to *False*.

**get\_current\_panel**() → *str*

Return current panel name.

**Returns**

“signal”, “image”, “macro”))

**Return type**

Panel name (valid values

**get\_group\_titles\_with\_object\_infos**() → *tuple*[*list*[*str*], *list*[*list*[*str*]], *list*[*list*[*str*]]]

Return groups titles and lists of inner objects uuids and titles.

**Returns**

groups titles, lists of inner objects uuids and titles

**Return type**

Tuple

**get\_object\_titles**(*panel*: *str* | *None* = *None*) → *list*[*str*]

Get object (signal/image) list for current panel. Objects are sorted by group number and object index in group.

**Parameters**

**panel** – panel name (valid values: “signal”, “image”, “macro”). If *None*, current data panel is used (i.e. signal or image panel).

**Returns**

List of object titles

**Raises****ValueError** – if panel not found**get\_object\_uuids**(*panel*: *str* | *None* = *None*, *group*: *int* | *str* | *None* = *None*) → *list*[*str*]

Get object (signal/image) uuid list for current panel. Objects are sorted by group number and object index in group.

**Parameters**

- **panel** – panel name (valid values: “signal”, “image”). If *None*, current panel is used.
- **group** – Group number, or group id, or group title. Defaults to *None* (all groups).

**Returns**

List of object uuids

**Raises****ValueError** – if panel not found**classmethod** **get\_public\_methods**() → *list*[*str*]

Return all public methods of the class, except itself.

**Returns**

List of public methods

**get\_sel\_object\_uuids**(*include\_groups*: *bool* = *False*) → *list*[*str*]

Return selected objects uuids.

**Parameters**

**include\_groups** – If *True*, also return objects from selected groups.

**Returns**

List of selected objects uuids.

**get\_version**() → *str*

Return DataLab public version.

**Returns**

DataLab version

**import\_h5\_file**(*filename*: *str*, *reset\_all*: *bool* | *None* = *None*) → *None*

Open DataLab HDF5 browser to Import HDF5 file.

**Parameters**

- **filename** – HDF5 file name
- **reset\_all** – Reset all application data. Defaults to *None*.

**import\_macro\_from\_file**(filename: *str*) → *None*

Import macro from file

**Parameters**

**filename** – Filename.

**load\_from\_directory**(path: *str*) → *None*

Open objects from directory in current panel (signals/images).

**Parameters**

**path** – directory path

**load\_from\_files**(filenames: *list[str]*) → *None*

Open objects from files in current panel (signals/images).

**Parameters**

**filenames** – list of file names

**open\_h5\_files**(h5files: *list[str] | None = None*, import\_all: *bool | None = None*, reset\_all: *bool | None = None*) → *None*

Open a DataLab HDF5 file or import from any other HDF5 file.

**Parameters**

- **h5files** – List of HDF5 files to open. Defaults to *None*.
- **import\_all** – Import all objects from HDF5 files. Defaults to *None*.
- **reset\_all** – Reset all application data. Defaults to *None*.

**raise\_window**() → *None*

Raise DataLab window

**reset\_all**() → *None*

Reset all application data

**run\_macro**(number\_or\_title: *int | str | None = None*) → *None*

Run macro.

**Parameters**

**number\_or\_title** – Macro number, or macro title. Defaults to *None* (current macro).

**Raises**

**ValueError** – if macro not found

**save\_to\_h5\_file**(filename: *str*) → *None*

Save to a DataLab HDF5 file.

**Parameters**

**filename** – HDF5 file name

**select\_groups**(selection: *list[int | str] | None = None*, panel: *str | None = None*) → *None*

Select groups in current panel.

**Parameters**

- **selection** – List of group numbers (1 to N), or list of group uuids, or *None* to select all groups. Defaults to *None*.
- **panel** – panel name (valid values: “signal”, “image”). If *None*, current panel is used. Defaults to *None*.

**select\_objects**(*selection*: *list[int | str]*, *panel*: *str | None = None*) → *None*

Select objects in current panel.

**Parameters**

- **selection** – List of object numbers (1 to N) or uuids to select
- **panel** – panel name (valid values: “signal”, “image”). If None, current panel is used. Defaults to None.

**set\_current\_panel**(*panel*: *str*) → *None*

Switch to panel.

**Parameters**

**panel** – Panel name (valid values: “signal”, “image”, “macro”)

**stop\_macro**(*number\_or\_title*: *int | str | None = None*) → *None*

Stop macro.

**Parameters**

**number\_or\_title** – Macro number, or macro title. Defaults to None (current macro).

**Raises**

**ValueError** – if macro not found

**toggle\_auto\_refresh**(*state*: *bool*) → *None*

Toggle auto refresh state.

**Parameters**

**state** – Auto refresh state

**toggle\_show\_titles**(*state*: *bool*) → *None*

Toggle show titles state.

**Parameters**

**state** – Show titles state

## Public API: additional methods

The remote control class methods (either using the proxy or the remote client) may be completed with additional methods which are dynamically added at runtime. This mechanism allows to access the methods of the processors of DataLab (see [cdl.core.gui.processor](#)).

## Signal processor

When working with signals, the methods of [cdl.core.gui.processor.signal.SignalProcessor](#) may be accessed.

**class** [cdl.core.gui.processor.signal.SignalProcessor](#)(*panel*: [SignalPanel](#) | [ImagePanel](#), *plotwidget*: [PlotWidget](#))

Object handling signal processing: operations, processing, analysis

**compute\_sum**() → *None*

Compute sum with [cdl.computation.signal.compute\\_addition\(\)](#)

**compute\_addition\_constant**(*param*: [ConstantParam](#) | *None = None*) → *None*

Compute sum with a constant with [cdl.computation.signal.compute\\_addition\\_constant\(\)](#)

**compute\_average()** → None

Compute average with `cdl.computation.signal.compute_addition()` and divide by the number of signals

**compute\_product()** → None

Compute product with `cdl.computation.signal.compute_product()`

**compute\_product\_constant**(*param*: ConstantParam | None = None) → None

Compute product with a constant with `cdl.computation.signal.compute_product_constant()`

**compute\_swap\_axes()** → None

Swap data axes with `cdl.computation.signal.compute_swap_axes()`

**compute\_inverse()** → None

Compute inverse

**compute\_abs()** → None

Compute absolute value with `cdl.computation.signal.compute_abs()`

**compute\_re()** → None

Compute real part with `cdl.computation.signal.compute_re()`

**compute\_im()** → None

Compute imaginary part with `cdl.computation.signal.compute_im()`

**compute\_astype**(*param*: DataTypeSParam | None = None) → None

Convert data type with `cdl.computation.signal.compute_astype()`

**compute\_log10()** → None

Compute Log10 with `cdl.computation.signal.compute_log10()`

**compute\_exp()** → None

Compute Log10 with `cdl.computation.signal.compute_exp()`

**compute\_sqrt()** → None

Compute square root with `cdl.computation.signal.compute_sqrt()`

**compute\_power**(*param*: PowerParam | None = None) → None

Compute power with `cdl.computation.signal.compute_power()`

**compute\_arithmetic**(*obj2*: SignalObj | None = None, *param*: ArithmeticParam | None = None) → None

Compute arithmetic operation between two signals with `cdl.computation.signal.compute_arithmetic()`

**compute\_difference**(*obj2*: SignalObj | list[SignalObj] | None = None) → None

Compute difference between two signals with `cdl.computation.signal.compute_difference()`

**compute\_difference\_constant**(*param*: ConstantParam | None = None) → None

Compute difference with a constant with `cdl.computation.signal.compute_difference_constant()`

**compute\_quadratic\_difference**(*obj2*: SignalObj | list[SignalObj] | None = None) → None

Compute quadratic difference between two signals with `cdl.computation.signal.compute_quadratic_difference()`

**compute\_division**(*obj2*: SignalObj | list[SignalObj] | None = None) → None

Compute division between two signals with `cdl.computation.signal.compute_division()`

**compute\_division\_constant**(*param*: ConstantParam | None = None) → None  
 Compute division by a constant with `cdl.computation.signal.compute_division_constant()`

**compute\_peak\_detection**(*param*: PeakDetectionParam | None = None) → None  
 Detect peaks from data with `cdl.computation.signal.compute_peak_detection()`

**compute\_reverse\_x**() → None  
 Reverse X axis with `cdl.computation.signal.compute_reverse_x()`

**compute\_cartesian2polar**(*param*: AngleUnitParam | None = None) → None  
 Convert cartesian to polar coordinates with `cdl.computation.signal.compute_cartesian2polar()`

**compute\_polar2cartesian**(*param*: AngleUnitParam | None = None) → None  
 Convert polar to cartesian coordinates with `cdl.computation.signal.compute_polar2cartesian()`.

**compute\_normalize**(*param*: NormalizeParam | None = None) → None  
 Normalize data with `cdl.computation.signal.compute_normalize()`

**compute\_derivative**() → None  
 Compute derivative with `cdl.computation.signal.compute_derivative()`

**compute\_integral**() → None  
 Compute integral with `cdl.computation.signal.compute_integral()`

**compute\_calibration**(*param*: XYCalibrateParam | None = None) → None  
 Compute data linear calibration with `cdl.computation.signal.compute_calibration()`

**compute\_clip**(*param*: ClipParam | None = None) → None  
 Compute maximum data clipping with `cdl.computation.signal.compute_clip()`

**compute\_offset\_correction**(*param*: ROIIDParam | None = None) → None  
 Compute offset correction with `cdl.computation.signal.compute_offset_correction()`

**compute\_gaussian\_filter**(*param*: GaussianParam | None = None) → None  
 Compute gaussian filter with `cdl.computation.signal.compute_gaussian_filter()`

**compute\_moving\_average**(*param*: MovingAverageParam | None = None) → None  
 Compute moving average with `cdl.computation.signal.compute_moving_average()`

**compute\_moving\_median**(*param*: MovingMedianParam | None = None) → None  
 Compute moving median with `cdl.computation.signal.compute_moving_median()`

**compute\_wiener**() → None  
 Compute Wiener filter with `cdl.computation.signal.compute_wiener()`

**compute\_lowpass**(*param*: LowPassFilterParam | None = None) → None  
 Compute high-pass filter with `cdl.computation.signal.compute_filter()`

**compute\_highpass**(*param*: HighPassFilterParam | None = None) → None  
 Compute high-pass filter with `cdl.computation.signal.compute_filter()`

**compute\_bandpass**(*param*: BandPassFilterParam | None = None) → None  
 Compute band-pass filter with `cdl.computation.signal.compute_filter()`

**compute\_bandstop**(*param*: BandStopFilterParam | None = None) → None  
 Compute band-stop filter with `cdl.computation.signal.compute_filter()`

**compute\_zero\_padding**(*param*: ZeroPadding1DParam | *None* = *None*) → *None*  
Compute zero padding with `cdl.computation.signal.compute_zero_padding()`

**compute\_fft**(*param*: FFTParam | *None* = *None*) → *None*  
Compute FFT with `cdl.computation.signal.compute_fft()`

**compute\_ifft**(*param*: FFTParam | *None* = *None*) → *None*  
Compute iFFT with `cdl.computation.signal.compute_ifft()`

**compute\_magnitude\_spectrum**(*param*: SpectrumParam | *None* = *None*) → *None*  
Compute magnitude spectrum with `cdl.computation.signal.compute_magnitude_spectrum()`

**compute\_phase\_spectrum**() → *None*  
Compute phase spectrum with `cdl.computation.signal.compute_phase_spectrum()`

**compute\_psd**(*param*: SpectrumParam | *None* = *None*) → *None*  
Compute power spectral density with `cdl.computation.signal.compute_psd()`

**compute\_interpolation**(*obj2*: SignalObj | *None* = *None*, *param*: InterpolationParam | *None* = *None*)  
Compute interpolation with `cdl.computation.signal.compute_interpolation()`

**compute\_resampling**(*param*: ResamplingParam | *None* = *None*)  
Compute resampling with `cdl.computation.signal.compute_resampling()`

**compute\_detrending**(*param*: DetrendingParam | *None* = *None*)  
Compute detrending with `cdl.computation.signal.compute_detrending()`

**compute\_XY\_mode**(*obj2*: SignalObj | *None* = *None*) → *None*  
Compute XY mode with `cdl.computation.signal.compute_XY_mode.()`

**compute\_convolution**(*obj2*: SignalObj | *None* = *None*) → *None*  
Compute convolution with `cdl.computation.signal.compute_convolution()`

**compute\_windowing**(*param*: WindowingParam | *None* = *None*) → *None*  
Compute windowing with `cdl.computation.signal.compute_windowing()`

**compute\_allan\_variance**(*param*: AllanVarianceParam | *None* = *None*) → *None*  
Compute Allan variance with `cdl.computation.signal.compute_allan_variance()`

**compute\_allan\_deviation**(*param*: AllanVarianceParam | *None* = *None*) → *None*  
Compute Allan deviation with `cdl.computation.signal.compute_allan_deviation()`

**compute\_overlapping\_allan\_variance**(*param*: AllanVarianceParam | *None* = *None*) → *None*  
Compute overlapping Allan variance with `cdl.computation.signal.compute_overlapping_allan_variance()`

**compute\_modified\_allan\_variance**(*param*: AllanVarianceParam | *None* = *None*) → *None*  
Compute modified Allan variance with `cdl.computation.signal.compute_modified_allan_variance()`

**compute\_hadamard\_variance**(*param*: AllanVarianceParam | *None* = *None*) → *None*  
Compute Hadamard variance with `cdl.computation.signal.compute_hadamard_variance()`

**compute\_total\_variance**(*param*: AllanVarianceParam | *None* = *None*) → *None*  
Compute total variance with `cdl.computation.signal.compute_total_variance()`

**compute\_time\_deviation**(*param*: AllanVarianceParam | *None* = *None*) → *None*  
Compute time deviation with `cdl.computation.signal.compute_time_deviation()`



**compute\_all\_stability**(*param*: AllanVarianceParam | *None* = *None*) → *None*

Compute all stability analysis features using the following functions:

- `cdl.computation.signal.compute_allan_variance()`
- `cdl.computation.signal.compute_allan_deviation()`
- `cdl.computation.signal.compute_overlapping_allan_variance()`
- `cdl.computation.signal.compute_modified_allan_variance()`
- `cdl.computation.signal.compute_hadamard_variance()`
- `cdl.computation.signal.compute_total_variance()`
- `cdl.computation.signal.compute_time_deviation()`

**compute\_polyfit**(*param*: PolynomialFitParam | *None* = *None*) → *None*

Compute polynomial fitting curve

**compute\_fit**(*title*: str, *fitdlgfunc*: Callable) → *None*

Compute fitting curve using an interactive dialog

#### Parameters

- **title** – Title of the dialog
- **fitdlgfunc** – Fitting dialog function

**compute\_multigaussianfit**() → *None*

Compute multi-Gaussian fitting curve using an interactive dialog

**compute\_fwhm**(*param*: FWHMParam | *None* = *None*) → dict[str, ResultShape]

Compute FWHM with `cdl.computation.signal.compute_fwhm()`

**compute\_fw1e2**() → dict[str, ResultShape]

Compute FW at  $1/e^2$  with `cdl.computation.signal.compute_fw1e2()`

**compute\_full\_width\_at\_y**(*param*: OrdinateParam | *None* = *None*) → dict[str, ResultShape]

Compute full width at a given y with `cdl.computation.signal.compute_full_width_at_y()`

**compute\_stats**() → dict[str, ResultProperties]

Compute data statistics with `cdl.computation.signal.compute_stats()`

**compute\_histogram**(*param*: HistogramParam | *None* = *None*) → dict[str, ResultShape]

Compute histogram with `cdl.computation.signal.compute_histogram()`

**compute\_contrast**() → dict[str, ResultProperties]

Compute contrast with `cdl.computation.signal.compute_contrast()`

**compute\_x\_at\_minmax**() → dict[str, ResultProperties]

Compute x at min/max with `cdl.computation.signal.compute_x_at_minmax()`

**compute\_x\_at\_y**(*param*: OrdinateParam | *None* = *None*) → dict[str, ResultProperties]

Compute x at y with `cdl.computation.signal.compute_x_at_y()`.

**compute\_y\_at\_x**(*param*: AbscissaParam | *None* = *None*) → dict[str, ResultProperties]

Compute y at x with `cdl.computation.signal.compute_y_at_x()`.

**compute\_sampling\_rate\_period**() → dict[str, ResultProperties]

Compute sampling rate and period (mean and std) with `cdl.computation.signal.compute_sampling_rate_period()`

**compute\_bandwidth\_3db()** → None

Compute bandwidth at -3dB with `cdl.computation.signal.compute_bandwidth_3db()`

**compute\_dynamic\_parameters**(*param*: DynamicParam | None = None) → dict[str, ResultProperties]

Compute Dynamic Parameters (ENOB, SINAD, THD, SFDR, SNR) with `cdl.computation.signal.compute_dynamic_parameters()`

## Image processor

When working with images, the methods of `cdl.core.gui.processor.image.ImageProcessor` may be accessed.

**class** `cdl.core.gui.processor.image.ImageProcessor`(*panel*: SignalPanel | ImagePanel, *plotwidget*: PlotWidget)

Object handling image processing: operations, processing, analysis

**compute\_normalize**(*param*: NormalizeParam | None = None) → None

Normalize data with `cdl.computation.image.compute_normalize()`

**compute\_sum**() → None

Compute sum with `cdl.computation.image.compute_addition()`

**compute\_addition\_constant**(*param*: ConstantParam | None = None) → None

Compute sum with a constant using `cdl.computation.image.compute_addition_constant()`

**compute\_average**() → None

Compute average with `cdl.computation.image.compute_addition()` and dividing by the number of images

**compute\_product**() → None

Compute product with `cdl.computation.image.compute_product()`

**compute\_product\_constant**(*param*: ConstantParam | None = None) → None

Compute product with a constant using `cdl.computation.image.compute_product_constant()`

**compute\_logp1**(*param*: LogP1Param | None = None) → None

Compute base 10 logarithm using `cdl.computation.image.compute_logp1()`

**compute\_rotate**(*param*: RotateParam | None = None) → None

Rotate data arbitrarily using `cdl.computation.image.compute_rotate()`

**compute\_rotate90**() → None

Rotate data 90° with `cdl.computation.image.compute_rotate90()`

**compute\_rotate270**() → None

Rotate data 270° with `cdl.computation.image.compute_rotate270()`

**compute\_fliph**() → None

Flip data horizontally using `cdl.computation.image.compute_fliph()`

**compute\_flipv**() → None

Flip data vertically with `cdl.computation.image.compute_flipv()`

**distribute\_on\_grid**(*param*: GridParam | None = None) → None

Distribute images on a grid

**reset\_positions**() → None

Reset image positions

**compute\_resize**(*param*: `ResizeParam` | *None* = *None*) → *None*  
 Resize image with `cdl.computation.image.compute_resize()`

**compute\_binning**(*param*: `BinningParam` | *None* = *None*) → *None*  
 Binning image with `cdl.computation.image.compute_binning()`

**compute\_line\_profile**(*param*: `LineProfileParam` | *None* = *None*) → *None*  
 Compute profile along a vertical or horizontal line with `cdl.computation.image.compute_line_profile()`

**compute\_segment\_profile**(*param*: `SegmentProfileParam` | *None* = *None*)  
 Compute profile along a segment with `cdl.computation.image.compute_segment_profile()`

**compute\_average\_profile**(*param*: `AverageProfileParam` | *None* = *None*) → *None*  
 Compute average profile with `cdl.computation.image.compute_average_profile()`

**compute\_radial\_profile**(*param*: `RadialProfileParam` | *None* = *None*) → *None*  
 Compute radial profile with `cdl.computation.image.compute_radial_profile()`

**compute\_histogram**(*param*: `HistogramParam` | *None* = *None*) → *None*  
 Compute histogram with `cdl.computation.image.compute_histogram()`

**compute\_swap\_axes**() → *None*  
 Swap data axes with `cdl.computation.image.compute_swap_axes()`.

**compute\_inverse**() → *None*  
 Compute inverse

**compute\_abs**() → *None*  
 Compute absolute value with `cdl.computation.image.compute_abs()`

**compute\_re**() → *None*  
 Compute real part with `cdl.computation.image.compute_re()`

**compute\_im**() → *None*  
 Compute imaginary part with `cdl.computation.image.compute_im()`

**compute\_astype**(*param*: `DataTypeIParam` | *None* = *None*) → *None*  
 Convert data type with `cdl.computation.image.compute_astype()`

**compute\_log10**() → *None*  
 Compute Log10 with `cdl.computation.image.compute_log10()`

**compute\_exp**() → *None*  
 Compute Log10 with `cdl.computation.image.compute_exp()`

**compute\_arithmetic**(*obj2*: `ImageObj` | *None* = *None*, *param*: `ArithmeticParam` | *None* = *None*) → *None*  
 Compute arithmetic operation between two images with `cdl.computation.image.compute_arithmetic()`

**compute\_difference**(*obj2*: `ImageObj` | *list*[`ImageObj`] | *None* = *None*) → *None*  
 Compute difference between two images with `cdl.computation.image.compute_difference()`

**compute\_difference\_constant**(*param*: `ConstantParam` | *None* = *None*) → *None*  
 Compute difference with a constant with `cdl.computation.image.compute_difference_constant()`

**compute\_quadratic\_difference**(obj2: ImageObj | list[ImageObj] | None = None) → None  
Compute quadratic difference between two images with `cdl.computation.image.compute_quadratic_difference()`

**compute\_division**(obj2: ImageObj | list[ImageObj] | None = None) → None  
Compute division between two images with `cdl.computation.image.compute_division()`

**compute\_division\_constant**(param: ConstantParam | None = None) → None  
Compute division by a constant with `cdl.computation.image.compute_division_constant()`

**compute\_flatfield**(obj2: ImageObj | None = None, param: FlatFieldParam | None = None) → None  
Compute flat field correction with `cdl.computation.image.compute_flatfield()`

**compute\_calibration**(param: ZCalibrateParam | None = None) → None  
Compute data linear calibration with `cdl.computation.image.compute_calibration()`

**compute\_clip**(param: ClipParam | None = None) → None  
Compute maximum data clipping with `cdl.computation.image.compute_clip()`

**compute\_offset\_correction**(param: ROI2DParam | None = None) → None  
Compute offset correction with `cdl.computation.image.compute_offset_correction()`

**compute\_gaussian\_filter**(param: GaussianParam | None = None) → None  
Compute gaussian filter with `cdl.computation.image.compute_gaussian_filter()`

**compute\_moving\_average**(param: MovingAverageParam | None = None) → None  
Compute moving average with `cdl.computation.image.compute_moving_average()`

**compute\_moving\_median**(param: MovingMedianParam | None = None) → None  
Compute moving median with `cdl.computation.image.compute_moving_median()`

**compute\_wiener**() → None  
Compute Wiener filter with `cdl.computation.image.compute_wiener()`

**compute\_zero\_padding**(param: ZeroPadding2DParam | None = None) → None  
Compute zero padding with `cdl.computation.image.compute_zero_padding()`

**compute\_fft**(param: FFTParam | None = None) → None  
Compute FFT with `cdl.computation.image.compute_fft()`

**compute\_ifft**(param: FFTParam | None = None) → None  
Compute iFFT with `cdl.computation.image.compute_ifft()`

**compute\_magnitude\_spectrum**(param: SpectrumParam | None = None) → None  
Compute magnitude spectrum with `cdl.computation.image.compute_magnitude_spectrum()`

**compute\_phase\_spectrum**() → None  
Compute phase spectrum with `cdl.computation.image.compute_phase_spectrum()`

**compute\_psd**(param: SpectrumParam | None = None) → None  
Compute Power Spectral Density (PSD) with `cdl.computation.image.compute_psd()`

**compute\_butterworth**(param: ButterworthParam | None = None) → None  
Compute Butterworth filter with `cdl.computation.image.compute_butterworth()`

**compute\_threshold**(param: ThresholdParam | None = None) → None  
Compute parametric threshold with `cdl.computation.image.threshold.compute_threshold()`

**compute\_threshold\_isodata()** → None

Compute threshold using Isodata algorithm with `cdl.computation.image.threshold.compute_threshold_isodata()`

**compute\_threshold\_li()** → None

Compute threshold using Li algorithm with `cdl.computation.image.threshold.compute_threshold_li()`

**compute\_threshold\_mean()** → None

Compute threshold using Mean algorithm with `cdl.computation.image.threshold.compute_threshold_mean()`

**compute\_threshold\_minimum()** → None

Compute threshold using Minimum algorithm with `cdl.computation.image.threshold.compute_threshold_minimum()`

**compute\_threshold\_otsu()** → None

Compute threshold using Otsu algorithm with `cdl.computation.image.threshold.compute_threshold_otsu()`

**compute\_threshold\_triangle()** → None

Compute threshold using Triangle algorithm with `cdl.computation.image.threshold.compute_threshold_triangle()`

**compute\_threshold\_yen()** → None

Compute threshold using Yen algorithm with `cdl.computation.image.threshold.compute_threshold_yen()`

**compute\_all\_threshold()** → None

Compute all threshold algorithms using the following functions:

- `cdl.computation.image.threshold.compute_threshold_isodata()`
- `cdl.computation.image.threshold.compute_threshold_li()`
- `cdl.computation.image.threshold.compute_threshold_mean()`
- `cdl.computation.image.threshold.compute_threshold_minimum()`
- `cdl.computation.image.threshold.compute_threshold_otsu()`
- `cdl.computation.image.threshold.compute_threshold_triangle()`
- `cdl.computation.image.threshold.compute_threshold_yen()`

**compute\_adjust\_gamma**(*param*: AdjustGammaParam | None = None) → None

Compute gamma correction with `cdl.computation.image.exposure.compute_adjust_gamma()`

**compute\_adjust\_log**(*param*: AdjustLogParam | None = None) → None

Compute log correction with `cdl.computation.image.exposure.compute_adjust_log()`

**compute\_adjust\_sigmoid**(*param*: AdjustSigmoidParam | None = None) → None

Compute sigmoid correction with `cdl.computation.image.exposure.compute_adjust_sigmoid()`

**compute\_rescale\_intensity**(*param*: RescaleIntensityParam | None = None) → None

Rescale image intensity levels with `:py:func`cdl.computation.image.exposure.compute_rescale_intensity``

**compute\_equalize\_hist**(*param*: EqualizeHistParam | None = None) → None

Histogram equalization with `cdl.computation.image.exposure.compute_equalize_hist()`

**compute\_equalize\_adapthist**(*param*: EqualizeAdaptHistParam | *None* = *None*) → *None*  
Adaptive histogram equalization with `cdl.computation.image.exposure.compute_equalize_adapthist()`

**compute\_denoise\_tv**(*param*: DenoiseTVParam | *None* = *None*) → *None*  
Compute Total Variation denoising with `cdl.computation.image.restoration.compute_denoise_tv()`

**compute\_denoise\_bilateral**(*param*: DenoiseBilateralParam | *None* = *None*) → *None*  
Compute bilateral filter denoising with `cdl.computation.image.restoration.compute_denoise_bilateral()`

**compute\_denoise\_wavelet**(*param*: DenoiseWaveletParam | *None* = *None*) → *None*  
Compute Wavelet denoising with `cdl.computation.image.restoration.compute_denoise_wavelet()`

**compute\_denoise\_tophat**(*param*: MorphologyParam | *None* = *None*) → *None*  
Denoise using White Top-Hat with `cdl.computation.image.restoration.compute_denoise_tophat()`

**compute\_all\_denoise**(*params*: list | *None* = *None*) → *None*  
Compute all denoising filters using the following functions:

- `cdl.computation.image.restoration.compute_denoise_tv()`
- `cdl.computation.image.restoration.compute_denoise_bilateral()`
- `cdl.computation.image.restoration.compute_denoise_wavelet()`
- `cdl.computation.image.restoration.compute_denoise_tophat()`

**compute\_white\_tophat**(*param*: MorphologyParam | *None* = *None*) → *None*  
Compute White Top-Hat with `cdl.computation.image.morphology.compute_white_tophat()`

**compute\_black\_tophat**(*param*: MorphologyParam | *None* = *None*) → *None*  
Compute Black Top-Hat with `cdl.computation.image.morphology.compute_black_tophat()`

**compute\_erosion**(*param*: MorphologyParam | *None* = *None*) → *None*  
Compute Erosion with `cdl.computation.image.morphology.compute_erosion()`

**compute\_dilation**(*param*: MorphologyParam | *None* = *None*) → *None*  
Compute Dilation with `cdl.computation.image.morphology.compute_dilation()`

**compute\_opening**(*param*: MorphologyParam | *None* = *None*) → *None*  
Compute morphological opening with `cdl.computation.image.morphology.compute_opening()`

**compute\_closing**(*param*: MorphologyParam | *None* = *None*) → *None*  
Compute morphological closing with `cdl.computation.image.morphology.compute_closing()`

**compute\_all\_morphology**(*param*: MorphologyParam | *None* = *None*) → *None*  
Compute all morphology filters using the following functions:

- `cdl.computation.image.morphology.compute_white_tophat()`
- `cdl.computation.image.morphology.compute_black_tophat()`
- `cdl.computation.image.morphology.compute_erosion()`
- `cdl.computation.image.morphology.compute_dilation()`
- `cdl.computation.image.morphology.compute_opening()`



- `cdl.computation.image.morphology.compute_closing()`

**compute\_canny**(*param*: CannyParam | None = None) → None  
 Compute Canny filter with `cdl.computation.image.edges.compute_canny()`

**compute\_roberts**() → None  
 Compute Roberts filter with `cdl.computation.image.edges.compute_roberts()`

**compute\_prewitt**() → None  
 Compute Prewitt filter with `cdl.computation.image.edges.compute_prewitt()`

**compute\_prewitt\_h**() → None  
 Compute Prewitt filter (horizontal) with `cdl.computation.image.edges.compute_prewitt_h()`

**compute\_prewitt\_v**() → None  
 Compute Prewitt filter (vertical) with `cdl.computation.image.edges.compute_prewitt_v()`

**compute\_sobel**() → None  
 Compute Sobel filter with `cdl.computation.image.edges.compute_sobel()`

**compute\_sobel\_h**() → None  
 Compute Sobel filter (horizontal) with `cdl.computation.image.edges.compute_sobel_h()`

**compute\_sobel\_v**() → None  
 Compute Sobel filter (vertical) with `cdl.computation.image.edges.compute_sobel_v()`

**compute\_scharr**() → None  
 Compute Scharr filter with `cdl.computation.image.edges.compute_scharr()`

**compute\_scharr\_h**() → None  
 Compute Scharr filter (horizontal) with `cdl.computation.image.edges.compute_scharr_h()`

**compute\_scharr\_v**() → None  
 Compute Scharr filter (vertical) with `cdl.computation.image.edges.compute_scharr_v()`

**compute\_farid**() → None  
 Compute Farid filter with `cdl.computation.image.edges.compute_farid()`

**compute\_farid\_h**() → None  
 Compute Farid filter (horizontal) with `cdl.computation.image.edges.compute_farid_h()`

**compute\_farid\_v**() → None  
 Compute Farid filter (vertical) with `cdl.computation.image.edges.compute_farid_v()`

**compute\_laplace**() → None  
 Compute Laplace filter with `cdl.computation.image.edges.compute_laplace()`

**compute\_all\_edges**() → None  
 Compute all edges filters using the following functions:

- `cdl.computation.image.edges.compute_roberts()`
- `cdl.computation.image.edges.compute_prewitt()`
- `cdl.computation.image.edges.compute_prewitt_h()`
- `cdl.computation.image.edges.compute_prewitt_v()`
- `cdl.computation.image.edges.compute_sobel()`
- `cdl.computation.image.edges.compute_sobel_h()`

- `cdl.computation.image.edges.compute_sobel_v()`
- `cdl.computation.image.edges.compute_scharr()`
- `cdl.computation.image.edges.compute_scharr_h()`
- `cdl.computation.image.edges.compute_scharr_v()`
- `cdl.computation.image.edges.compute_farid()`
- `cdl.computation.image.edges.compute_farid_h()`
- `cdl.computation.image.edges.compute_farid_v()`
- `cdl.computation.image.edges.compute_laplace()`

**compute\_stats()** → dict[str, *ResultProperties*]

Compute data statistics with `cdl.computation.image.compute_stats()`

**compute\_centroid()** → dict[str, *ResultShape*]

Compute image centroid with `cdl.computation.image.compute_centroid()`

**compute\_enclosing\_circle()** → dict[str, *ResultShape*]

Compute minimum enclosing circle with `cdl.computation.image.compute_enclosing_circle()`

**compute\_peak\_detection()**(*param*: *Peak2DDetectionParam* | *None* = *None*) → dict[str, *ResultShape*]

Compute 2D peak detection with `cdl.computation.image.compute_peak_detection()`

**compute\_contour\_shape()**(*param*: *ContourShapeParam* | *None* = *None*) → dict[str, *ResultShape*]

Compute contour shape fit with `cdl.computation.image.detection.compute_contour_shape()`

**compute\_hough\_circle\_peaks()**(*param*: *HoughCircleParam* | *None* = *None*) → dict[str, *ResultShape*]

Compute peak detection based on a circle Hough transform with `cdl.computation.image.compute_hough_circle_peaks()`

**compute\_blob\_dog()**(*param*: *BlobDOGParam* | *None* = *None*) → dict[str, *ResultShape*]

Compute blob detection using Difference of Gaussian method with `cdl.computation.image.detection.compute_blob_dog()`

**compute\_blob\_doh()**(*param*: *BlobDOHParam* | *None* = *None*) → dict[str, *ResultShape*]

Compute blob detection using Determinant of Hessian method with `cdl.computation.image.detection.compute_blob_doh()`

**compute\_blob\_log()**(*param*: *BlobLOGParam* | *None* = *None*) → dict[str, *ResultShape*]

Compute blob detection using Laplacian of Gaussian method with `cdl.computation.image.detection.compute_blob_log()`

**compute\_blob\_opencv()**(*param*: *BlobOpenCVParam* | *None* = *None*) → dict[str, *ResultShape*]

Compute blob detection using OpenCV with `cdl.computation.image.detection.compute_blob_opencv()`



## 2.2.5 Internal data model

In its internal data model, DataLab stores data using two main classes:

- `cdl.obj.SignalObj`, which represents a signal object, and
- `cdl.obj.ImageObj`, which represents an image object.

These classes are defined in the `cdl.core.model` package but are exposed publicly in the `cdl.obj` package.

Also, DataLab uses many different datasets (based on guidata's `DataSet` class) to store the parameters of the computations. These datasets are defined in different modules but are exposed publicly in the `cdl.param` package.

**See also:**

The [API](#) section for more information on the public API.

## 2.2.6 Plugins

DataLab is a modular application. It is possible to add new features to DataLab by writing plugins. A plugin is a Python module that is loaded at startup by DataLab. A plugin may add new features to DataLab, or modify existing features.

The plugin system currently supports the following features:

- Processing features: add new processing tasks to the DataLab processing system, including specific graphical user interfaces.
- Input/output features: add new file formats to the DataLab file I/O system.
- HDF5 features: add new HDF5 file formats to the DataLab HDF5 I/O system.

### What is a plugin?

A plugin is a Python module that is loaded at startup by DataLab. A plugin may add new features to DataLab, or modify existing features.

A plugin is a Python module which file name starts with `cdl_`, and which contains a class derived from the `cdl.plugins.PluginBase` class. The name of the class is not important, as long as it is derived from `cdl.plugins.PluginBase` and has a `PLUGIN_INFO` attribute that is an instance of the `cdl.plugins.PluginInfo` class. The `PLUGIN_INFO` attribute is used by DataLab to retrieve information about the plugin.

---

**Note:** DataLab's plugin discovery mechanism will only load plugins that are defined in Python files which names start with `cdl_` (e.g. `cdl_myplugin.py`).

---

### Where to put a plugin?

As plugins are Python modules, they can be put anywhere in the Python path of the DataLab installation.

Special additional locations are available for plugins:

- The *plugins* directory in the user configuration folder (e.g. `C:/Users/JohnDoe/.DataLab/plugins` on Windows or `~/.DataLab/plugins` on Linux).
- The *plugins* directory in the same folder as the *DataLab* executable in case of a standalone installation.
- The *plugins* directory in the *cdl* package in case for internal plugins only (i.e. it is not recommended to put your own plugins there).

## How to develop a plugin?

To develop a plugin, you may start by deriving from one of the example plugins (see below) and modify it to suit your needs.

If you want to code a plugin in your usual Python development environment (preferably with an IDE like [Spyder](#)) and take advantage of the code completion, you can add the *cdl* package to your Python path.

This can be done:

- By installing DataLab in your Python environment (using one of the following methods: *Conda package*, *Package manager pip*, *Wheel package*, or *Source package*),
- Or by adding the *cdl* package to your Python path manually:
  - Download the DataLab source code from the [PyPI page](#),
  - Unzip the source code to a folder on your computer,
  - Add the *cdl* package to your Python path (e.g. by using the *PYTHONPATH Manager* in Spyder).

---

**Note:** Even if you have installed the *cdl* package properly in your Python environment, you won't be able to run the DataLab application from your development environment to test your plugin. You will need to run DataLab from the command line or from the shortcut created during the installation.

---

## Example: processing plugin

Here is a simple example of a plugin that adds a new features to DataLab.

```
# Copyright (c) DataLab Platform Developers, BSD 3-Clause license, see LICENSE file.

"""
Test Data Plugin for DataLab
-----

This plugin is an example of DataLab plugin. It provides test data samples
and some actions to test DataLab functionalities.
"""

import cdl.obj as dlo
import cdl.tests.data as test_data
from cdl.computation import image as cpima
from cdl.computation import signal as cpsig
from cdl.config import _
from cdl.plugins import PluginBase, PluginInfo

# -----
# All computation functions must be defined as global functions, otherwise
# they cannot be pickled and sent to the worker process
# -----

def add_noise_to_signal(
    src: dlo.SignalObj, p: test_data.GaussianNoiseParam
) -> dlo.SignalObj:
```

(continues on next page)

(continued from previous page)

```

"""Add gaussian noise to signal"""
dst = cpsig.dst_11(src, "add_gaussian_noise", f"mu={p.mu},sigma={p.sigma}")
test_data.add_gaussian_noise_to_signal(dst, p)
return dst

def add_noise_to_image(src: dlo.ImageObj, p: dlo.NormalRandomParam) -> dlo.ImageObj:
    """Add gaussian noise to image"""
    dst = cpima.dst_11(src, "add_gaussian_noise", f"mu={p.mu},sigma={p.sigma}")
    test_data.add_gaussian_noise_to_image(dst, p)
    return dst

class PluginTestData(PluginBase):
    """DataLab Test Data Plugin"""

    PLUGIN_INFO = PluginInfo(
        name=_("Test data"),
        version="1.0.0",
        description=_("Testing DataLab functionalities"),
    )

    # Signal processing features -----
    def add_noise_to_signal(self) -> None:
        """Add noise to signal"""
        self.signalpanel.processor.compute_11(
            add_noise_to_signal,
            paramclass=test_data.GaussianNoiseParam,
            title=_("Add noise"),
        )

    def create_paracetamol_signal(self) -> None:
        """Create paracetamol signal"""
        obj = test_data.create_paracetamol_signal()
        self.proxy.add_object(obj)

    def create_noisy_signal(self) -> None:
        """Create noisy signal"""
        obj = self.signalpanel.new_object(add_to_panel=False)
        if obj is not None:
            noiseparam = test_data.GaussianNoiseParam(_("Noise"))
            self.signalpanel.processor.update_param_defaults(noiseparam)
            if noiseparam.edit(self.main):
                test_data.add_gaussian_noise_to_signal(obj, noiseparam)
                self.proxy.add_object(obj)

    # Image processing features -----
    def add_noise_to_image(self) -> None:
        """Add noise to image"""
        self.imagepanel.processor.compute_11(
            add_noise_to_image,
            paramclass=dlo.NormalRandomParam,

```

(continues on next page)

(continued from previous page)

```

        title=_("Add noise"),
    )

def create_peak2d_image(self) -> None:
    """Create 2D peak image"""
    obj = self.imagepanel.new_object(add_to_panel=False)
    if obj is not None:
        param = test_data.PeakDataParam.create(size=max(obj.data.shape))
        self.imagepanel.processor.update_param_defaults(param)
        if param.edit(self.main):
            obj.data = test_data.get_peak2d_data(param)
            self.proxy.add_object(obj)

def create_sincos_image(self) -> None:
    """Create 2D sin cos image"""
    newparam = self.edit_new_image_parameters(hide_image_type=True)
    if newparam is not None:
        obj = test_data.create_sincos_image(newparam)
        self.proxy.add_object(obj)

def create_noisygauss_image(self) -> None:
    """Create 2D noisy gauss image"""
    newparam = self.edit_new_image_parameters(
        hide_image_height=True, hide_image_type=True
    )
    if newparam is not None:
        obj = test_data.create_noisygauss_image(newparam, add_annotations=False)
        self.proxy.add_object(obj)

def create_multigauss_image(self) -> None:
    """Create 2D multi gauss image"""
    newparam = self.edit_new_image_parameters(
        hide_image_height=True, hide_image_type=True
    )
    if newparam is not None:
        obj = test_data.create_multigauss_image(newparam)
        self.proxy.add_object(obj)

def create_2dstep_image(self) -> None:
    """Create 2D step image"""
    newparam = self.edit_new_image_parameters(hide_image_type=True)
    if newparam is not None:
        obj = test_data.create_2dstep_image(newparam)
        self.proxy.add_object(obj)

def create_ring_image(self) -> None:
    """Create 2D ring image"""
    param = test_data.RingParam(_("Ring"))
    if param.edit(self.main):
        obj = test_data.create_ring_image(param)
        self.proxy.add_object(obj)

```

(continues on next page)

(continued from previous page)

```

def create_annotated_image(self) -> None:
    """Create annotated image"""
    obj = test_data.create_annotated_image()
    self.proxy.add_object(obj)

# Plugin menu entries -----
def create_actions(self) -> None:
    """Create actions"""
    # Signal Panel -----
    sah = self.signalpanel.acthandler
    with sah.new_menu(_("Test data")):
        sah.new_action(_("Add noise to signal"), triggered=self.add_noise_to_signal)
        sah.new_action(
            _("Load spectrum of paracetamol"),
            triggered=self.create_paracetamol_signal,
            select_condition="always",
            separator=True,
        )
        sah.new_action(
            _("Create noisy signal"),
            triggered=self.create_noisy_signal,
            select_condition="always",
        )
    # Image Panel -----
    iah = self.imagepanel.acthandler
    with iah.new_menu(_("Test data")):
        iah.new_action(_("Add noise to image"), triggered=self.add_noise_to_image)
        # with iah.new_menu(_("Data samples")):
        iah.new_action(
            _("Create image with peaks"),
            triggered=self.create_peak2d_image,
            select_condition="always",
            separator=True,
        )
        iah.new_action(
            _("Create 2D sin cos image"),
            triggered=self.create_sincos_image,
            select_condition="always",
        )
        iah.new_action(
            _("Create 2D noisy gauss image"),
            triggered=self.create_noisygauss_image,
            select_condition="always",
        )
        iah.new_action(
            _("Create 2D multi gauss image"),
            triggered=self.create_multigauss_image,
            select_condition="always",
        )
        iah.new_action(
            _("Create annotated image"),
            triggered=self.create_annotated_image,

```

(continues on next page)

(continued from previous page)

```

        select_condition="always",
    )
    iah.new_action(
        _("Create 2D step image"),
        triggered=self.create_2dstep_image,
        select_condition="always",
    )
    iah.new_action(
        _("Create ring image"),
        triggered=self.create_ring_image,
        select_condition="always",
    )

```

### Example: input/output plugin

Here is a simple example of a plugin that adds a new file formats to DataLab.

```

# Copyright (c) DataLab Platform Developers, BSD 3-Clause license, see LICENSE file.

"""
Image file formats Plugin for DataLab
-----

This plugin is an example of DataLab plugin.
It provides image file formats from cameras, scanners, and other acquisition devices.
"""

import struct

import numpy as np

from cdl.core.io.base import FormatInfo
from cdl.core.io.image.base import ImageFormatBase

# =====
# Thales Pixium FXD file format
# =====

class FXDFile:
    """Class implementing Thales Pixium FXD Image file reading feature

    Args:
        fname (str): path to FXD file
        debug (bool): debug mode
    """

    HEADER = "<111111ffl"

    def __init__(self, fname: str = None, debug: bool = False) -> None:
        self.__debug = debug

```

(continues on next page)

(continued from previous page)

```

self.file_format = None # long
self.nbcolls = None # long
self.nbrows = None # long
self.nbframes = None # long
self.pixeltype = None # long
self.quantlevels = None # long
self.maxlevel = None # float
self.minlevel = None # float
self.comment_length = None # long
self.fname = None
self.data = None
if fname is not None:
    self.load(fname)

def __repr__(self) -> str:
    """Return a string representation of the object"""
    info = (
        ("Image width", f"{self.nbcolls:d}"),
        ("Image Height", f"{self.nbrows:d}"),
        ("Frame number", f"{self.nbframes:d}"),
        ("File format", f"{self.file_format:d}"),
        ("Pixel type", f"{self.pixeltype:d}"),
        ("Quantlevels", f"{self.quantlevels:d}"),
        ("Min. level", f"{self.minlevel:f}"),
        ("Max. level", f"{self.maxlevel:f}"),
        ("Comment length", f"{self.comment_length:d}"),
    )
    desc_len = max(len(d) for d in list(zip(*info))[0]) + 3
    res = ""
    for description, value in info:
        res += ("{: " + str(desc_len) + "}}{}\n").format(description + ": ", value)

    res = object.__repr__(self) + "\n" + res
    return res

def load(self, fname: str) -> None:
    """Load header and image pixel data

    Args:
        fname (str): path to FXD file
    """
    with open(fname, "rb") as data_file:
        header_s = struct.Struct(self.HEADER)
        record = data_file.read(9 * 4)
        unpacked_rec = header_s.unpack(record)
        (
            self.file_format,
            self.nbcolls,
            self.nbrows,
            self.nbframes,
            self.pixeltype,
            self.quantlevels,

```

(continues on next page)

(continued from previous page)

```

        self.maxlevel,
        self.minlevel,
        self.comment_length,
    ) = unpacked_rec
    if self.__debug:
        print(unpacked_rec)
        print(self)
    data_file.seek(128 + self.comment_length)
    if self.pixeltype == 0:
        size, dtype = 4, np.float32
    elif self.pixeltype == 1:
        size, dtype = 2, np.uint16
    elif self.pixeltype == 2:
        size, dtype = 1, np.uint8
    else:
        raise NotImplementedError(f"Unsupported pixel type: {self.pixeltype}")
    block = data_file.read(self.nbrows * self.nbcolls * size)
    data = np.frombuffer(block, dtype=dtype)
    self.data = data.reshape(self.nbrows, self.nbcolls)

class FXDImageFormat(ImageFormatBase):
    """Object representing Thales Pixium (FXD) image file type"""

    FORMAT_INFO = FormatInfo(
        name="Thales Pixium",
        extensions="*.fxd",
        readable=True,
        writeable=False,
    )

    @staticmethod
    def read_data(filename: str) -> np.ndarray:
        """Read data and return it

        Args:
            filename (str): path to FXD file

        Returns:
            np.ndarray: image data
        """
        fxd_file = FXDFile(filename)
        return fxd_file.data

# =====
# Dürr NDT XYZ file format
# =====

class XYZImageFormat(ImageFormatBase):
    """Object representing Dürr NDT XYZ image file type"""

```

(continues on next page)



(continued from previous page)

```

FORMAT_INFO = FormatInfo(
    name="Dürr NDT",
    extensions="*.xyz",
    readable=True,
    writeable=True,
)

@staticmethod
def read_data(filename: str) -> np.ndarray:
    """Read data and return it

    Args:
        filename (str): path to XYZ file

    Returns:
        np.ndarray: image data
    """
    with open(filename, "rb") as fdesc:
        cols = int(np.fromfile(fdesc, dtype=np.uint16, count=1)[0])
        rows = int(np.fromfile(fdesc, dtype=np.uint16, count=1)[0])
        arr = np.fromfile(fdesc, dtype=np.uint16, count=cols * rows)
        arr = arr.reshape((rows, cols))
    return np.fliplr(arr)

@staticmethod
def write_data(filename: str, data: np.ndarray) -> None:
    """Write data to file

    Args:
        filename: File name
        data: Image array data
    """
    data = np.fliplr(data)
    with open(filename, "wb") as fdesc:
        fdesc.write(np.array(data.shape[1], dtype=np.uint16).tobytes())
        fdesc.write(np.array(data.shape[0], dtype=np.uint16).tobytes())
        fdesc.write(data.tobytes())

```

## Other examples

Other examples of plugins can be found in the *plugins/examples* directory of the DataLab source code (explore [here on GitHub](#)).

## Public API

### DataLab plugin system

DataLab plugin system provides a way to extend the application with new functionalities.

Plugins are Python modules that relies on two classes:

- *PluginInfo*, which stores information about the plugin
- *PluginBase*, which is the base class for all plugins

Plugins may also extends DataLab I/O features by providing new image or signal formats. To do so, they must provide a subclass of *ImageFormatBase* or *SignalFormatBase*, in which format infos are defined using the *FormatInfo* class.

```
class cdl.plugins.PluginRegistry(name, bases, attrs)
    Metaclass for registering plugins

    classmethod get_plugin_classes() → list[type[PluginBase]]
        Return plugin classes

    classmethod get_plugins() → list[PluginBase]
        Return plugin instances

    classmethod get_plugin(name_or_class: str | type[PluginBase]) → PluginBase | None
        Return plugin instance

    classmethod register_plugin(plugin: PluginBase)
        Register plugin

    classmethod unregister_plugin(plugin: PluginBase)
        Unregister plugin

    classmethod get_plugin_infos(html: bool = True) → str
        Return plugin infos (names, versions, descriptions) in html format

    Parameters
        html – return html formatted text (default: True)

class cdl.plugins.PluginInfo(name: str = None, version: str = '0.0.0', description: str = "", icon: str = None)
    Plugin info

class cdl.plugins.PluginBaseMeta(name, bases, namespace, /, **kwargs)
    Mixed metaclass to avoid conflicts

class cdl.plugins.PluginBase
    Plugin base class

    property signalpanel: SignalPanel
        Return signal panel

    property imagepanel: ImagePanel
        Return image panel

    show_warning(message: str)
        Show warning message
```

**show\_error**(*message: str*)

Show error message

**show\_info**(*message: str*)

Show info message

**ask\_ynsno**(*message: str, title: str | None = None, cancelable: bool = False*) → *bool*

Ask yes/no question

**edit\_new\_signal\_parameters**(*title: str | None = None, size: int | None = None, hide\_signal\_type: bool = True*) → *NewSignalParam*

Create and edit new signal parameter dataset

#### Parameters

- **title** – title of the new signal
- **size** – size of the new signal (default: None, get from current signal)
- **hide\_signal\_type** – hide signal type parameter (default: True)

#### Returns

New signal parameter dataset (or None if canceled)

**edit\_new\_image\_parameters**(*title: str | None = None, shape: tuple[int, int] | None = None, hide\_image\_height: bool = False, hide\_image\_type: bool = True, hide\_image\_dtype: bool = False*) → *NewImageParam | None*

Create and edit new image parameter dataset

#### Parameters

- **title** – title of the new image
- **shape** – shape of the new image (default: None, get from current image)
- **hide\_image\_height** – hide image height parameter (default: False)
- **hide\_image\_type** – hide image type parameter (default: True)
- **hide\_image\_dtype** – hide image data type parameter (default: False)

#### Returns

New image parameter dataset (or None if canceled)

**is\_registered**()

Return True if plugin is registered

**register**(*main: main.CDLMainWindow*) → *None*

Register plugin

**unregister**()

Unregister plugin

**register\_hooks**()

Register plugin hooks

**unregister\_hooks**()

Unregister plugin hooks

**abstract\_create\_actions**()

Create actions

`cdl.plugins.discover_plugins()` → `list[type[PluginBase]]`

Discover plugins using naming convention

**Returns**

List of discovered plugins (as classes)

`cdl.plugins.get_available_plugins()` → `list[PluginBase]`

Instantiate and get available plugins

**Returns**

List of available plugins (as instances)

## 2.2.7 Log viewer

Despite countless efforts (unit testing, test coverage...), DataLab might crash or behave unexpectedly.

**For those situations, DataLab provides two logs (located in your home directory):**

- “Traceback log”, for Python exceptions
- “Faulthandler log”, for system failures (e.g. Qt-related crash)

---

**Note:** The log viewer is accessible from the “?” menu in the main window.

---

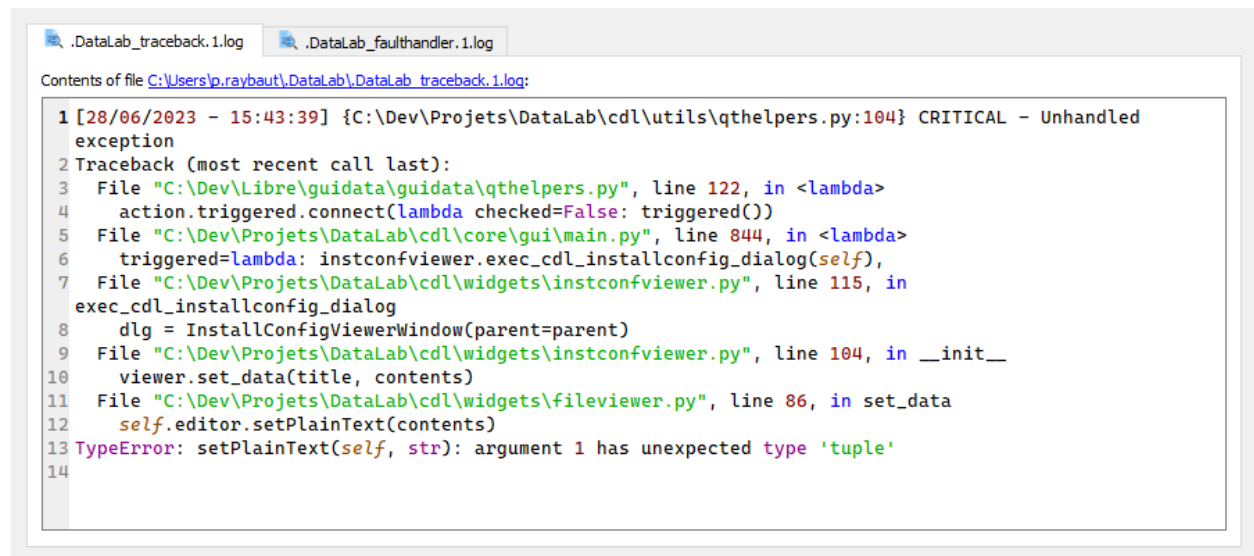


Fig. 6: DataLab log viewer (see “?” menu)

If DataLab crashed or if any Python exception is raised during its execution, those log files will be updated accordingly. DataLab will even notify that new informations are available in log files at next startup. This is an invitation to submit a bug report.

Reporting unexpected behavior or any other bug on [GitHub Issues](#) will be greatly appreciated, especially if those log file contents are attached to the report (as information on your installation configuration, see [Installation and configuration viewer](#)).

## 2.2.8 Installation and configuration viewer

Because of the multiple ways of installing DataLab on your machine, understanding why the application behaves unexpectedly without any information on your configuration could be very challenging.

That is why DataLab provides the dialog box “Installation and configuration” which gathers all the information about your installation and configuration.

**Note:** The installation and configuration viewer is accessible from the “?” menu in the main window.

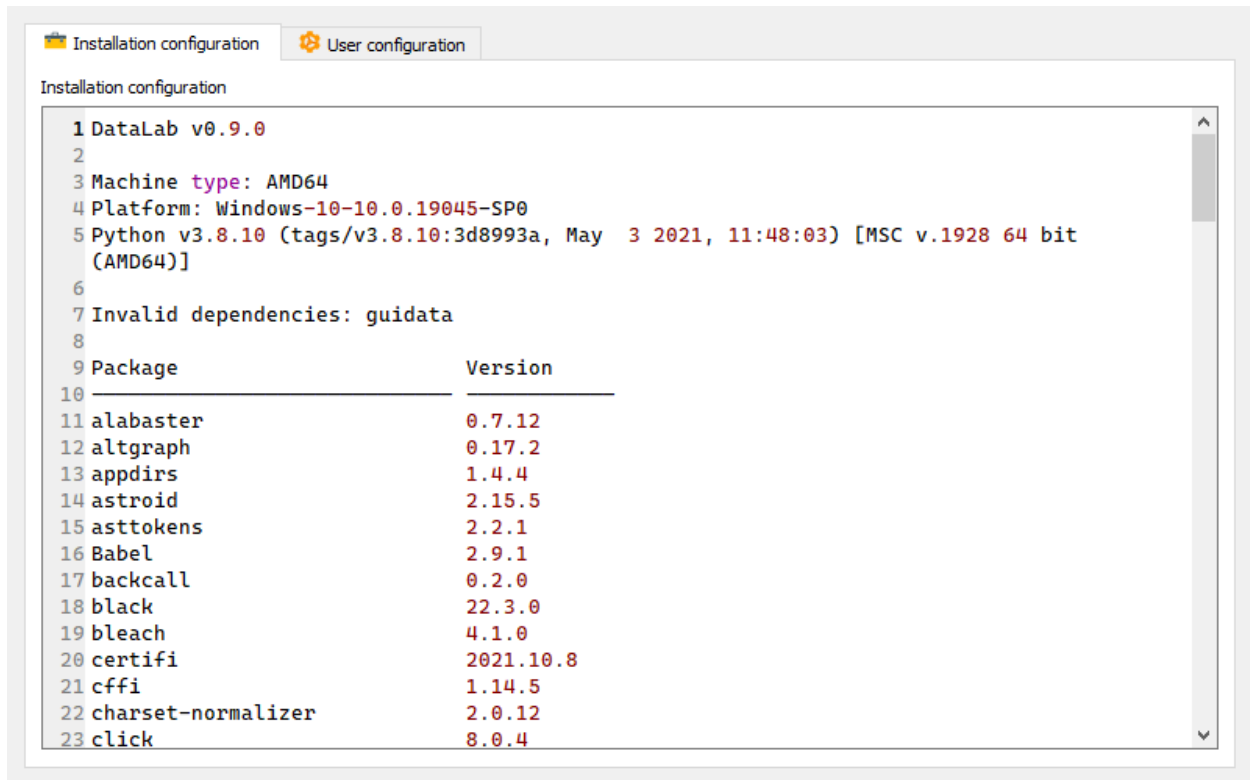


Fig. 7: Installation and configuration (see “?” menu)

Reporting unexpected behavior or any other bug on [GitHub Issues](#) will be greatly appreciated, especially if above contents are attached to the report (as well log files, see [Log viewer](#)).

## 2.2.9 Settings

### 2.2.10 Command line features

#### Run DataLab

To run DataLab from the command line, type the following:

```
$ datalab
```

To show help on command line usage, simply run:

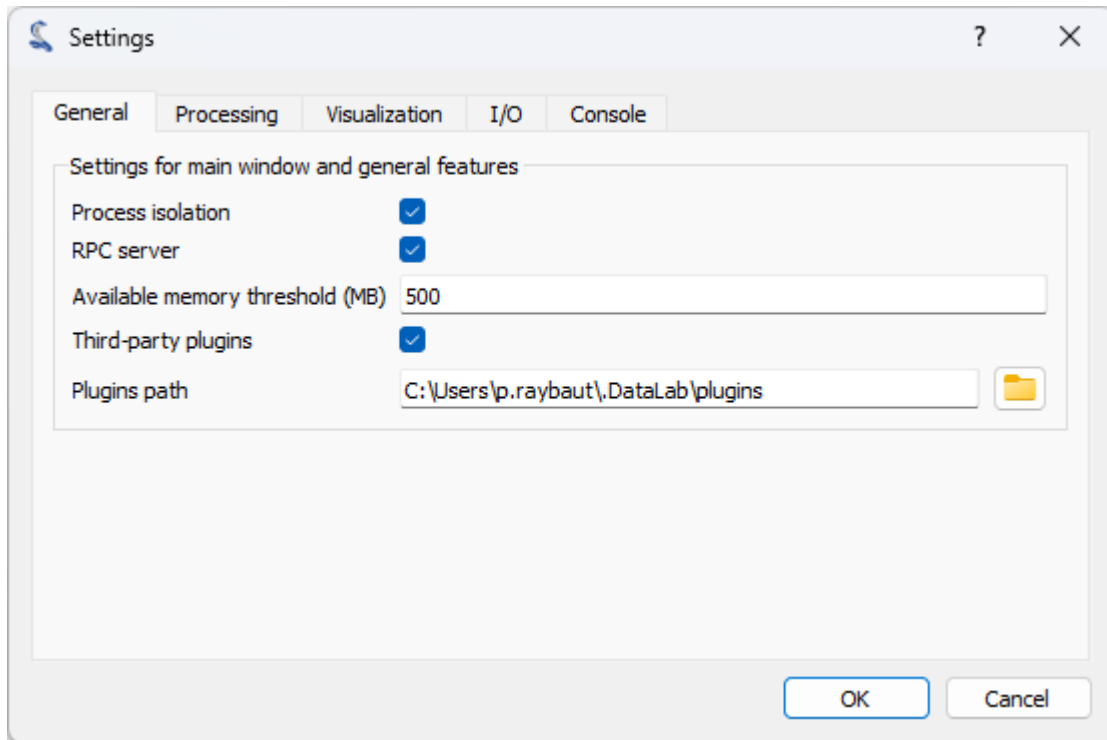


Fig. 8: Screenshot of the “Settings” dialog box.

```
$ datalab --help
usage: app.py [-h] [-b path] [-v] [--unattended] [--screenshot] [--delay DELAY] [--
  ↪xmlrpcport PORT]
               [--verbose {quiet,minimal,normal}]
               [h5]

Run DataLab

positional arguments:
h5                      HDF5 file names (separated by ';'), optionally with dataset name.
  ↪(separated by ',')

options:
-h, --help              show this help message and exit
-b path, --h5browser path
                        path to open with HDF5 browser
-v, --version           show DataLab version
--reset                reset DataLab configuration
--unattended           non-interactive mode
--accept_dialogs       accept dialogs in unattended mode
--screenshot           automatic screenshots
--delay DELAY          delay (ms) before quitting application in unattended mode
--xmlrpcport XMLRPCPORT
                        XML-RPC port number
--verbose {quiet,normal,debug}
                        verbosity level: for debugging/testing purpose
```

### Open HDF5 file at startup

To open HDF5 files, or even import only a specified HDF5 dataset, use the following:

```
$ datalab /path/to/file1.h5
$ datalab /path/to/file1.h5,/path/to/dataset1
$ datalab /path/to/file1.h5,/path/to/dataset1;/path/to/file2.h5,/path/to/dataset2
```

### Open HDF5 browser at startup

To open the HDF5 browser at startup, use one of the following commands:

```
$ datalab -b /path/to/file1.h5
$ datalab --h5browser /path/to/file1.h5
```

### Run DataLab demo

To execute DataLab demo, run the following:

```
$ datalab-demo
```

### Run technical validation tests

---

**Note:** Technical validation tests are directly included in individual unit tests and are disseminated throughout the code. The test functions including validation tests are marked with the `@pytest.mark.validation` decorator.

---

To execute DataLab technical validation tests, run the following:

```
$ pytest -m validation
```

#### See also:

See section [Validation](#) for more information on DataLab's validation strategy.

### Run complete test suite

To execute all DataLab unit tests, simply run:

```
$ pytest
```

## Run interactive tests

To execute DataLab interactive tests, run the following:

```
$ datalab-tests
```

The screenshot shows the DataLab IDE interface. On the left, a file explorer lists various test modules under 'Tests - cdLapp module'. The file 'features/embedded2\_unit.py' is selected. On the right, the 'Description' pane for the selected test is shown. It contains a title 'Application embedded test 2', a description of the test's purpose, and the corresponding Python code.

**Description**

**Application embedded test 2**

DataLab main window is simply hidden when closing application.  
It is shown and raised above other windows when reopening application.

```
1 from cdLapp.core.gui.main import CDLMainWindow
2 from cdLapp.tests.features import embedded1_unit
3
4
5 class HostWindow(embedded1_unit.AbstractHostWindow):
6     """Test main view"""
7
8     def init_cdl(self) → None:
9         """Open DataLab test"""
10        if self.cdLapp is None:
11            self.cdLapp = CDLMainWindow(console=False, hide_on_close=True)
12            self.host.log("✳ Initialized DataLab window")
13            self.cdLapp.show()
14        else:
15            self.cdLapp.show()
16            self.cdLapp.raise_()
17            self.host.log("⇒ Shown DataLab window")
18
19    def close_cdl(self) → None:
20        """Close DataLab window"""
21        if self.cdLapp is not None:
22            self.host.log("⇒ Closed DataLab")
23            self.cdLapp.close()
24
25    def closeEvent(self, event) → None: # pylint: disable=invalid-name
26        """Close event
27
28        Reimplemented from QWidget.closeEvent"""
29        if self.cdLapp is None or self.cdLapp.close_properly():
30            event.accept()
```

Run this script



## 2.3 Signal processing

This section describes the features specific to the signal processing panel. The signal processing panel is the default panel when DataLab is started.

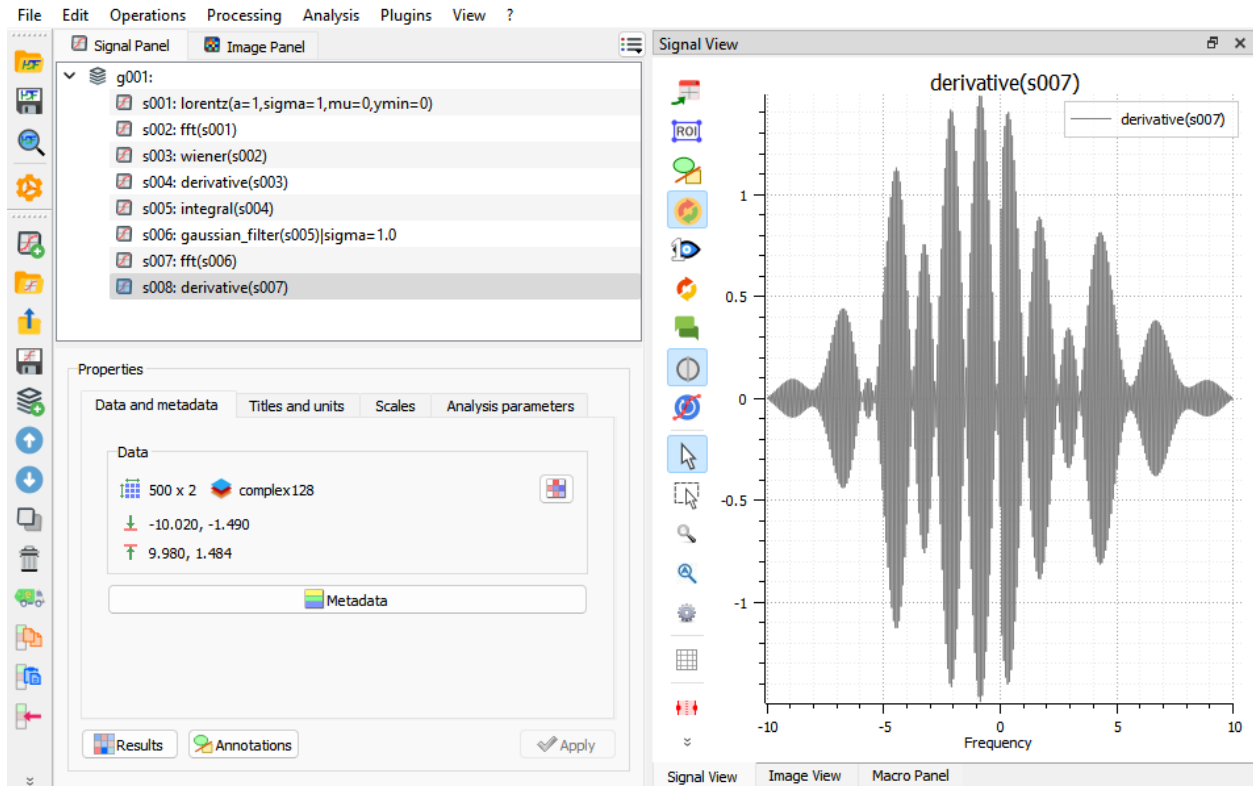


Fig. 9: DataLab main window: Signal processing view

### 2.3.1 Create, open and save Signals

This section describes how to create, open and save signals (and workspaces).

When the “Signal Panel” is selected, the menus and toolbars are updated to provide signal-related actions.

The “File” menu allows you to:

- Create, open, save and close signals (see below).
- Save and restore the current workspace or browse HDF5 files (see [Workspace](#)).
- Edit DataLab preferences (see [Settings](#)).

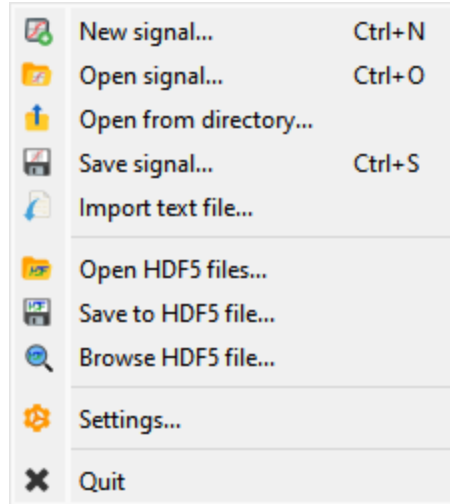


Fig. 10: Screenshot of the “File” menu.

## New signal

Create a new signal from various models:

Model	Equation
Zeros	$y[i] = 0$
Gaussian	$y = y_0 + \frac{A}{\sqrt{2\pi} \cdot \sigma} \cdot \exp(-\frac{1}{2} \cdot (\frac{x - x_0}{\sigma})^2)$
Lorentzian	$y = y_0 + \frac{A}{\sigma \cdot \pi} \cdot \frac{1}{1 + (\frac{x - x_0}{\sigma})^2}$
Voigt	$y = y_0 + A \cdot \frac{\text{Re}(\exp(-z^2) \cdot \text{erfc}(-j \cdot z))}{\sqrt{2\pi} \cdot \sigma}$ with $z = \frac{x - x_0 - j \cdot \sigma}{\sqrt{2} \cdot \sigma}$
Random (uniform law)	$y[i] \in [-0.5, 0.5]$
Random (normal law)	$y[i] \sim \mathcal{N}(-0.5, 0.5)$
Sine	$y = y_0 + A \cdot \sin(2\pi \cdot f \cdot x + \phi)$
Cosine	$y = y_0 + A \cdot \cos(2\pi \cdot f \cdot x + \phi)$
Sawtooth	$y = y_0 + A \cdot \left( 2 \left( f x + \frac{\phi}{2\pi} - \left\lfloor f x + \frac{\phi}{2\pi} + \frac{1}{2} \right\rfloor \right) \right)$
Triangle	$y = y_0 + A \cdot \text{sawtooth}(2\pi f x + \phi, \text{width} = 0.5)$
Square	$y = y_0 + A \cdot \text{sgn}(\sin(2\pi f x + \phi))$
Cardinal sine	$y = y_0 + A \cdot \text{sinc}(2\pi f x + \phi)$
Step	$y = y_0 + A \cdot \begin{cases} 1 & \text{if } x > x_0 \\ 0 & \text{otherwise} \end{cases}$
Exponential	$y = y_0 + A \cdot \exp(B \cdot x)$
Pulse	$y = y_0 + A \cdot \begin{cases} 1 & \text{if } x_0 < x < x_1 \\ 0 & \text{otherwise} \end{cases}$
Polynomial	$y = y_0 + A_0 + A_1 \cdot x + A_2 \cdot x^2 + \dots + A_n \cdot x^n$
Experimental	Manual input of X and Y values

## Open signal

Create a new signal from the following supported filetypes:

File type	Extensions
Text files	.txt, .csv
NumPy arrays	.npy
MAT-Files	.mat

## Save signal

Save current signal to the following supported filetypes:

File type	Extensions
Text files	.csv

## Import text file

DataLab can natively import signal files (e.g. CSV, NPY, etc.). However some specific text file formats may not be supported. In this case, you can use the *Import text file* feature, which allows you to import a text file and convert it to a signal.

This feature is accessible from the *File* menu, under the *Import text file* option.

It opens an import wizard that guides you through the process of importing the text file.

### Step 1: Select the source

The first step is to select the source of the text file. You can either select a file from your computer or the clipboard if you have copied the text from another application.

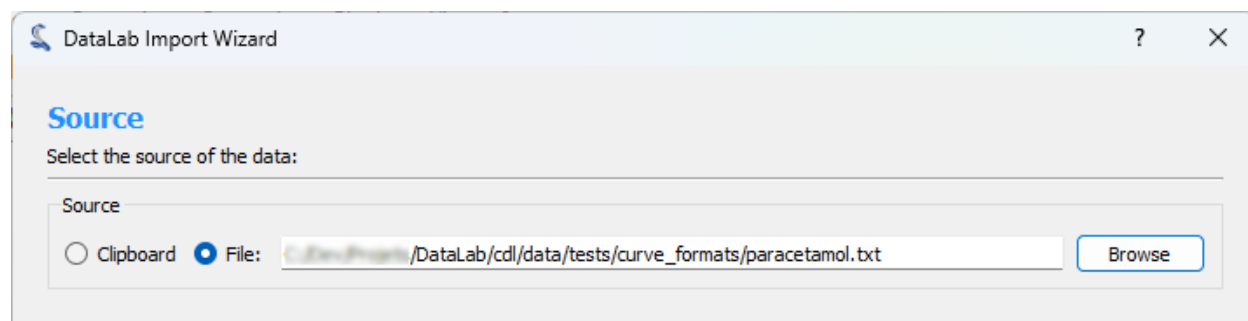


Fig. 11: Step 1: Select the source

## Step 2: Preview and configure the import

The second step consists of configuring the import and previewing the result. You can configure the following options:

- **Delimiter:** The character used to separate the values in the text file.
- **Comments:** The character used to indicate that the line is a comment and should be ignored.
- **Rows to Skip:** The number of rows to skip at the beginning of the file.
- **Maximum Number of Rows:** The maximum number of rows to import. If the file contains more rows, they will be ignored.
- **Transpose:** If checked, the rows and columns will be transposed.
- **Data type:** The destination data type of the imported data.
- **First Column is X:** If checked, the first column will be used as the X axis.

When you are done configuring the import, click the *Apply* button to see the result.

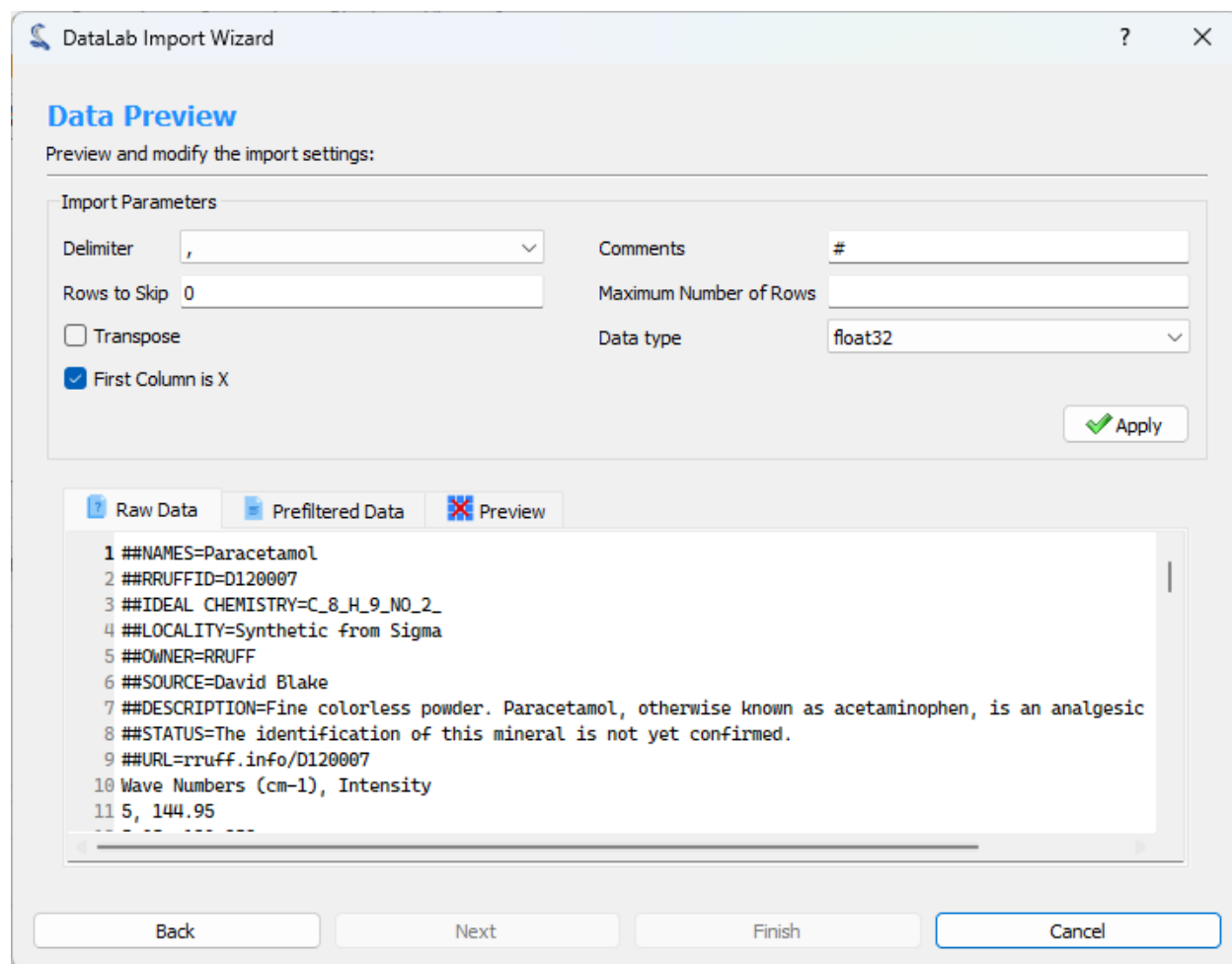


Fig. 12: Step 2: Configure the import

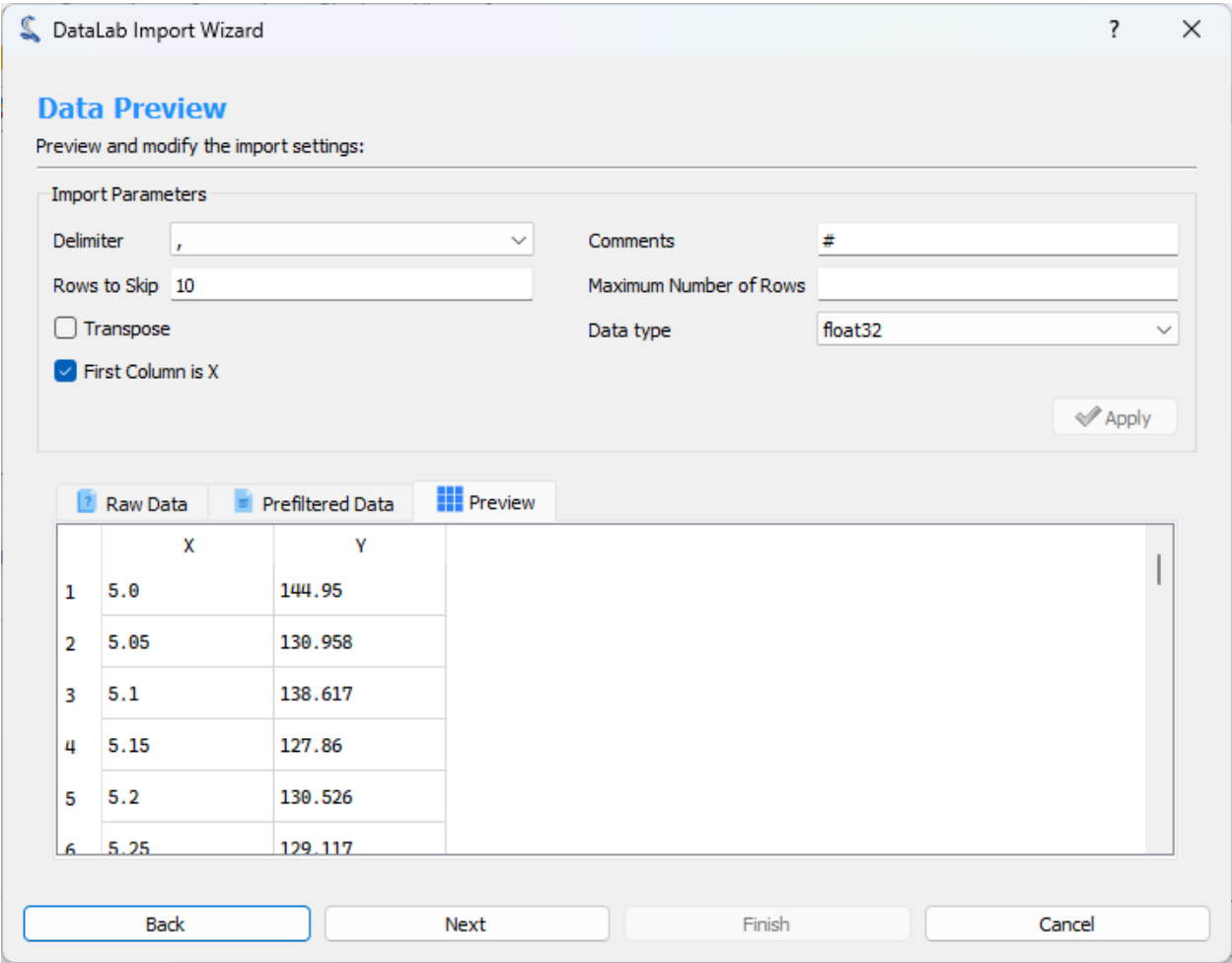


Fig. 13: Step 2: Preview the result

### Step 3: Show graphical representation

The third step shows a graphical representation of the imported data. You can use the *Finish* button to import the data into DataLab workspace.

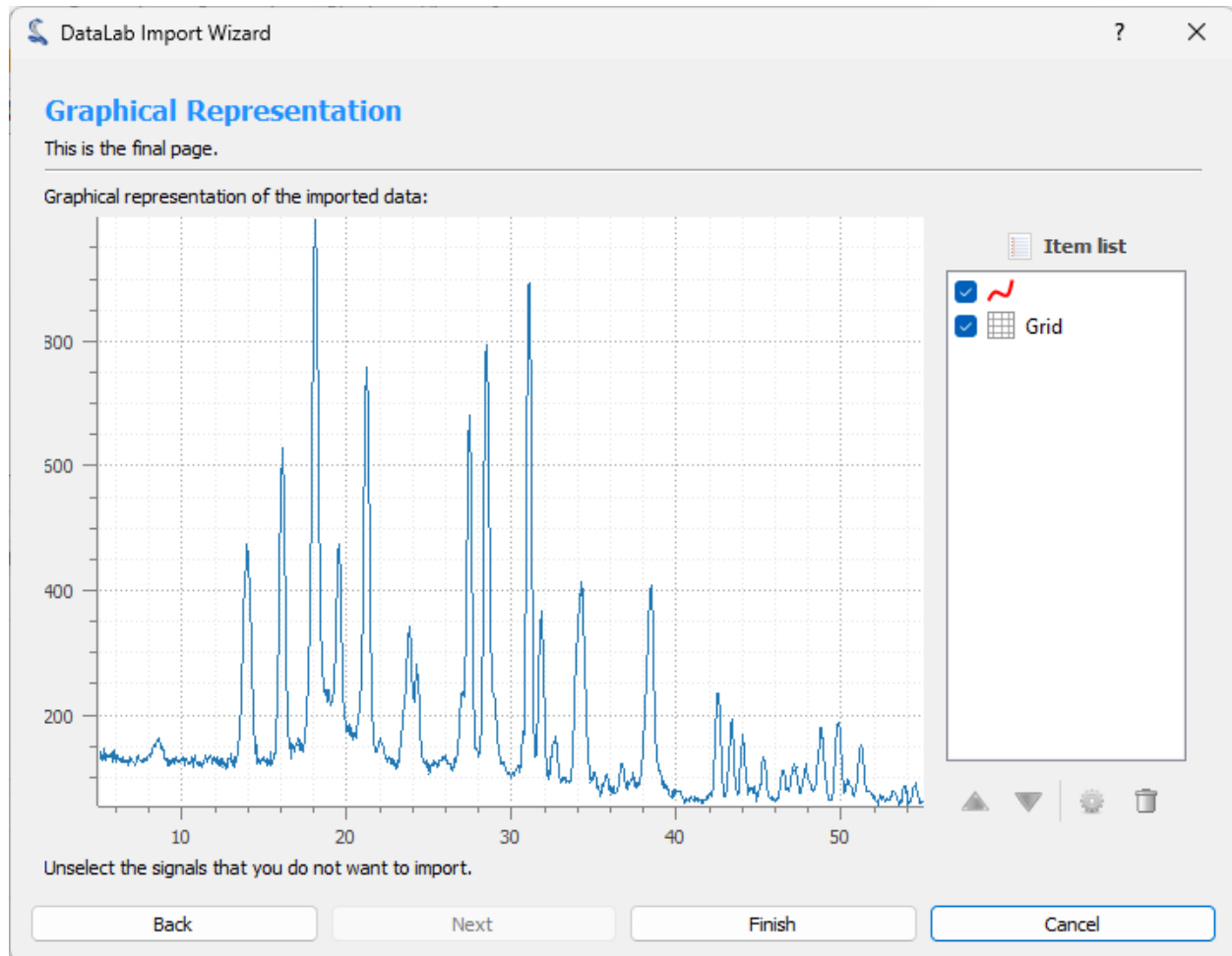


Fig. 14: Step 3: Show graphical representation

### 2.3.2 Manipulate metadata

This section describes how to manipulate metadata in DataLab.

The “Edit” menu allows you to perform classic editing operations on the current signal or group of signals (create/rename group, move up/down, delete signal/group of signals, etc.).

It also allows you to manipulate metadata associated with the current signal.

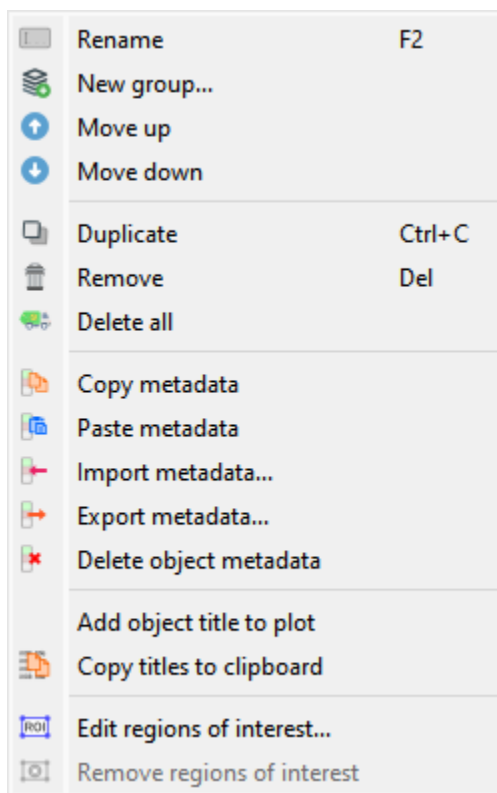


Fig. 15: Screenshot of the “Edit” menu.

### Copy/paste metadata

As metadata contains useful information about the signal, it can be copied and pasted from one signal to another by selecting the “Copy metadata” and “Paste metadata” actions in the “Edit” menu.

This feature allows you to transfer those information from one signal to another:

- *Regions Of Interest (ROIs)*: that is a very efficient way to reuse the same ROI on different signals and easily compare the results of the analysis on those signals
- Analyze results, such as peak positions or FWHM intervals (the relevance of transferring such information depends on the context and is up to the user to decide)
- Any other information that you may have added to the metadata of a signal

---

**Note:** Copying metadata from a signal to another will overwrite the metadata of the destination signal (for the metadata keys that are common to both signals) or simply add the metadata keys that are not present in the destination signal.

---

## Import/export metadata

Metadata can also be imported and exported from/to a JSON file using the “Import metadata” and “Export metadata” actions in the “Edit” menu. This is exactly the same as the copy/paste metadata feature (see above for more details on the use cases of this feature), but it allows you to save the metadata to a file and then import it back later.

## Delete metadata

When deleting metadata using the “Delete metadata” action in the “Edit” menu, you will be prompted to confirm the deletion of Region of Interests (ROIs) if they are present in the metadata. After this eventual confirmation, the metadata will be deleted, meaning that analysis results, ROIs, and any other information associated with the signal will be lost.

## Signal titles

Signal titles may be considered as metadata from a user point of view, even if they are not stored in the metadata of the signal (but in an attribute of the signal object).

The “Edit” menu allows you to:

- “Add object title to plot”: this action will add a label on top of the signal with its title.
- “Copy titles to clipboard”: this action will copy the titles of the selected signals to the clipboard, which might be useful to paste them in a text editor or in a spreadsheet.

Example of the content of the clipboard:

```
g001:
  s001: lorentz(a=1,sigma=1,mu=0,ymin=0)
  s002: derivative(s001)
  s003: wiener(s002)
g002: derivative(g001)
  s004: derivative(s001)
  s005: derivative(s002)
  s006: derivative(s003)
g003: fft(g002)
  s007: fft(s004)
  s008: fft(s005)
  s009: fft(s006)
```

## Regions Of Interest (ROI)

The Regions Of Interest (ROI) are signal areas that are defined by the user to perform specific operations, processing, or analysis on them.

ROI are taken into account almost in all computing features in DataLab:

- The “Operations” menu features are done only on the ROI if one is defined (except if the operation changes the number of points, like interpolation or resampling).
- The “Processing” menu actions are performed only on the ROI if one is defined (except if the destination signal data type is different from the source’s, like in the Fourier analysis features).
- The “Analysis” menu actions are done only on the ROI if one is defined.



**Note:** ROI are stored as metadata, and thus attached to signal.

The “Edit” menu allows you to:

- “Edit regions of interest” : open a dialog box to manage ROI associated with the selected signal (add, remove, move, resize, etc.). The ROI definition dialog is exactly the same as ROI extraction (see below): the ROI is defined by moving the position and adjusting the width of an horizontal range.

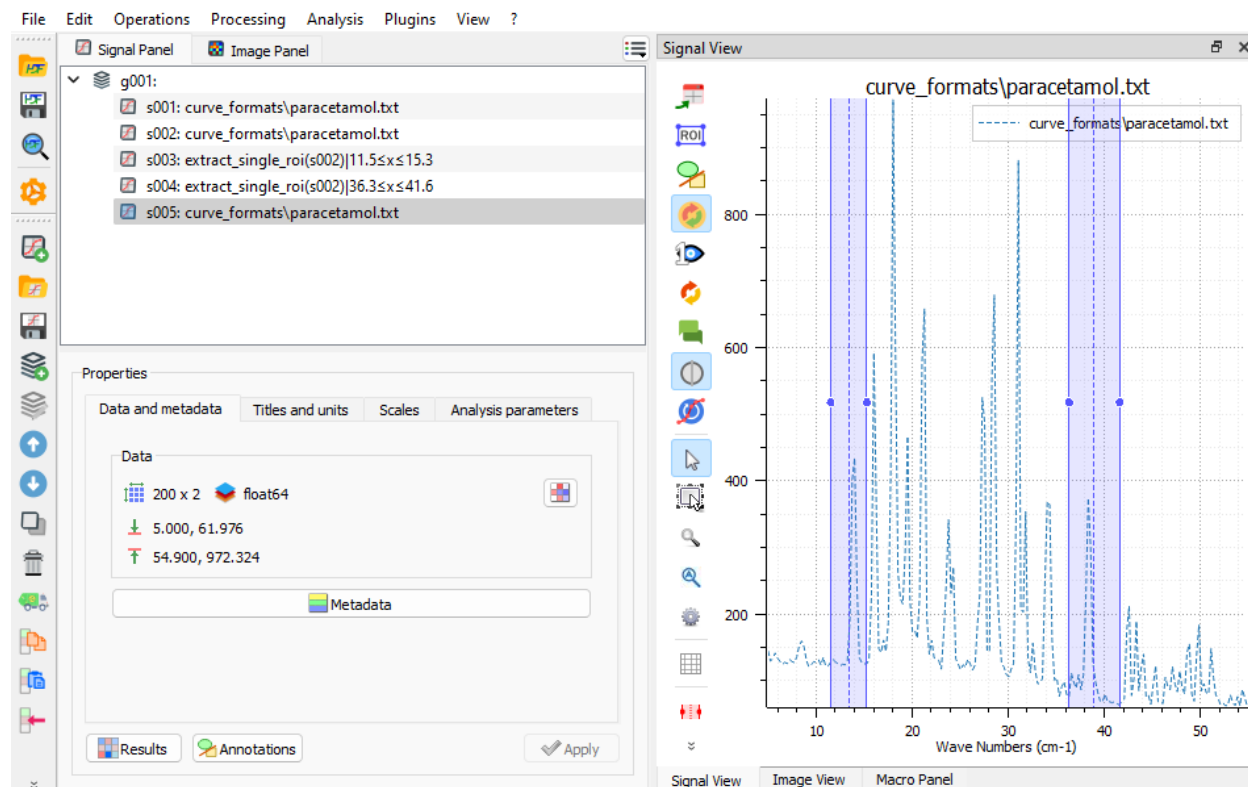


Fig. 16: A signal with an ROI.

- “Remove regions of interest” : remove all defined ROI for the selected signals.

### 2.3.3 Operations on Signals

This section describes the operations that can be performed on signals.

**See also:**

[Processing Signals](#) for more information on signal processing features, or [Analysis features on Signals](#) for information on analysis features on signals.

When the “Signal Panel” is selected, the menus and toolbars are updated to provide signal-related actions.

The “Operations” menu allows you to perform various operations on the selected signals, such as arithmetic operations, peak detection, or convolution.

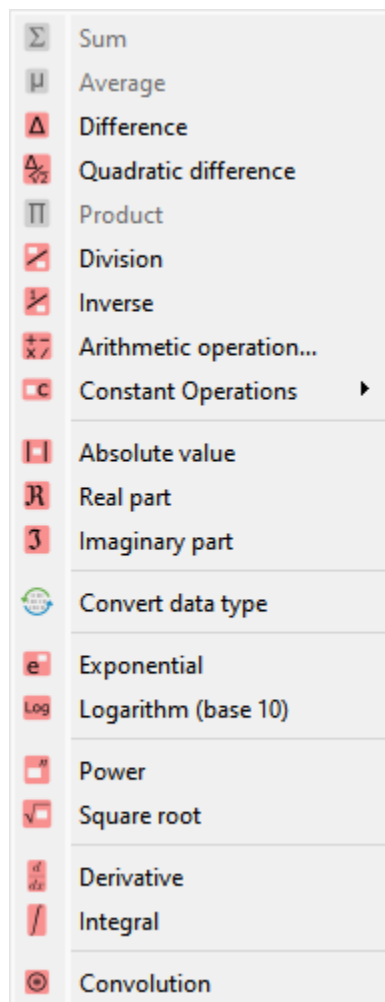


Fig. 17: Screenshot of the “Operations” menu.

## Basic arithmetic operations

Operation	Description
Sum	$y_M = \sum_{k=0}^{M-1} y_k$
Average	$y_M = \frac{1}{M} \sum_{k=0}^{M-1} y_k$
Difference	$y_2 = y_1 - y_0$
Product	$y_M = \prod_{k=0}^{M-1} y_k$
Division	$y_2 = \frac{y_1}{y_0}$
Inverse	$y_2 = \frac{1}{y_1}$

## Operations with a constant

Create a new signal which is the result of a constant operation on each selected signal:

Operation	Description
Addition	$y_k = y_{k-1} + c$
Subtraction	$y_k = y_{k-1} - c$
Multiplication	$y_k = y_{k-1} \times c$
Division	$y_k = \frac{y_{k-1}}{c}$

## Absolute value, real and imaginary parts

Operation	Description
Absolute value	$y_k =  y_{k-1} $
Real part	$y_k = \Re(y_{k-1})$
Imaginary part	$y_k = \Im(y_{k-1})$

## Data type conversion

The “Convert data type” action allows you to convert the data type of the selected signals.

**Note:** Data type conversion relies on `numpy.ndarray.astype()` function with the default parameters (`casting='unsafe'`).

## Basic mathematical functions

Function	Description
Exponential	$y_k = \exp(y_{k-1})$
Logarithm (base 10)	$y_k = \log_{10}(y_{k-1})$
Power	$y_k = y_{k-1}^n$
Square root	$y_k = \sqrt{y_{k-1}}$

## Other mathematical operations

Operation	Implementation
Derivative	Based on <code>numpy.gradient</code>
Integral	Based on <code>scipy.integrate.cumulative_trapezoid</code>
Convolution	Based on <code>scipy.signal.convolve</code>

## 2.3.4 Processing Signals

This section describes the signal processing features available in DataLab.

### See also:

[Operations on Signals](#) for more information on operations that can be performed on signals, or [Analysis features on Signals](#) for information on analysis features on signals.

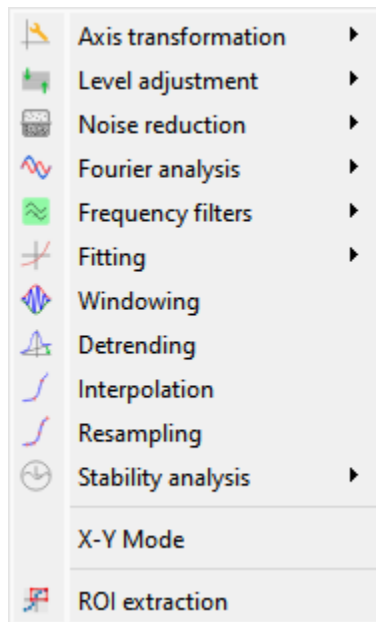


Fig. 18: Screenshot of the “Processing” menu.

When the “Signal Panel” is selected, the menus and toolbars are updated to provide signal-related actions.

The “Processing” menu allows you to perform various processing on the selected signals, such as smoothing, normalization, or interpolation.

## Axis transformation

### Linear calibration

Create a new signal which is a linear calibration of each selected signal with respect to X or Y axis:

Parameter	Linear calibration
X-axis	$x_1 = a.x_0 + b$
Y-axis	$y_1 = a.y_0 + b$

### Swap X/Y axes

Create a new signal which is the result of swapping X/Y data.

### Reverse X-axis

Create a new signal which is the result of reversing X data.

### Convert to Cartesian coordinates

Create a new signal which is the result of converting polar coordinates to Cartesian coordinates.

---

**Note:** This function assumes that the x-axis represents the radius and the y-axis represents the angle. Negative values are not allowed for the radius, and will be clipped to 0 (a warning will be raised).

---

### Convert to polar coordinates

Create a new signal which is the result of converting Cartesian coordinates to polar coordinates.

### Level adjustment

#### Normalize

Create a new signal which is the normalization of each selected signal by maximum, amplitude, sum, energy or RMS:

Parameter	Normalization
Maximum	$y_1 = \frac{y_0}{\max(y_0)}$
Amplitude	$y_1 = \frac{y'_0}{\max(y'_0)}$ with $y'_0 = y_0 - \min(y_0)$
Area	$y_1 = \frac{y_0}{\sum_{n=0}^N y_0[n]}$
Energy	$y_1 = \frac{y_0}{\sqrt{\sum_{n=0}^N  y_0[n] ^2}}$
RMS	$y_1 = \frac{y_0}{\sqrt{\frac{1}{N} \sum_{n=0}^N  y_0[n] ^2}}$

## Clipping

Create a new signal which is the result of clipping each selected signal.

## Offset correction

Create a new signal which is the result of offset correction of each selected signal. This operation is performed by subtracting the signal baseline which is estimated by the mean value of a user-defined range.

## Noise reduction

Create a new signal which is the result of noise reduction of each selected signal.

The following filters are available:

Filter	Formula/implementation
Gaussian filter	<code>scipy.ndimage.gaussian_filter</code>
Moving average	<code>scipy.ndimage.uniform_filter</code>
Moving median	<code>scipy.ndimage.median_filter</code>
Wiener filter	<code>scipy.signal.wiener</code>

## Fourier analysis

### Zero padding

Create a new signal which is the result of zero padding of each selected signal.

The following parameters are available:

Parameter	Description
Strategy	Zero padding strategy (see below)
Number of points	Custom length (if <i>strategy</i> is “custom”)

Zero padding strategy refers to the method used to increase the length of the signal, and it can be one of the following:

Strategy	Description
next_pow2	Next power of 2 (e.g. 1024 → 2048)
double	Double the length (e.g. 1024 → 2048)
triple	Triple the length (e.g. 1024 → 3072)
custom	Custom length (user-defined)

## FFT related functions

Create a new signal which is the result of a Fourier analysis of each selected signal.

The following functions are available:

Function	Description	Formula/implementation
FFT	Fast Fourier Transform	<code>numpy.fft.fft</code>
Inverse FFT	Inverse Fast Fourier Transform	<code>numpy.fft.ifft</code>
Magnitude spectrum	Optionnal: use logarithmic scale (dB)	$y_1 =  FFT(y_0) $ or $20 \cdot \log_{10}( FFT(y_0) )$ (dB)
Phase spectrum	Phase of the FFT expressed in degrees, using <code>numpy.angle</code> function	$y_1 = \angle FFT(y_0)$
Power spectral density (PSD)	Optionnal: use logarithmic scale (dB). PSD is estimated using Welch's method (see <code>scipy.signal.welch</code> )	$Y_k = PSD(y_k)$ or $10 \cdot \log_{10}(PSD(y_k))$ (dB)

**Note:** FFT and inverse FFT are performed using frequency shifting if the option is enabled in DataLab settings (see *Settings*).

## Frequency filters

Create a new signal which is the result of applying a frequency filter to each selected signal.

The following filters are available:

Filter	Description
Low-pass	Filter out high frequencies, above a cutoff frequency
High-pass	Filter out low frequencies, below a cutoff frequency
Band-pass	Filter out frequencies outside a range
Band-stop	Filter out frequencies inside a range

For each filter, the following methods are available:

Method	Description
Bessel	Bessel filter, using SciPy's <code>scipy.signal.bessel</code> function
Butterworth	Butterworth filter, using SciPy's <code>scipy.signal.butter</code> function
Chebyshev I	Chebyshev type I filter, using SciPy's <code>scipy.signal.cheby1</code> function
Chebyshev II	Chebyshev type II filter, using SciPy's <code>scipy.signal.cheby2</code> function
Elliptic	Elliptic filter, using SciPy's <code>scipy.signal.ellip</code> function

## Fitting

Open an interactive curve fitting tool in a modal dialog box.

Model	Equation
Linear	$y = c_0 + c_1.x$
Polynomial	$y = c_0 + c_1.x + c_2.x^2 + \dots + c_n.x^n$
Gaussian	$y = y_0 + \frac{A}{\sqrt{2\pi}.\sigma} .exp(-\frac{1}{2} . (\frac{x - x_0}{\sigma})^2)$
Lorentzian	$y = y_0 + \frac{A}{\sigma.\pi} . \frac{1}{1 + (\frac{x - x_0}{\sigma})^2}$
Voigt	$y = y_0 + A . \frac{Re(exp(-z^2) . erf(-j.z))}{\sqrt{2\pi}.\sigma}$ with $z = \frac{x - x_0 - j.\sigma}{\sqrt{2}.\sigma}$
Multi-Gaussian	$y = y_0 + \sum_{i=0}^K \frac{A_i}{\sqrt{2\pi}.\sigma_i} .exp(-\frac{1}{2} . (\frac{x - x_{0,i}}{\sigma_i})^2)$
Exponential	$y = y_0 + A.exp(B.x)$
Sinusoidal	$y = y_0 + A.sin(2\pi.f.x + \phi)$
Cumulative Distribution Function (CDF)	$y = y_0 + A.erf(\frac{x - x_0}{\sigma.\sqrt{2}})$

## Windowing

Create a new signal which is the result of applying a window function to each selected signal.

The following window functions are available:



Window function	Reference
Barthann	<code>scipy.signal.windows.barthann()</code>
Bartlett	<code>numpy.bartlett()</code>
Blackman	<code>scipy.signal.windows.blackman()</code>
Blackman-Harris	<code>scipy.signal.windows.blackmanharris()</code>
Bohman	<code>scipy.signal.windows.bohman()</code>
Boxcar	<code>scipy.signal.windows.boxcar()</code>
Cosine	<code>scipy.signal.windows.cosine()</code>
Exponential	<code>scipy.signal.windows.exponential()</code>
Flat top	<code>scipy.signal.windows.flattop()</code>
Gaussian	<code>scipy.signal.windows.gaussian()</code>
Hamming	<code>numpy.hamming()</code>
Hanning	<code>numpy.hanning()</code>
Kaiser	<code>scipy.signal.windows.kaiser()</code>
Lanczos	<code>scipy.signal.windows.lanczos()</code>
Nuttall	<code>scipy.signal.windows.nuttall()</code>
Parzen	<code>scipy.signal.windows.parzen()</code>
Rectangular	<code>numpy.ones()</code>
Taylor	<code>scipy.signal.windows.taylor()</code>
Tukey	<code>scipy.signal.windows.tukey()</code>

## Detrending

Create a new signal which is the detrending of each selected signal. This features is based on SciPy's `scipy.signal.detrend` function.

The following parameters are available:

Parameter	Description
Method	Detrending method: 'linear' or 'constant'. See SciPy's <code>scipy.signal.detrend</code> function.

## Interpolation

Create a new signal which is the interpolation of each selected signal with respect to a second signal X-axis (which might be the same as one of the selected signals).

The following interpolation methods are available:

Method	Description
Linear	Linear interpolation, using using NumPy's <code>interp</code> function
Spline	Cubic spline interpolation, using using SciPy's <code>scipy.interpolate.splev</code> function
Quadratic	Quadratic interpolation, using using NumPy's <code>polyval</code> function
Cubic	Cubic interpolation, using using SciPy's <code>Akima1DInterpolator</code> class
Barycentric	Barycentric interpolation, using using SciPy's <code>BarycentricInterpolator</code> class
PCHIP	Piecewise Cubic Hermite Interpolating Polynomial (PCHIP) interpolation, using using SciPy's <code>PchipInterpolator</code> class

## Resampling

Create a new signal which is the resampling of each selected signal.

The following parameters are available:

Parameter	Description
Method	Interpolation method (see previous section)
Fill value	Interpolation fill value (see previous section)
Xmin	Minimum X value
Xmax	Maximum X value
Mode	Resampling mode: step size or number of points
Step size	Resampling step size
Number of points	Resampling number of points

## Stability analysis

Create a new signal which is the result of a stability analysis of each selected signal.

The following stability analysis methods are available:

Function	Description
Allan variance	Measure of the stability of a signal: defined as the variance of the difference between two successive measurements as a function of the time interval between them.
Allan deviation	Square root of the Allan variance.
Overlapping Allan deviation	A more robust version of the Allan variance that overlaps successive segments to improve statistical confidence.
Modified Allan variance	A variation of the Allan variance that accounts for phase noise by introducing a filtering operation.
Hadamard variance	An alternative to Allan variance, more robust to linear frequency drift in the signal
Total variance	Extends the Allan variance concept to cover all possible averaging intervals.
Time deviation	Derived from Allan deviation, quantifies stability in terms of time rather than frequency.

---

**Note:** The “All stability features” option allows to compute all stability analysis methods at once.

---

## X-Y Mode

Simulate the X-Y mode of an oscilloscope.

## ROI extraction

Create a new signal from a user-defined Region of Interest (ROI).

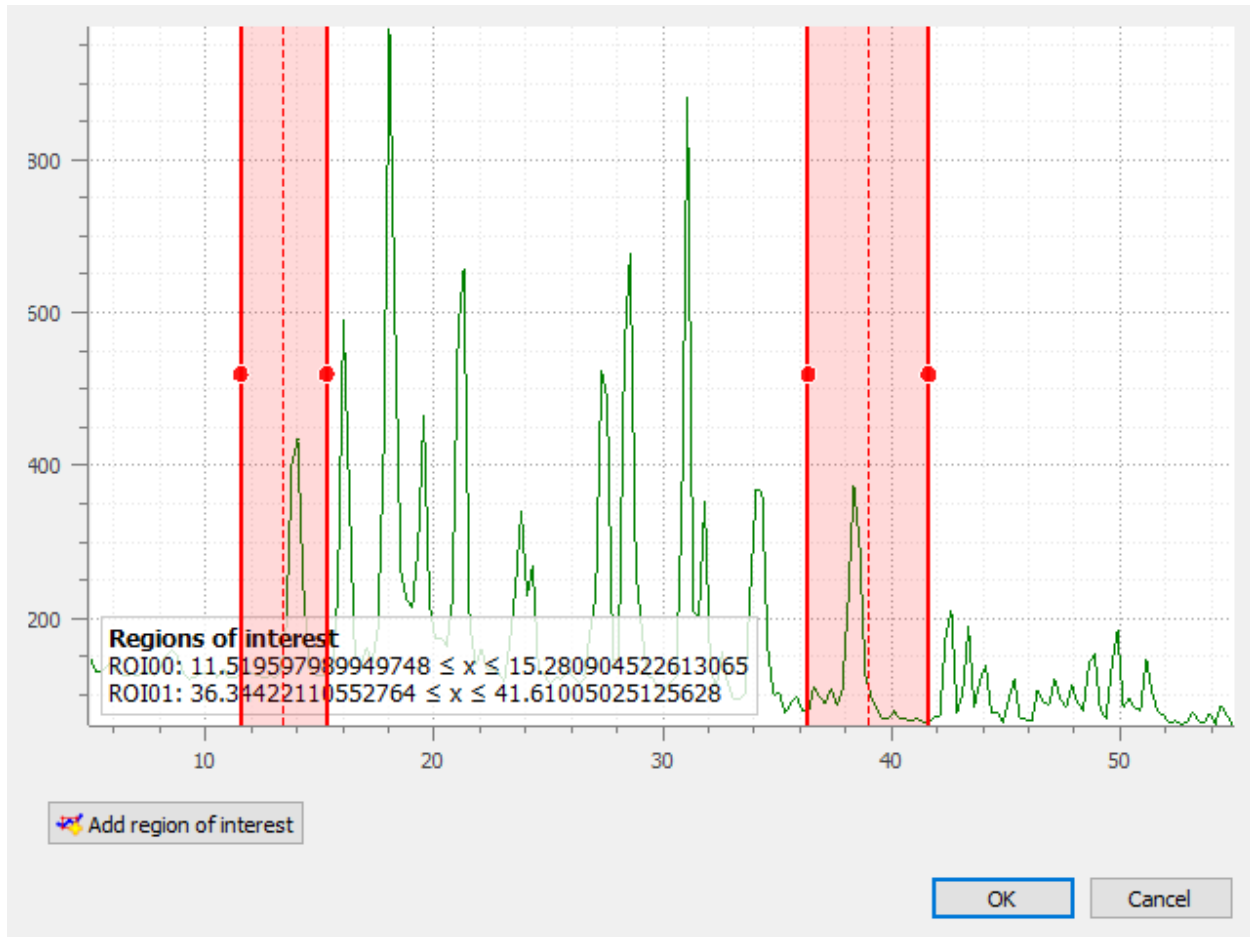


Fig. 19: ROI extraction dialog: the ROI is defined by moving the position and adjusting the width of an horizontal range.

## 2.3.5 Analysis features on Signals

This section describes the signal analysis features available in DataLab.

### See also:

[Operations on Signals](#) for more information on operations that can be performed on signals, or [Processing Signals](#) for information on processing features on signals.

When the “Signal Panel” is selected, the menus and toolbars are updated to provide signal-related actions.

The “Analysis” menu allows you to perform various computations on the selected signals, such as statistics, full width at half-maximum, or full width at  $1/e^2$ .

**Note:** In DataLab vocabulary, an “analysis” is a feature that computes a scalar result from a signal. This result is stored as metadata, and thus attached to signal. This is different from a “processing” which creates a new signal from

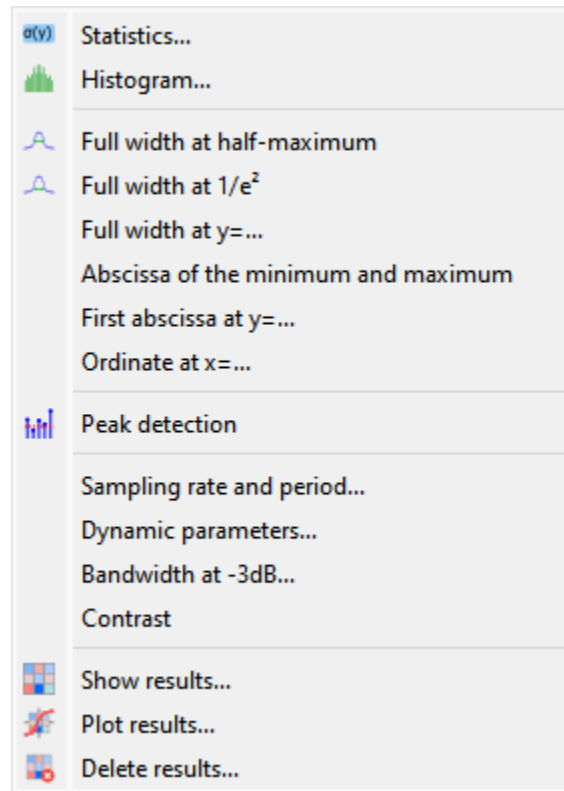


Fig. 20: Screenshot of the “Analysis” menu.

an existing one.

---

## Statistics

Compute statistics on selected signal and show a summary table.

## Histogram

Compute histogram of selected signal and show it.

Parameters are:

Parameter	Description
Bins	Number of bins
Lower limit	Lower limit of the histogram
Upper limit	Upper limit of the histogram

	min(y)	max(y)	<y>	$\sigma(y)$	$\Sigma(y)$	$\int y dx$
s000	7.6946e-23	0.398862	0.0499	0.107641	24.95	1
s000 ROI00	1.1479e-22	0.398862	0.0501004	0.10781	12.475	0.492007

Format

Resize

☒ Background color

Close

Fig. 21: Example of statistical summary table: each row is associated to an ROI (the first row gives the statistics for the whole data).

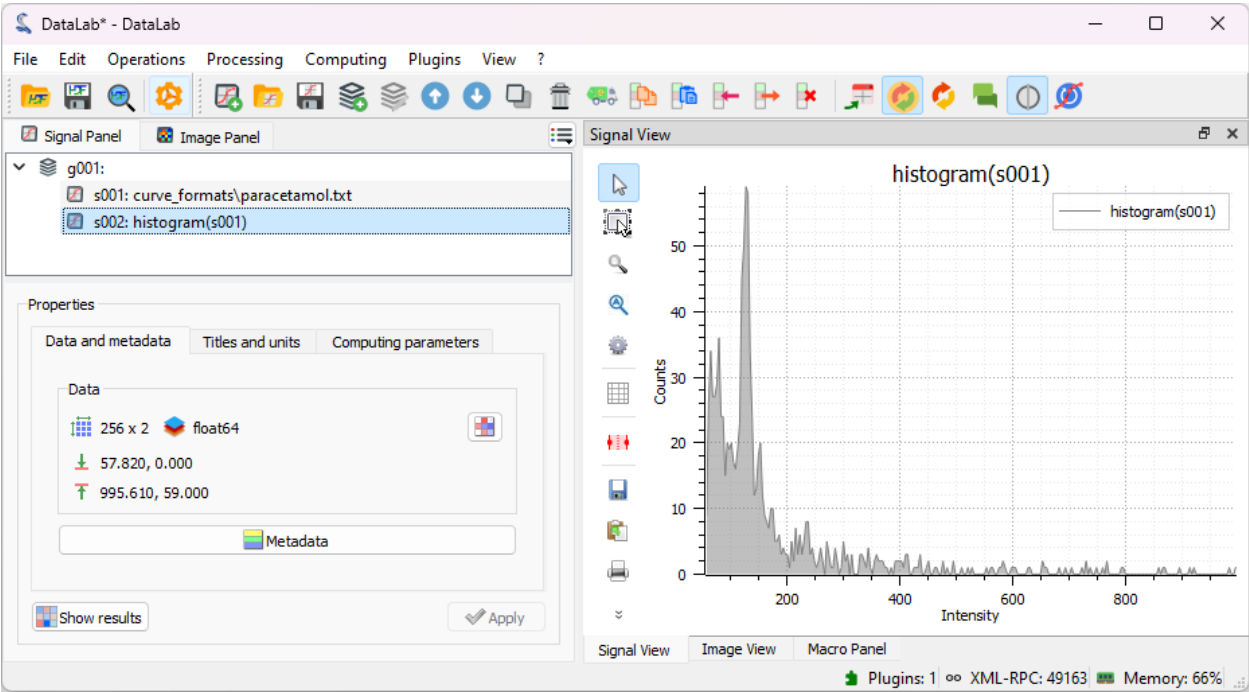


Fig. 22: Example of histogram.

## Full width at half-maximum

Compute the Full Width at Half-Maximum (FWHM) of selected signal, using one of the following methods:

Method	Description
Zero-crossing	Find the zero-crossings of the signal after having centered its amplitude around zero
Gauss	Fit data to a Gaussian model using least-square method
Lorentz	Fit data to a Lorentzian model using least-square method
Voigt	Fit data to a Voigt model using least-square method

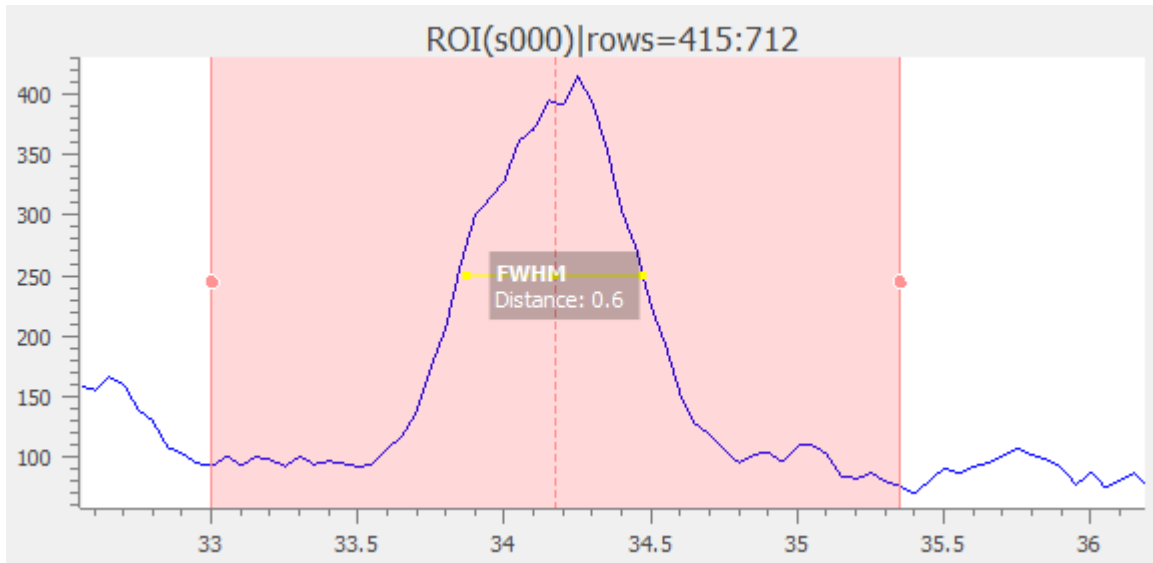


Fig. 23: The computed result is displayed as an annotated segment.

## Full width at $1/e^2$

Fit data to a Gaussian model using least-square method. Then, compute the full width at  $1/e^2$ .

**Note:** Computed scalar results are systematically stored as metadata. Metadata is attached to signal and serialized with it when exporting current session in a HDF5 file.

## Arguments of the min and max

Compute the smallest argument of the minima and the smallest argument of the maxima of the selected signal.

### Abscissa at y=...

Compute the abscissa at a given ordinate value for the selected signal. If there is no solution, the displayed result is NaN. If there are multiple solutions, the displayed result is the smallest value.

### Peak detection

Create a new signal from semi-automatic peak detection of each selected signal.

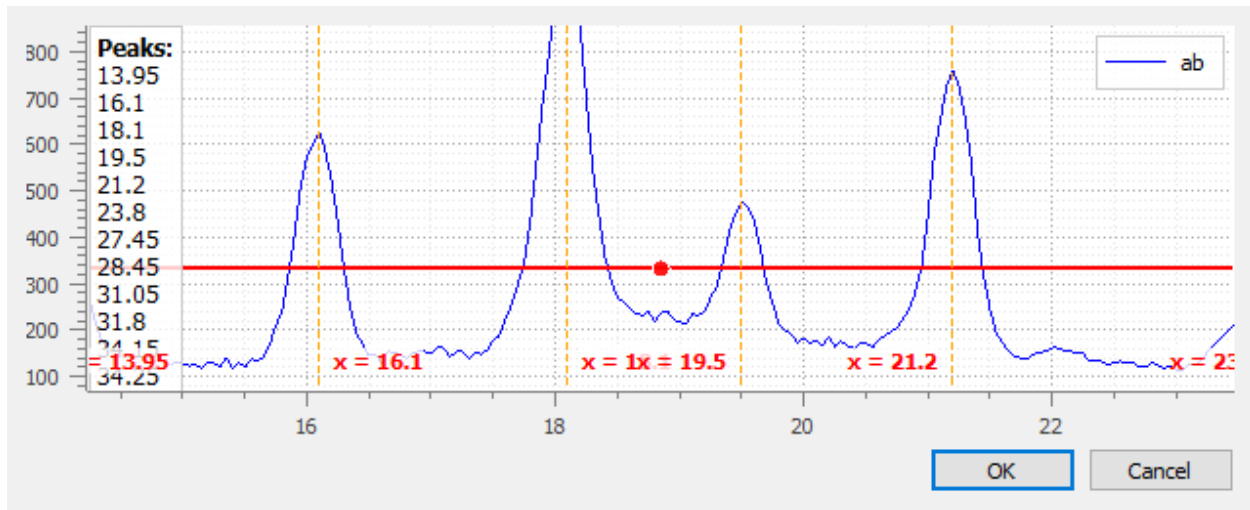


Fig. 24: Peak detection dialog: threshold is adjustable by moving the horizontal marker, peaks are detected automatically (see vertical markers with labels indicating peak position)

### Sampling rate and period

Compute the sampling rate and period of selected signal.

**Warning:** This feature assumes that the X values are regularly spaced.

### Dynamic parameters

Compute the following dynamic parameters on selected signal:

Parameter	Description
f	Frequency (sinusoidal fit)
ENOB	Effective Number Of Bits
SNR	Signal-to-Noise Ratio
SINAD	Signal-to-Noise And Distortion Ratio
THD	Total Harmonic Distortion
SFDR	Spurious-Free Dynamic Range

## Bandwidth at -3 dB

Assuming the signal is a filter response, compute the bandwidth at -3 dB by finding the frequency range where the signal is above -3 dB.

**Warning:** This feature assumes that the signal is a filter response, already expressed in dB.

## Contrast

Compute the contrast of selected signal.

The contrast is defined as the ratio of the difference and the sum of the maximum and minimum values:

$$\text{Contrast} = \frac{\max(y) - \min(y)}{\max(y) + \min(y)}$$

---

**Note:** This feature assumes that the signal is a profile from an image, where the contrast is meaningful. This justifies the optical definition of contrast.

---

## Show results

Show the results of all analyses performed on the selected signals. This shows the same table as the one shown after having performed a computation.

## Plot results

Plot the results of analyses performed on the selected signals, with user-defined X and Y axes (e.g. plot the FWHM as a function of the signal index).

## 2.3.6 View options for Signals

When the “Signal Panel” is selected, the menus and toolbars are updated to provide signal-related actions.

The “View” menu allows you to visualize the current signal or group of signals. It also allows you to show/hide titles, to enable/disable anti-aliasing, or to refresh the visualization.

### View in a new window

Open a new window to visualize the selected signals.

This option allows you to visualize the selected signals in a separate window, in which you can visualize the data more comfortably (e.g., by maximizing the window) and you can also annotate the data.

When you click on the button “Annotations” in the toolbar of the new window, the annotation editing mode is activated (see the section “Edit annotations” below).



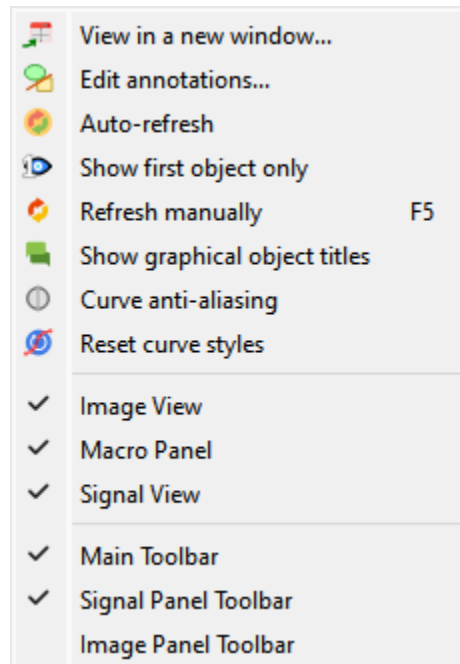


Fig. 25: Screenshot of the “View” menu.

## Edit annotations

Open a new window to edit annotations.

This option allows you to show the selected signals in a separate window, in which you can visualize the data more comfortably (e.g., by maximizing the window) as well as to add or edit annotations.

Annotations are used to add comments or labels to the data, and they can be used to highlight specific events or regions of interest.

---

**Note:** The annotations are saved in the metadata of the signal, so they are persistent and will be displayed every time you visualize the signal.

---

The typical workflow to edit annotations is as follows:

1. Select the “Edit annotations” option.
2. In the new window, add annotations by clicking on the corresponding buttons at the bottom of the window.
3. Eventually, customize the annotations by changing their properties (e.g., the text, the color, the position, etc.) using the “Parameters” option in the context menu of the annotations.
4. When you are done, click on the “OK” button to save the annotations. This will close the window and the annotations will be saved in the metadata of the signal and will be displayed in the main window.

---

**Note:** Annotations may be copied from a signal to another by using the “copy/paste metadata” features.

---

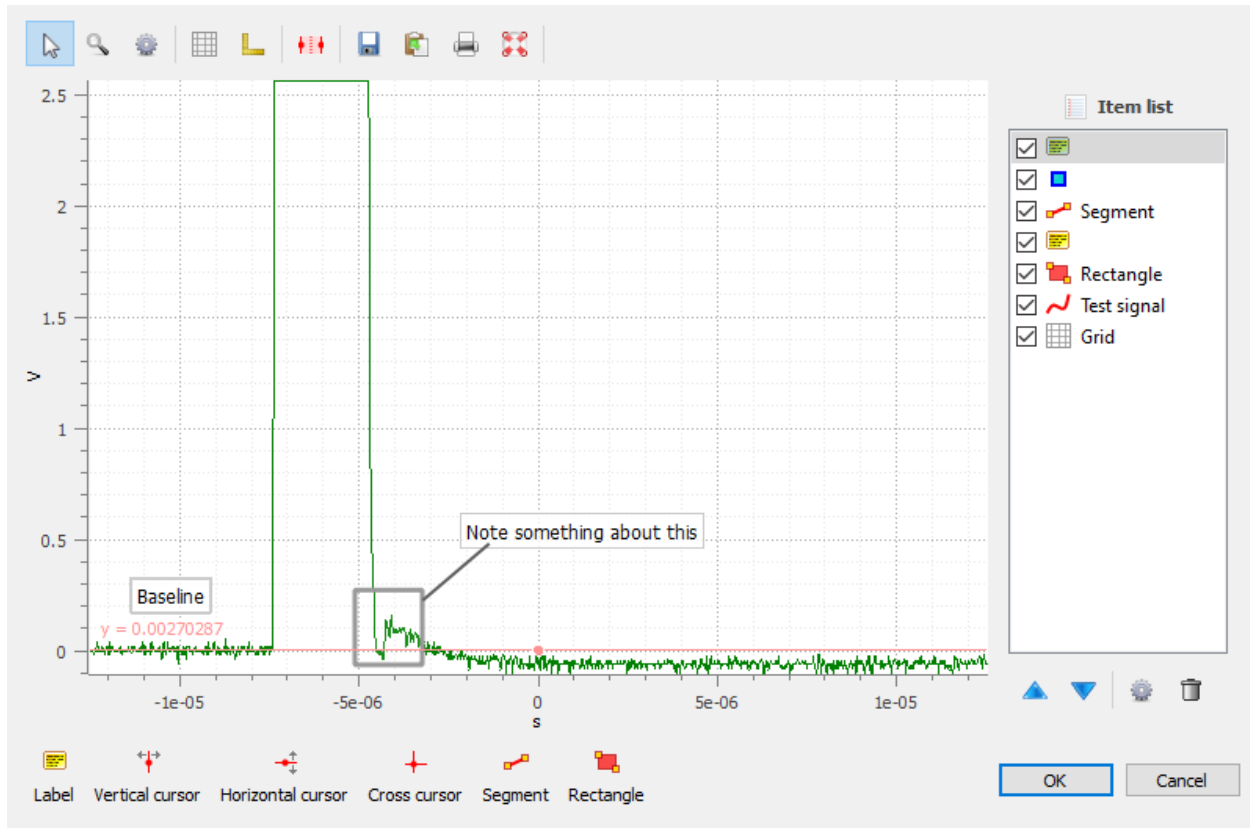


Fig. 26: Annotations may be added in the separate view.

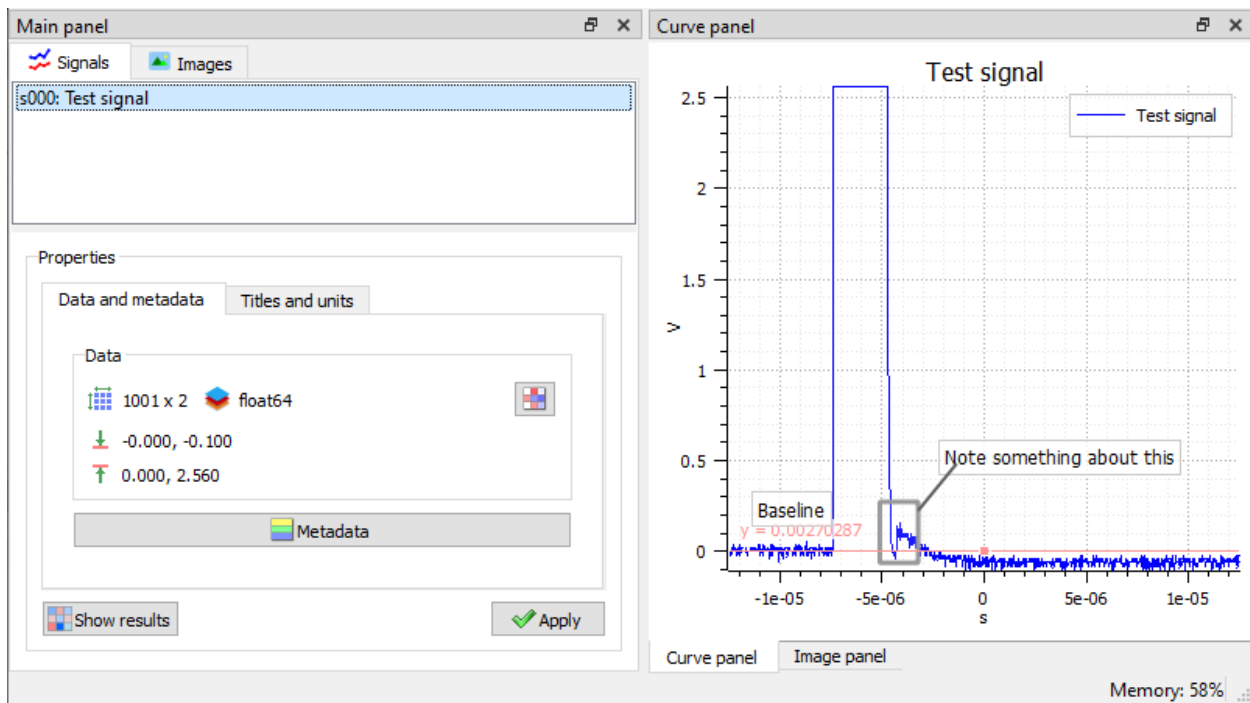


Fig. 27: Annotations are now part of the signal metadata.

## Auto-refresh

Automatically refresh the visualization when the data changes:

- When enabled (default), the plot view is automatically refreshed when the data changes.
- When disabled, the plot view is not refreshed until you manually refresh it by using the “Refresh manually” menu entry.

Even though the refresh algorithm is optimized, it may still take some time to refresh the plot view when the data changes, especially when the data set is large. Therefore, you may want to disable the auto-refresh feature when you are working with large data sets, and enable it again when you are done. This will avoid unnecessary refreshes.

## Refresh manually

Refresh the visualization manually.

This triggers a refresh of the plot view. It is useful when the auto-refresh feature is disabled, or when you want to force a refresh of the plot view.

## Show graphical object titles

Show/hide titles of analysis results or annotations.

This option allows you to show or hide the titles of the graphical objects (e.g., the titles of the analysis results or annotations). Hiding the titles can be useful when you want to visualize the data without any distractions, or if there are too many titles and they are overlapping.

## Curve anti-aliasing

Enable/disable anti-aliasing of curves.

Anti-aliasing makes the curves look smoother, but it may also make them look less sharp.

---

**Note:** Anti-aliasing is enabled by default.

---

**Warning:** Anti-aliasing may slow down the visualization, especially when working with large data sets.

## Reset curve styles

Reset the curve style cycle to the beginning.

When plotting curves, DataLab automatically assigns a color and a line style to each curve. Both parameters are chosen from a predefined list of colors and line styles, and are assigned in a round-robin fashion.

This menu entry allows you to reset the curve styles, so that the next time you plot curves, the first curve will be assigned the first color and the first line style of the predefined lists, and the loop will start again from there.

## 2.4 Image processing

This section describes the features specific to the image processing panel. The image processing panel can be selected by clicking on the “Images” tab at the bottom-right of the DataLab main window.

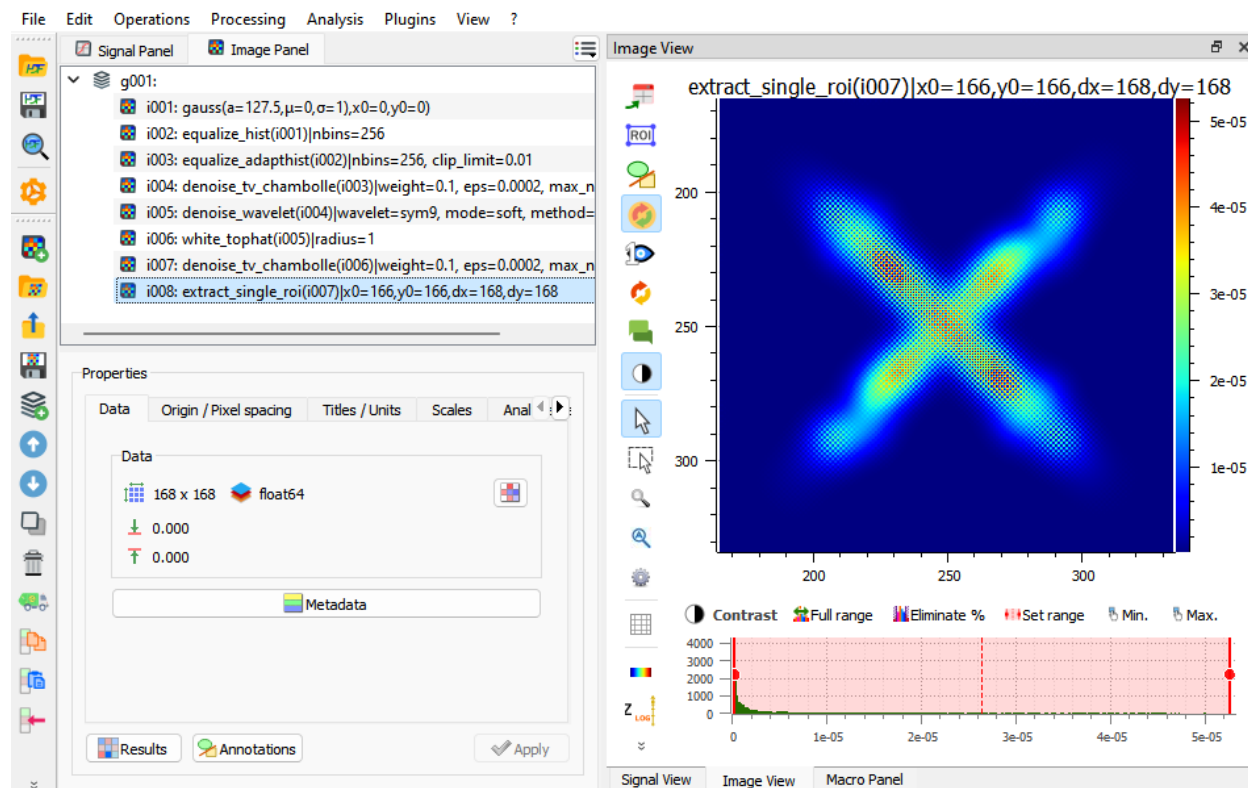


Fig. 28: DataLab main window: Image processing view

### 2.4.1 Create, open and save Images

This section describes how to create, open and save images (and workspaces).

When the “Image Panel” is selected, the menus and toolbars are updated to provide image-related actions.

The “File” menu allows you to:

- Create, open, save and close images (see below).
- Save and restore the current workspace or browse HDF5 files (see [Workspace](#)).
- Edit DataLab preferences (see [Settings](#)).

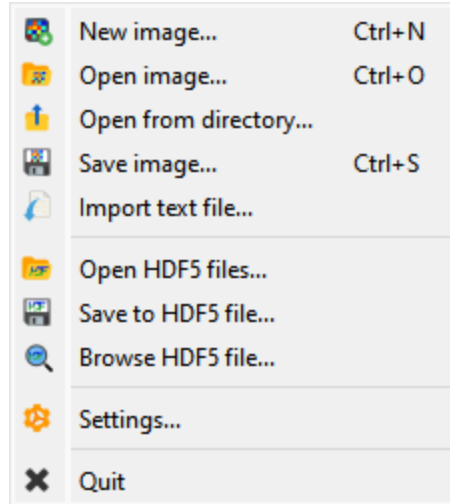


Fig. 29: Screenshot of the “File” menu.

## New image

Create a new image from various models (supported datatypes: uint8, uint16, int16, float32, float64):

Model	Equation
Zeros	$z[i] = 0$
Empty	Data is directly taken from memory as it is
Random	$z[i] \in [0, z_{max})$ where $z_{max}$ is the datatype maximum value
2D Gaussian	$z = A.exp(-\frac{(\sqrt{(x-x_0)^2 + (y-y_0)^2} - \mu)^2}{2\sigma^2})$

## Open image

Create a new image from the following supported filetypes:

File type	Extensions
PNG files	.png
TIFF files	.tif, .tiff
8-bit images	.jpg, .gif
NumPy arrays	.npy
MAT-Files	.mat
Text files	.txt, .csv, .asc
Andor SIF files	.sif
Princeton Instruments SPE files	.spe
Opticks GEL files	.gel
Hamamatsu NDPI files	.ndpi
PCO Camera REC files	.rec
SPIRICON files	.scor-data
FXD files	.fxd
Bitmap images	.bmp

**Note:** DataLab also supports any image format that can be read by the *imageio* library, provided that the associated plugin(s) are installed (see [imageio documentation](#)) and that the output NumPy array data type and shape are supported by DataLab.

To add a new file format, you may use the *imageio\_formats* entry of DataLab configuration file. This entry is a formatted like the *IMAGEIO\_FORMATS* object which represents the natively supported formats:

```
cdl.config.IMAGEIO_FORMATS = (('*.gel', 'Opticks GEL'), ('*.spe', 'Princeton Instruments SPE'), ('*.ndpi', 'Hamamatsu Slide Scanner NDPI'), ('*.rec', 'PCO Camera REC'))
```

---

## Save image

Save current image (see “Open image” supported filetypes).

## Import text file

DataLab can natively import many types of image files (e.g. TIFF, JPEG, PNG, etc.). However some specific text file formats may not be supported. In this case, you can use the *Import text file* feature, which allows you to import a text file and convert it to an image.

This feature is accessible from the *File* menu, under the *Import text file* option.

It opens an import wizard that guides you through the process of importing the text file.

### Step 1: Select the source

The first step is to select the source of the text file. You can either select a file from your computer or the clipboard if you have copied the text from another application.

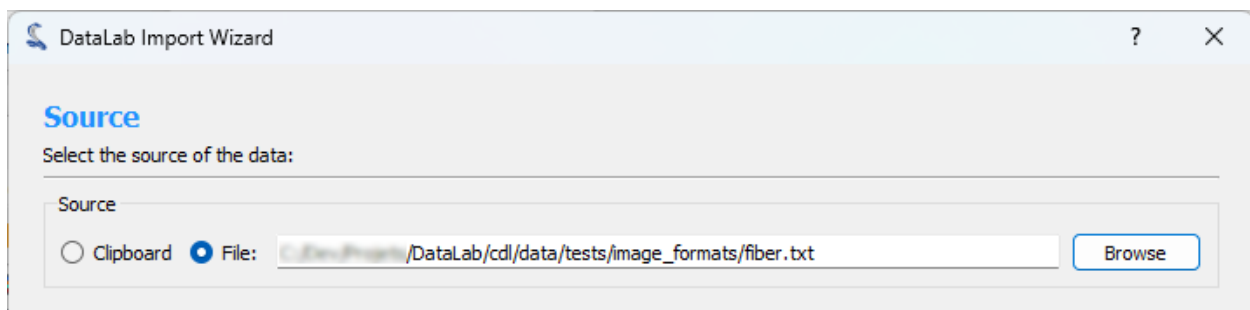


Fig. 30: Step 1: Select the source

## Step 2: Preview and configure the import

The second step consists of configuring the import and previewing the result. You can configure the following options:

- **Delimiter:** The character used to separate the values in the text file.
- **Comments:** The character used to indicate that the line is a comment and should be ignored.
- **Rows to Skip:** The number of rows to skip at the beginning of the file.
- **Maximum Number of Rows:** The maximum number of rows to import. If the file contains more rows, they will be ignored.
- **Transpose:** If checked, the rows and columns will be transposed.
- **Data type:** The destination data type of the imported data.

When you are done configuring the import, click the *Apply* button to see the result.

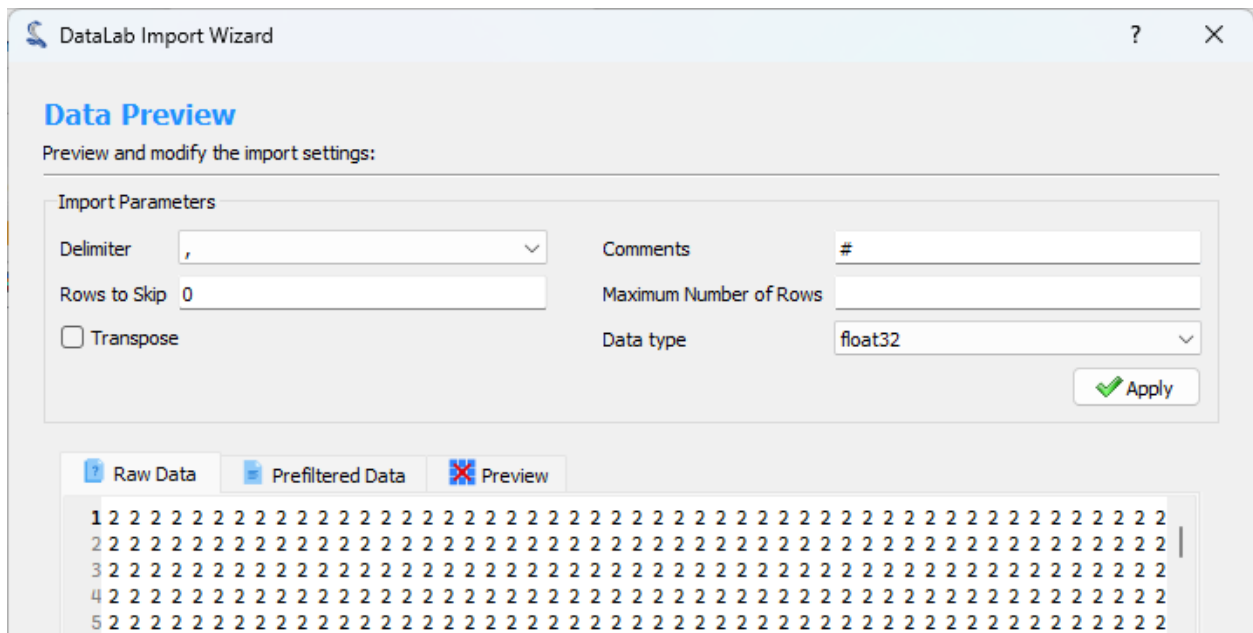


Fig. 31: Step 2: Configure the import

## Step 3: Show graphical representation

The third step shows a graphical representation of the imported data. You can use the *Finish* button to import the data into DataLab workspace.

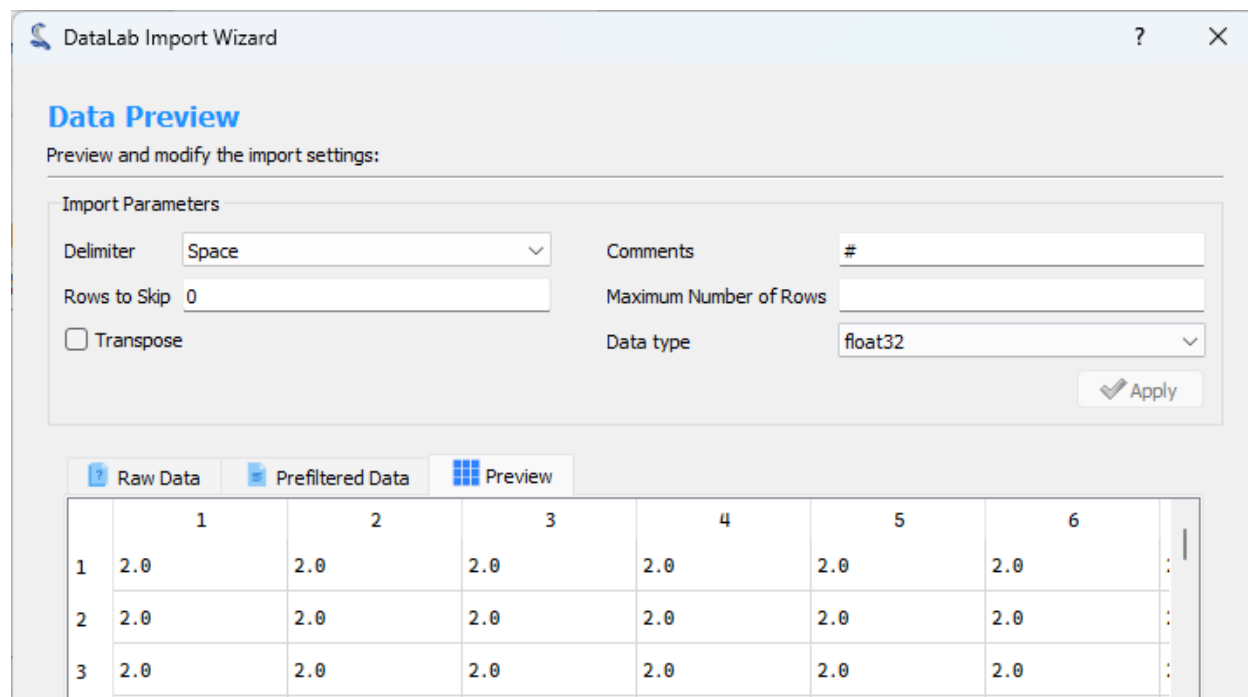


Fig. 32: Step 2: Preview the result

## 2.4.2 Manipulate metadata

This section describes how to manipulate metadata in DataLab.

The “Edit” menu allows you to perform classic editing operations on the current image or group of images (create/rename group, move up/down, delete image/group of images, etc.).

It also allows you to manipulate metadata associated with the current image.

### Copy/paste metadata

As metadata contains useful information about the image, it can be copied and pasted from one image to another by selecting the “Copy metadata” and “Paste metadata” actions in the “Edit” menu.

This feature allows you to transfer that information from one image to another:

- *Regions Of Interest (ROIs)*: that is a very efficient way to reuse the same ROI on different images and easily compare the results of the analysis on those images
- Analyze results, such as a centroid position or a contour detection (the relevance of transferring such information depends on the context and is up to the user to decide)
- Any other information that you may have added to the metadata of an image

**Note:** Copying metadata from an image to another will overwrite the metadata of the destination image (for the metadata keys that are common to both images) or simply add the metadata keys that are not present in the destination image.



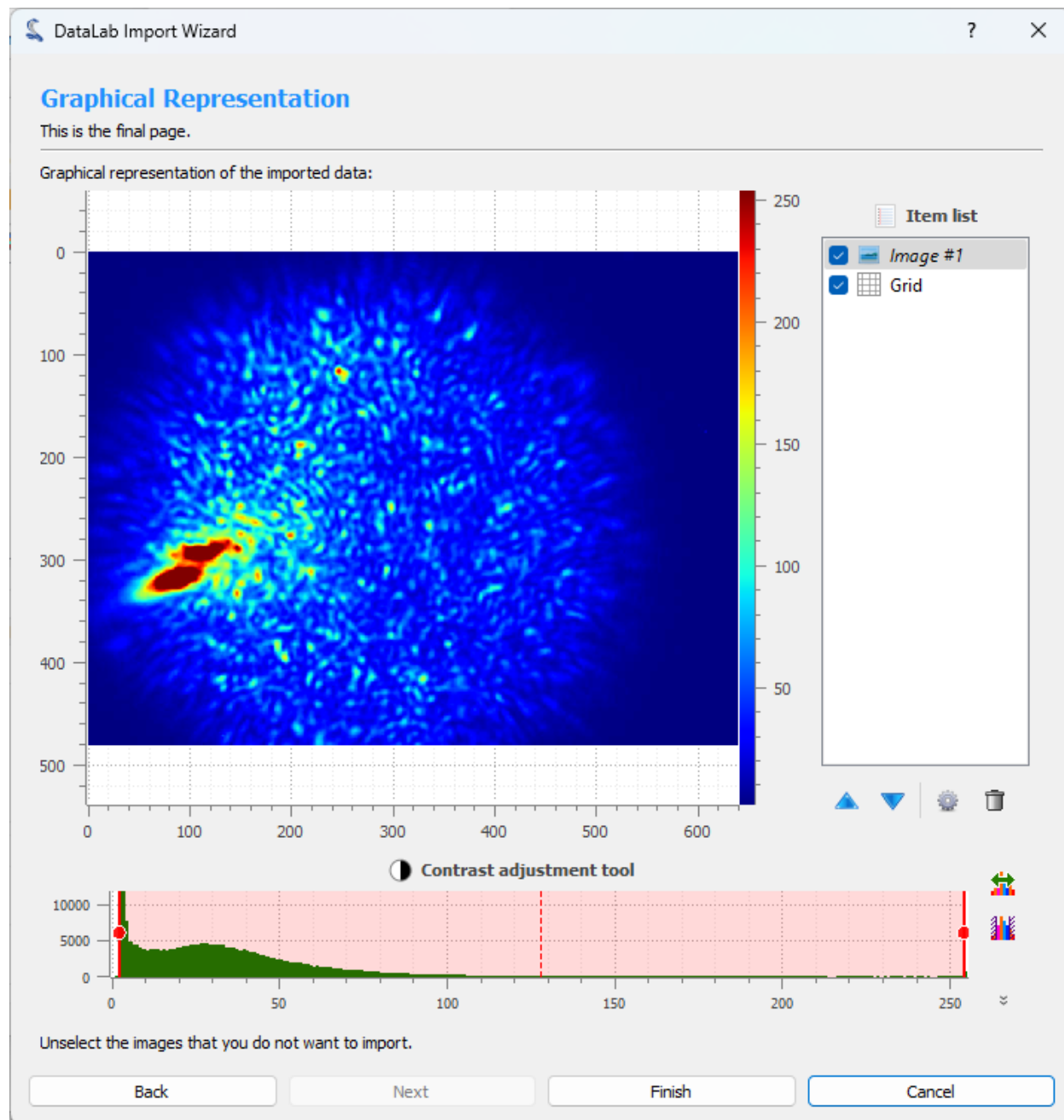


Fig. 33: Step 3: Show graphical representation

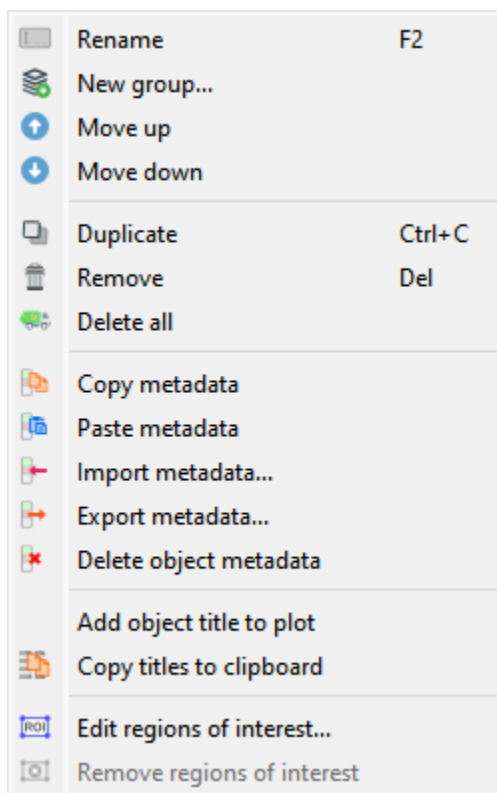


Fig. 34: Screenshot of the “Edit” menu.

## Import/export metadata

Metadata can also be imported and exported from/to a JSON file using the “Import metadata” and “Export metadata” actions in the “Edit” menu. This is exactly the same as the copy/paste metadata feature (see above for more details on the use cases of this feature), but it allows you to save the metadata to a file and then import it back later.

## Delete metadata

When deleting metadata using the “Delete metadata” action in the “Edit” menu, you will be prompted to confirm the deletion of Region of Interests (ROIs) if they are present in the metadata. After this eventual confirmation, the metadata will be deleted, meaning that analysis results, ROIs, and any other information associated with the image will be lost.

## Image titles

Image titles may be considered as metadata from a user point of view, even if they are not stored in the metadata of the image (but in an attribute of the image object).

The “Edit” menu allows you to:

- “Add object title to plot”: this action will add a label on top of the image with its title, which might be useful in combination with the “Distribute on a grid” operation (see [Operations on Images](#)) to easily identify the images.
- “Copy titles to clipboard” : this action will copy the titles of the selected images to the clipboard, which might be useful to paste them in a text editor or in a spreadsheet.

Example of the content of the clipboard:

```

g001:
  s001: lorentz(a=1,sigma=1,mu=0,ymin=0)
  s002: derivative(s001)
  s003: wiener(s002)
g002: derivative(g001)
  s004: derivative(s001)
  s005: derivative(s002)
  s006: derivative(s003)
g003: fft(g002)
  s007: fft(s004)
  s008: fft(s005)
  s009: fft(s006)

```

## Regions Of Interest (ROI)

The Regions Of Interest (ROI) are image areas that are defined by the user to perform specific operations, processing, or analysis on them.

ROI are taken into account almost in all computing features in DataLab:

- The “Operations” menu features are done only on the ROI if one is defined (except if the operation changes the data shape - like the resize operation - or the pixel size - like the binning operation).
- The “Processing” menu actions are performed only on the ROI if one is defined (except if the destination signal data type is different from the source’s, like in the Fourier analysis features or like the thresholding operations).
- The “Analysis” menu actions are done only on the ROI if one is defined.

---

**Note:** ROI are stored as metadata, and thus attached to image.

---

The “Edit” menu allows you to:

- “Edit regions of interest” : open a dialog box to manage ROI associated with the selected image (add, remove, move, resize, etc.). The ROI definition dialog is exactly the same as ROI extraction (see below).
- “Remove regions of interest” : remove all defined ROI for the selected images.

## 2.4.3 Operations on Images

This section describes the operations that can be performed on images.

**See also:**

[Processing Images](#) for more information on image processing features, or [Analysis features on Images](#) for information on analysis features on images.

When the “Image Panel” is selected, the menus and toolbars are updated to provide image-related actions.

The “Operations” menu allows you to perform various operations on the current image or group of images. It also allows you to extract profiles, distribute images on a grid, or resize images.

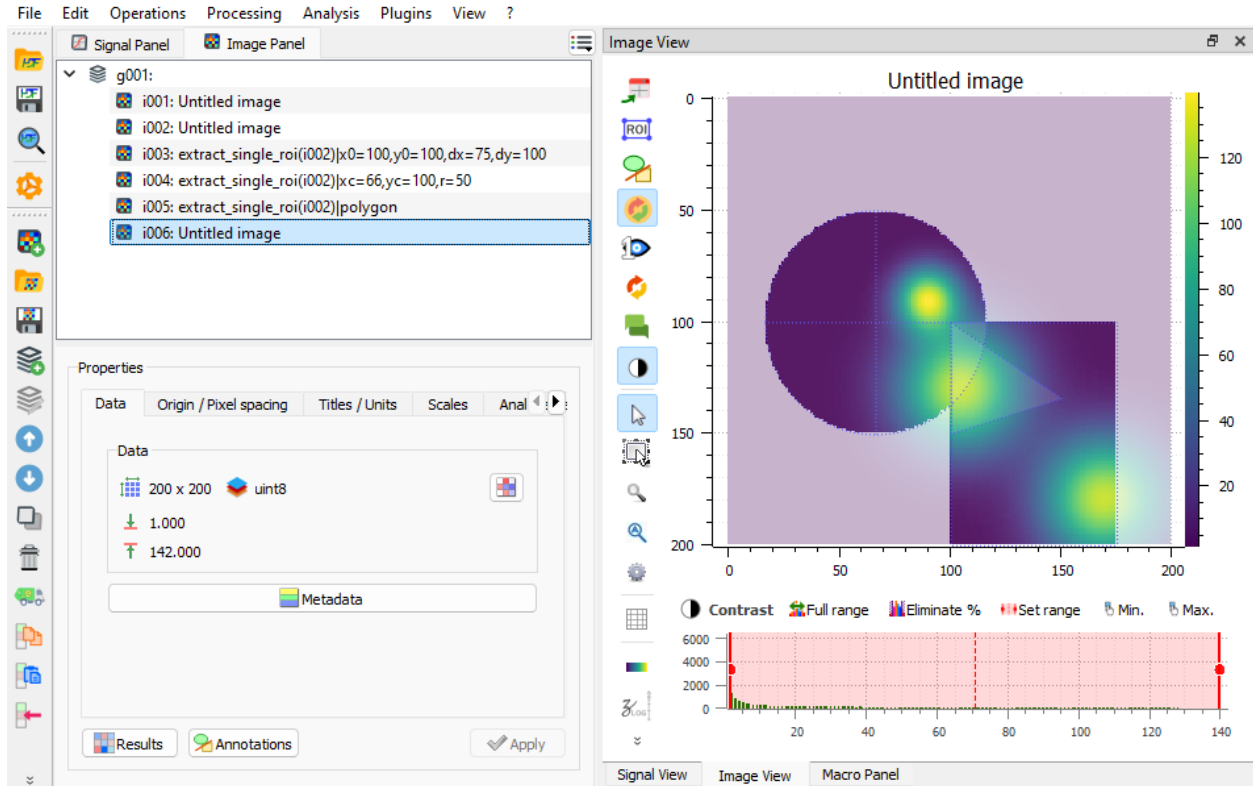


Fig. 35: An image with an ROI.

## Basic arithmetic operations

Operation	Description
Sum	$z_M = \sum_{k=0}^{M-1} z_k$
Average	$z_M = \frac{1}{M} \sum_{k=0}^{M-1} z_k$
Difference	$z_2 = z_1 - z_0$
Quadratic difference	$z_2 = \frac{z_1 - z_0}{\sqrt{2}}$
Product	$z_M = \prod_{k=0}^{M-1} z_k$
Division	$z_2 = \frac{z_1}{z_0}$
Inverse	$z_2 = \frac{1}{z_1}$

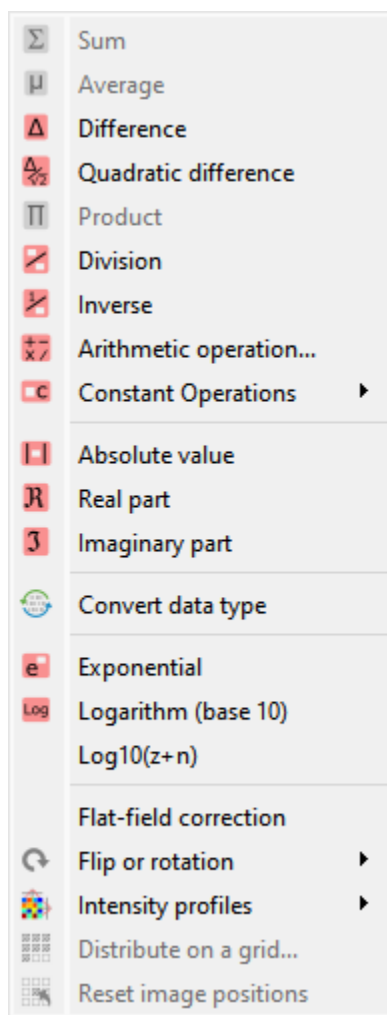


Fig. 36: Screenshot of the “Operations” menu.

## Operations with a constant

Create a new image which is the result of a constant operation on each selected image:

Operation	Equation
Addition	$z_k = z_{k-1} + \text{conv}(c)$
Subtraction	$z_k = z_{k-1} - \text{conv}(c)$
Multiplication	$z_k = \text{conv}(z_{k-1} \times c)$
Division	$z_k = \text{conv}\left(\frac{z_{k-1}}{c}\right)$

where  $c$  is the constant value and  $\text{conv}$  is the conversion function which handles data type conversion (keeping the same data type as the input image).

## Real and imaginary parts

Operation	Description
Absolute value	$z_k =  z_{k-1} $
Real part	$z_k = \Re(z_{k-1})$
Imaginary part	$z_k = \Im(z_{k-1})$

## Data type conversion

The “Convert data type” action allows you to convert the data type of the selected images. For integer data types, the conversion is done by clipping the values to the new data type range before effectively converting the data type. For floating point data types, the conversion is straightforward.

**Note:** Data type conversion uses the `cdl.algorithms.datatypes.clip_astype()` function which relies on `numpy.ndarray.astype()` function with the default parameters (`casting='unsafe'`).

## Basic mathematical functions

Function	Description
Exponential	$z_k = \exp(z_{k-1})$
Logarithm (base 10)	$z_k = \log_{10}(z_{k-1})$
Log10(z+n)	$z_k = \log_{10}(z_{k-1} + n)$ (avoid $\text{Log10}(0)$ on image background)

## Other operations

### Flat-field correction

Create a new image which is flat-field correction of the **two** selected images:

$$z_1 = \begin{cases} \frac{z_0}{z_f} \cdot \overline{z_f} & \text{if } z_0 > z_{threshold} \\ z_0 & \text{otherwise} \end{cases}$$

where  $z_0$  is the raw image,  $z_f$  is the flat field image,  $z_{threshold}$  is an adjustable threshold and  $\overline{z_f}$  is the flat field image average value:

$$\overline{z_f} = \frac{1}{N_{row} \cdot N_{col}} \cdot \sum_{i=0}^{N_{row}} \sum_{j=0}^{N_{col}} z_f(i, j)$$

---

**Note:** Raw image and flat field image are supposedly already corrected by performing a dark frame subtraction.

---

### Flip or rotation

Create a new image by flipping or rotating the data of the selected image. The image may be flipped horizontally, vertically, or diagonally (transposition). It may be rotated by 90°, 270° or any user-defined value.

## Intensity profiles

### Line profile

Extract an horizontal or vertical profile from each selected image, and create new signals from these profiles.

### Segment profile

Extract a segment profile from each selected image, and create new signals from these profiles.

### Average profile

Extract an horizontal or vertical profile averaged over a rectangular area, from each selected image, and create new signals from these profiles.

### Radial profile extraction

Extract a radial profile from each selected image, and create new signals from these profiles.

The following parameters are available:

Parameter	Description
Center	Center around which the radial profile is computed: centroid, image center, or user-defined
X	X coordinate of the center (if user-defined), in pixels
Y	Y coordinate of the center (if user-defined), in pixels

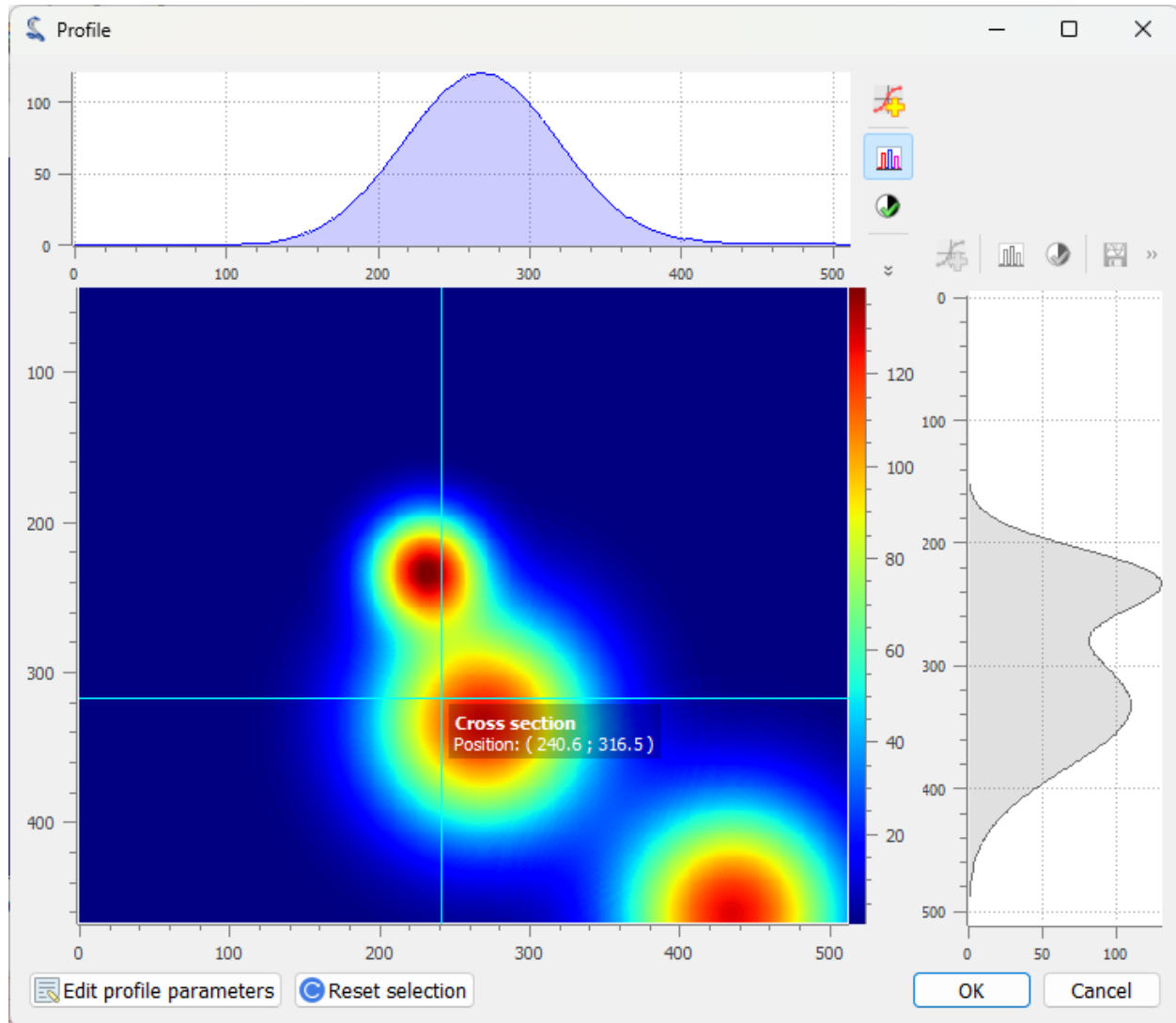


Fig. 37: Line profile dialog. Parameters may also be set manually (“Edit profile parameters” button).



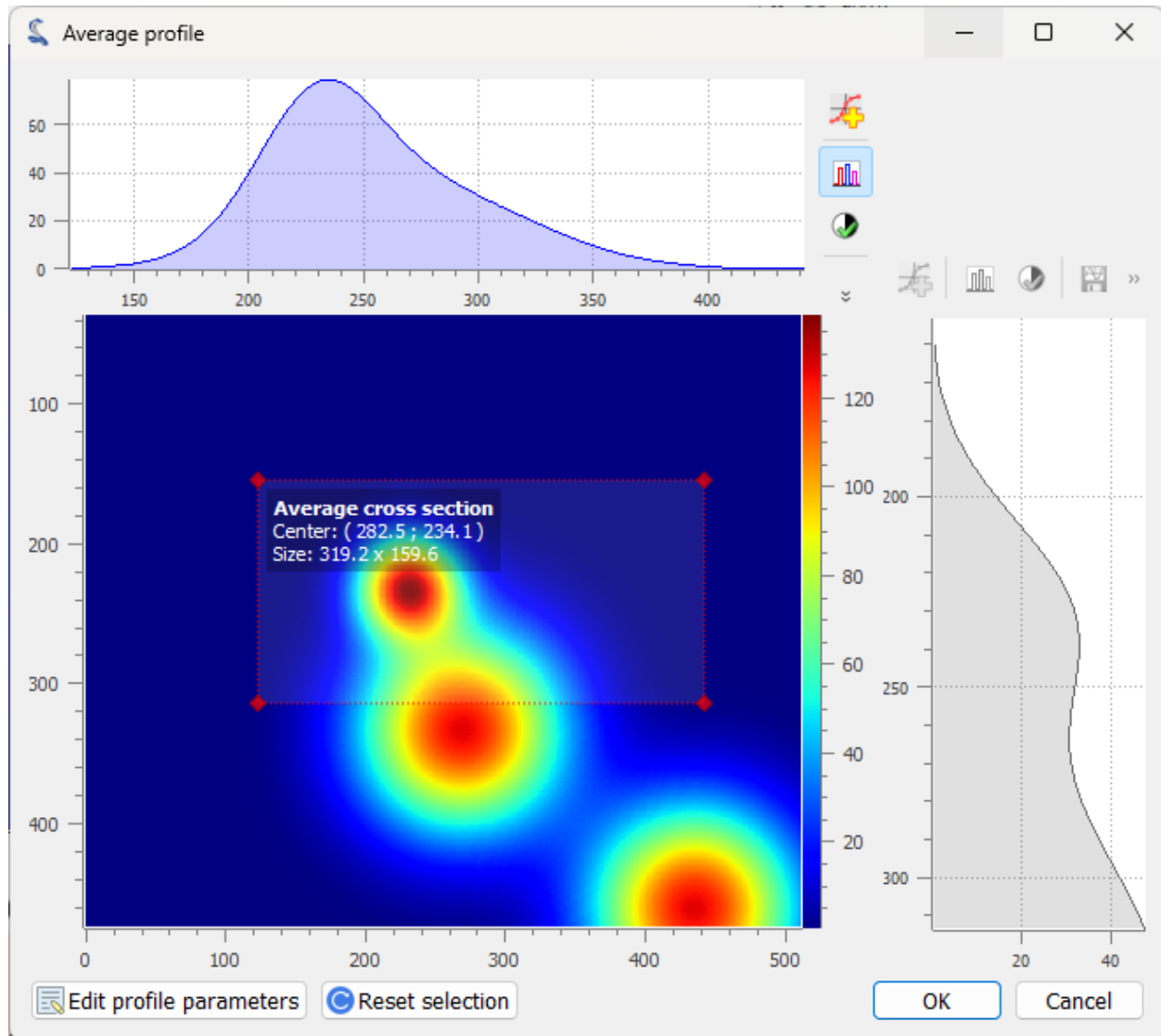


Fig. 38: Average profile dialog: the area is defined by a rectangle shape. Parameters may also be set manually ("Edit profile parameters" button).

## Distribute images along a grid

Feature	Description
Distribute on a grid	Distribute selected images on a regular grid
Reset image positions	Reset the positions of the selected images to first image (x0, y0) coordinates

### 2.4.4 Processing Images

This section describes the image processing features available in DataLab.

**See also:**

*Operations on Images* for more information on operations that can be performed on images, or *Analysis features on Images* for information on analysis features on images.

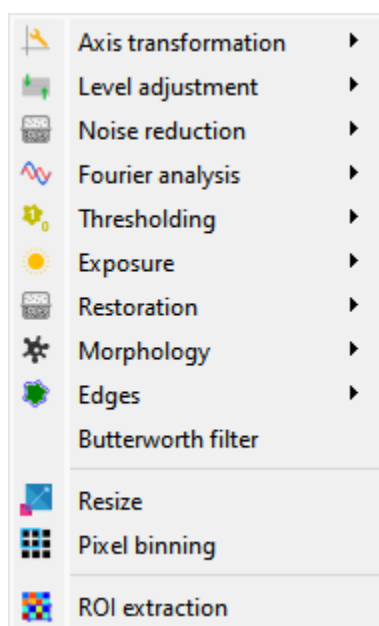


Fig. 39: Screenshot of the “Processing” menu.

When the “Image Panel” is selected, the menus and toolbars are updated to provide image-related actions.

The “Processing” menu allows you to perform various processing on the current image or group of images: it allows you to apply filters, to perform exposure correction, to perform denoising, to perform morphological operations, and so on.

## Axis transformation

### Linear calibration

Create a new image which is a linear calibration of each selected image with respect to Z axis:

Parameter	Linear calibration
Z-axis	$z_1 = a.z_0 + b$

### Swap X/Y axes

Create a new image which is the result of swapping X/Y data.

### Level adjustment

#### Normalize

Create a new image which is the normalized version of each selected image by maximum, amplitude, sum, energy or RMS:

Normalization	Equation
Maximum	$z_1 = \frac{z_0}{z_{max}}$
Amplitude	$z_1 = \frac{z_0}{z_{max} - z_{min}}$
Area	$z_1 = \frac{z_0}{\sum_{i=0}^{N-1} z_i}$
Energy	$z_1 = \frac{z_0}{\sqrt{\sum_{n=0}^N  z_0[n] ^2}}$
RMS	$z_1 = \frac{z_0}{\sqrt{\frac{1}{N} \sum_{n=0}^N  z_0[n] ^2}}$

### Clipping

Apply the clipping to each selected image.

### Offset correction

Create a new image which is the result of offset correction on each selected image. This operation is performed by subtracting the image background value which is estimated by the mean value of a user-defined rectangular area.

## Noise reduction

Create a new image which is the result of noise reduction on each selected image.

The following filters are available:

Filter	Formula/implementation
Gaussian filter	<code>scipy.ndimage.gaussian_filter</code>
Moving average	<code>scipy.ndimage.uniform_filter</code>
Moving median	<code>scipy.ndimage.median_filter</code>
Wiener filter	<code>scipy.signal.wiener</code>

## Fourier analysis

### Zero padding

Create a new image which is the result of zero padding on each selected image.

The following parameters are available:

Parameter	Description
Strategy	Zero padding strategy (see below)
Rows	Number of rows to add (if <i>strategy</i> is 'custom')
Columns	Number of columns to add (if <i>strategy</i> is 'custom')
Position	Position of the added zeros: 'bottom-right', 'centered'

Zero padding strategy refers to the method used to add zeros to the image, and it can be one of the following:

Strategy	Description
next_pow2	Next power of 2 (e.g. 512, 1024, ...)
multiple_of_64	Next multiple of 64 (e.g. 512, 576, ...)
custom	Custom size (user-defined)

## FFT related functions

Create a new image which is the result of a Fourier analysis on each selected image.

The following functions are available:

Function	Description	Formula/implementation
FFT	Fast Fourier Transform	<code>numpy.fft.fft2</code>
Inverse FFT	Inverse Fast Fourier Transform	<code>numpy.fft.ifft2</code>
Magnitude spectrum	Optionnal: use logarithmic scale (dB)	$z_1 =  FFT(z_0) $ or $z_1 = 20 \log_{10}( FFT(z_0) )$ (dB)
Phase spectrum		$z_1 = \angle(FFT(z_0))$
Power spectral density	Optionnal: use logarithmic scale (dB)	$z_1 =  FFT(z_0) ^2$ or $z_1 = 10 \log_{10}( FFT(z_0) ^2)$ (dB)

---

**Note:** FFT and inverse FFT are performed using frequency shifting if the option is enabled in DataLab settings (see *Settings*).

---

## Thresholding

Create a new image which is the result of thresholding on each selected image, eventually based on user-defined parameters (“Parametric thresholding”).

The following parameters are available when selecting “Parametric thresholding”:

Parameter	Description
Threshold method	The thresholding method to use (see table below)
Bins	Number of bins for histogram calculation
Value	Threshold value
Operation	Operation to apply (> or <)

The following thresholding methods are available:

Method	Implementation
Manual	Manual thresholding (user-defined parameters)
ISODATA	<code>skimage.filters.threshold_isodata</code>
Li	<code>skimage.filters.threshold_li</code>
Mean	<code>skimage.filters.threshold_mean</code>
Minimum	<code>skimage.filters.threshold_minimum</code>
Otsu	<code>skimage.filters.threshold_otsu</code>
Triangle	<code>skimage.filters.threshold_triangle</code>
Yen	<code>skimage.filters.threshold_yen</code>

---

**Note:** The “All thresholding methods” option allows to perform all thresholding methods on the same image. Combined with the “distribute on a grid” option, this allows to compare the different thresholding methods on the same image.

---

## Exposure

Create a new image which is the result of exposure correction on each selected image.

The following functions are available:

Function	Implementation	Comments
Gamma correction	<code>skim-age.exposure.adjust_gamma</code>	
Logarithmic correction	<code>skim-age.exposure.adjust_log</code>	
Sigmoid correction	<code>skim-age.exposure.adjust_sigmoi</code>	
Histogram equalization	<code>skim-age.exposure.equalize_hist</code>	
Adaptive histogram equalization	<code>skim-age.exposure.equalize_adap</code>	Contrast Limited Adaptive Histogram Equalization (CLAHE) algorithm
Intensity rescaling	<code>skim-age.exposure.rescale_intens</code>	Stretch or shrink image intensity levels

## Restoration

Create a new image which is the result of restoration on each selected image.

The following functions are available:

Function	Implementation	Comments
Total variation denoising	<code>skim-age.restoration.denoise_tv_</code>	
Bilateral filter denoising	<code>skim-age.restoration.denoise_bila</code>	
Wavelet denoising	<code>skim-age.restoration.denoise_wav</code>	
White Top-Hat denoising	<code>skim-age.morphology.white_toph</code>	Denoise image by subtracting its white top hat transform

**Note:** The “All denoising methods” option allows to perform all denoising methods on the same image. Combined with the “distribute on a grid” option, this allows to compare the different denoising methods on the same image.

## Morphology

Create a new image which is the result of morphological operations on each selected image, using a disk footprint.

The following functions are available:

Function	Implementation
White Top-Hat (disk)	<code>skimage.morphology.white_tophat</code>
Black Top-Hat (disk)	<code>skimage.morphology.black_tophat</code>
Erosion (disk)	<code>skimage.morphology.erosion</code>
Dilation (disk)	<code>skimage.morphology.dilation</code>
Opening (disk)	<code>skimage.morphology.opening</code>
Closing (disk)	<code>skimage.morphology.closing</code>

---

**Note:** The “All morphological operations” option allows to perform all morphological operations on the same image. Combined with the “distribute on a grid” option, this allows to compare the different morphological operations on the same image.

---

## Edges

Create a new image which is the result of edge filtering on each selected image.

The following functions are available:

Function	Implementation
Roberts filter	<code>skimage.filters.roberts</code>
Prewitt filter	<code>skimage.filters.prewitt</code>
Prewitt filter (horizontal)	<code>skimage.filters.prewitt_h</code>
Prewitt filter (vertical)	<code>skimage.filters.prewitt_v</code>
Sobel filter	<code>skimage.filters.sobel</code>
Sobel filter (horizontal)	<code>skimage.filters.sobel_h</code>
Sobel filter (vertical)	<code>skimage.filters.sobel_v</code>
Scharr filter	<code>skimage.filters.scharr</code>
Scharr filter (horizontal)	<code>skimage.filters.scharr_h</code>
Scharr filter (vertical)	<code>skimage.filters.scharr_v</code>
Farid filter	<code>skimage.filters.farid</code>
Farid filter (horizontal)	<code>skimage.filters.farid_h</code>
Farid filter (vertical)	<code>skimage.filters.farid_v</code>
Laplace filter	<code>skimage.filters.laplace</code>
Canny filter	<code>skimage.feature.canny</code>

---

**Note:** The “All edges filters” option allows to perform all edge filtering algorithms on the same image. Combined with the “distribute on a grid” option, this allows to compare the different edge filters on the same image.

---

## Butterworth filter

Perform Butterworth filter on an image (implementation based on `skimage.filters.butterworth`)

## Resize

Create a new image which is a resized version of each selected image.

## Pixel binning

Combine clusters of adjacent pixels, throughout the image, into single pixels. The result can be the sum, average, median, minimum, or maximum value of the cluster.

## ROI extraction

Create a new image from a user-defined Region of Interest.

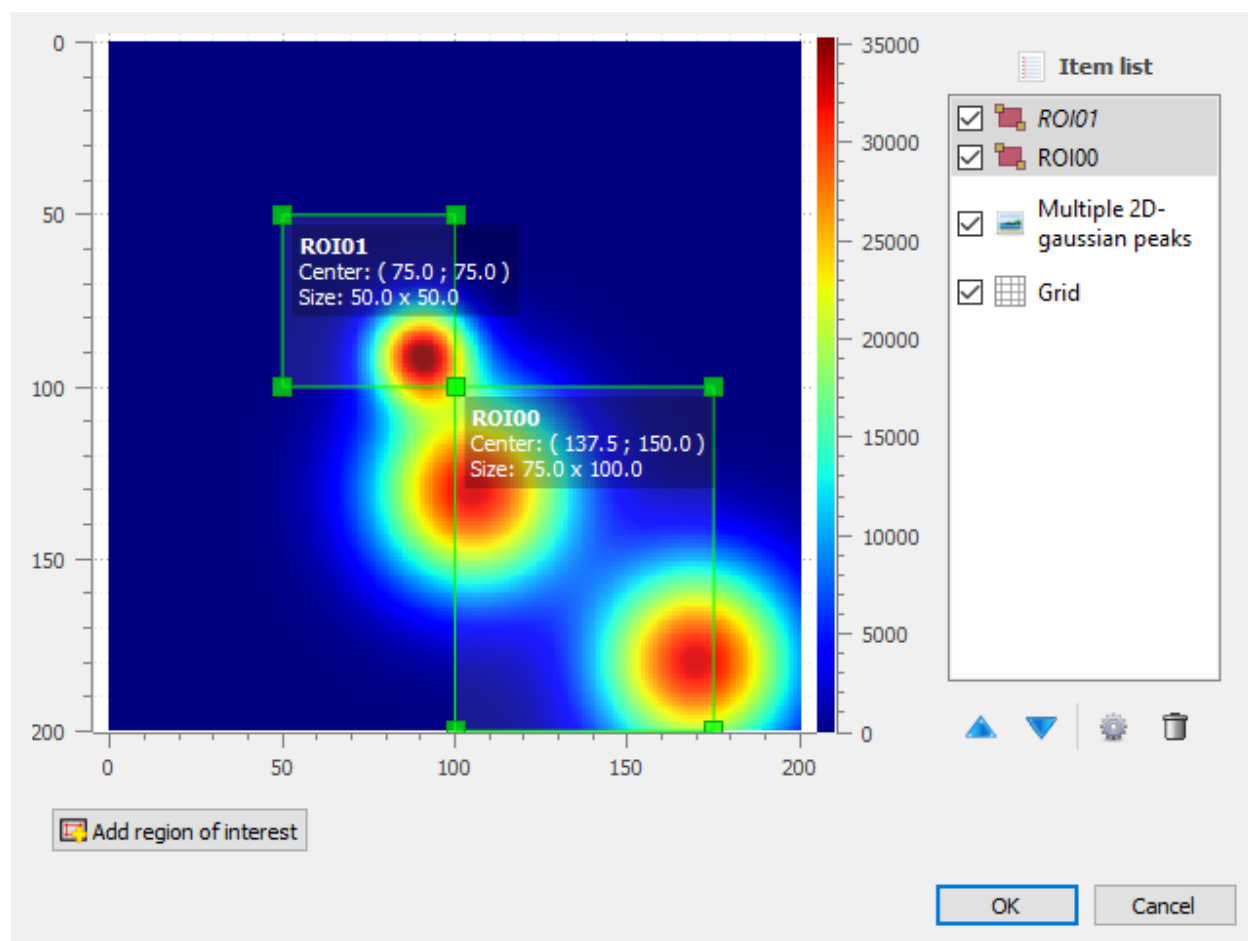


Fig. 40: ROI extraction dialog: the ROI is defined by moving the position and adjusting the size of a rectangle shape.

## 2.4.5 Analysis features on Images

This section describes the image analysis features available in DataLab.

### See also:

[Operations on Images](#) for more information on operations that can be performed on images, or [Processing Images](#) for information on processing features on images.

When the “Image Panel” is selected, the menus and toolbars are updated to provide image-related actions.

The “Analysis” menu allows you to perform various computations on the current image or group of images. It also allows you to compute statistics, to compute the centroid, to detect peaks, to detect contours, and so on.



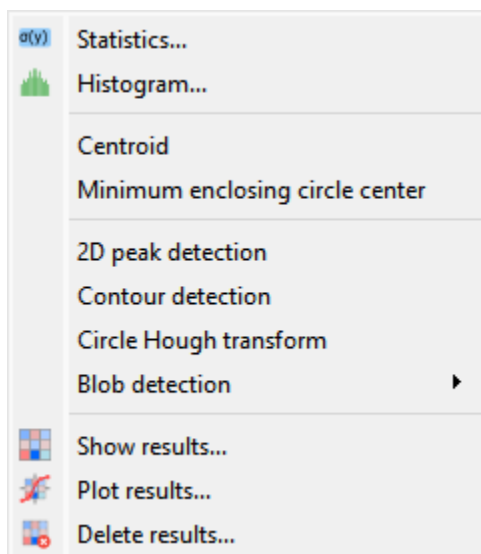


Fig. 41: Screenshot of the “Analysis” menu.

**Note:** In DataLab vocabulary, an “analysis” is a feature that computes a scalar result from an image. This result is stored as metadata, and thus attached to image. This is different from a “processing” which creates a new image from an existing one.

## Statistics

Compute statistics on selected image and show a summary table.

	min(z)	max(z)	<z>	$\sigma(z)$	$\Sigma(z)$	SNR(z)
i000	0	32754	512.558	2852.36	1.28139e+08	5.56494
i000 ROI00	0	32754	512.558	2852.36	6.40697e+07	5.56494
i000 ROI01	0	0	0	0	0	nan
i000 ROI02	0	32754	512.558	2852.36	3.20349e+07	5.56494

Format
Resize
☒ Background color

Close

Fig. 42: Example of statistical summary table: each row is associated to an ROI (the first row gives the statistics for the whole data).

## Histogram

Compute histogram of selected image and show it in the Signal Panel.

Parameters are:

Parameter	Description
Bins	Number of bins
Lower limit	Lower limit of the histogram
Upper limit	Upper limit of the histogram

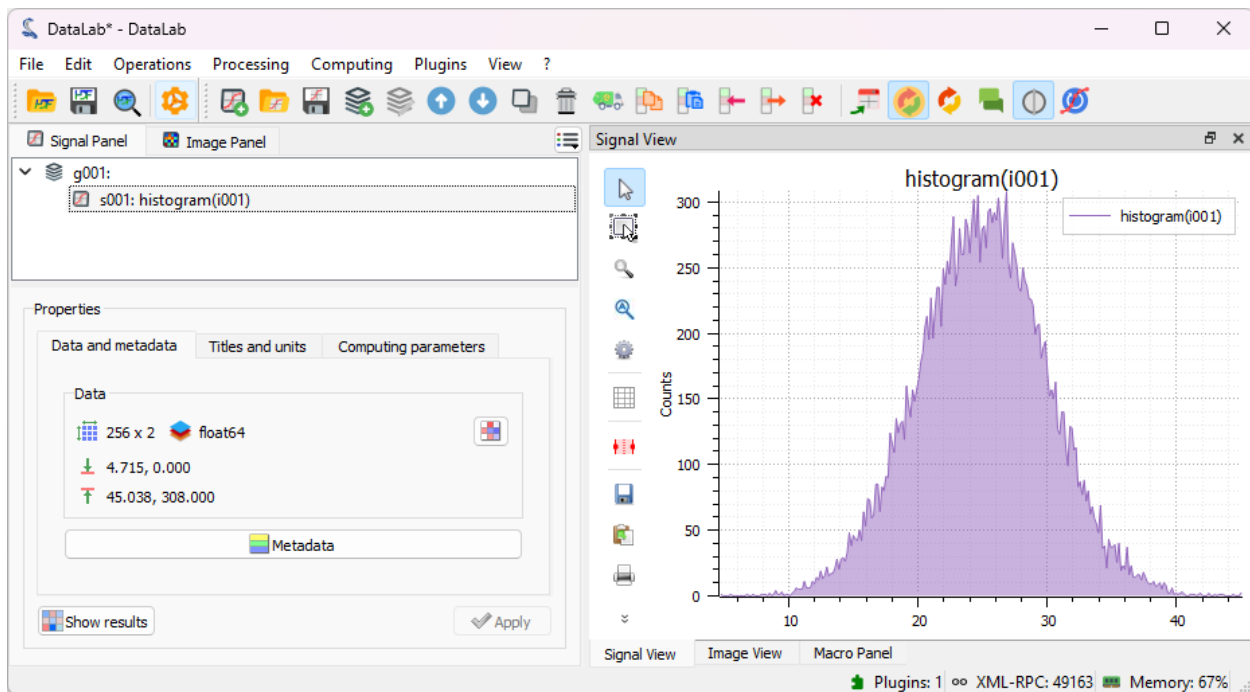


Fig. 43: Example of histogram.

## Centroid

Compute image centroid using a Fourier transform method (as discussed by [Weisshaar et al.](#)). This method is quite insensitive to background noise.

## Minimum enclosing circle center

Compute the circle contour enclosing image values above a threshold level defined as the half-maximum value.

## 2D peak detection

Automatically find peaks on image using a minimum-maximum filter algorithm.

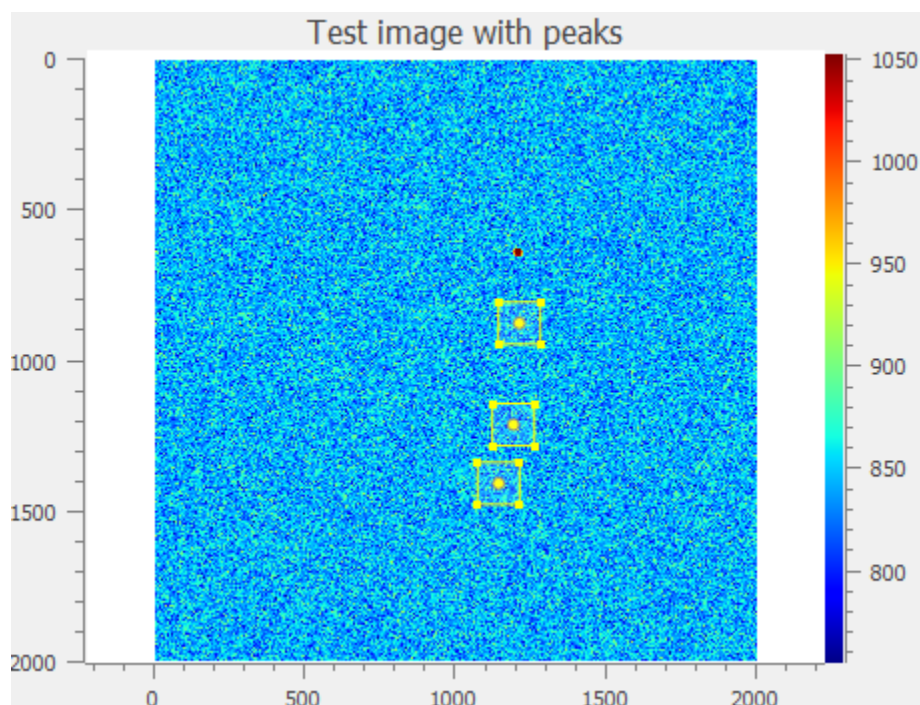


Fig. 44: Example of 2D peak detection.

### See also:

See [2D Peak Detection](#) for more details on algorithm and associated parameters.

## Contour detection

Automatically extract contours and fit them using a circle or an ellipse, or directly represent them as a polygon.

### See also:

See [Contour Detection](#) for more details on algorithm and associated parameters.

---

**Note:** Computed scalar results are systematically stored as metadata. Metadata is attached to image and serialized with it when exporting current session in a HDF5 file.

---

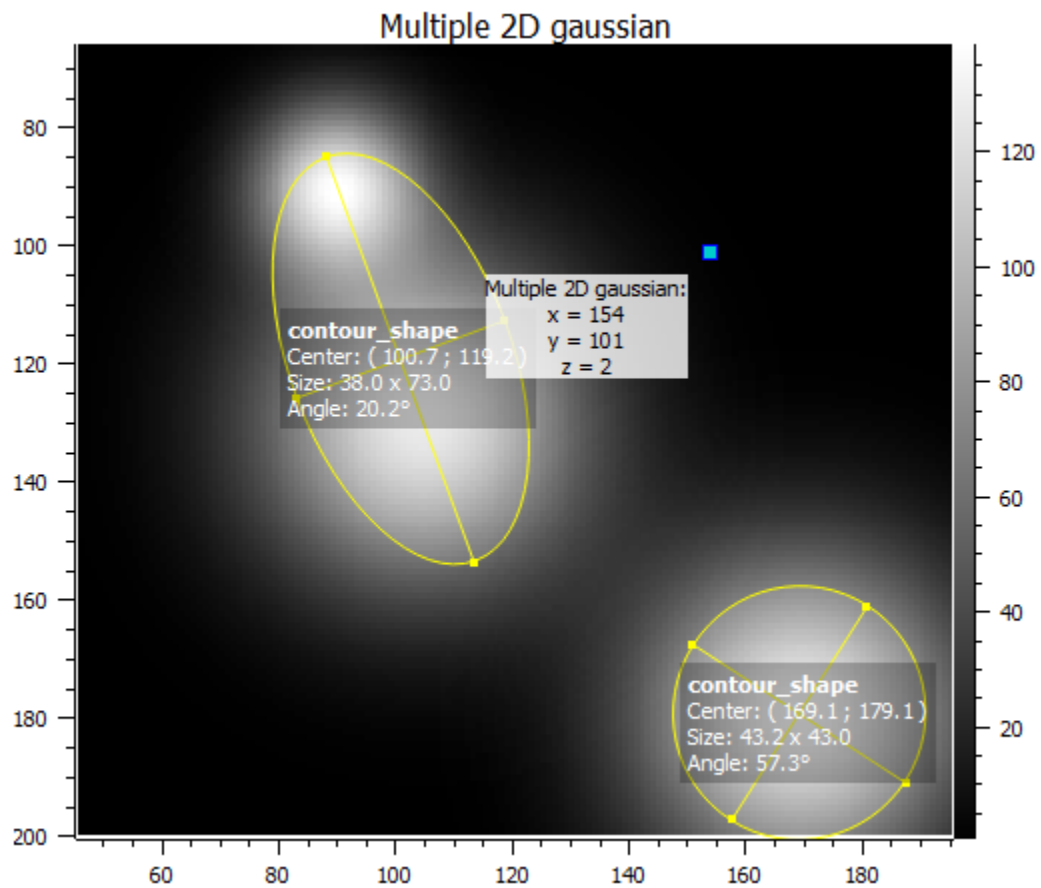


Fig. 45: Example of contour detection.

## Circle Hough transform

Detect circular shapes using circle Hough transform (implementation based on `skimage.transform.hough_circle_peaks`).

## Blob detection

### Blob detection (DOG)

Detect blobs using Difference of Gaussian (DOG) method (implementation based on `skimage.feature.blob_dog`).

### Blob detection (DOH)

Detect blobs using Determinant of Hessian (DOH) method (implementation based on `skimage.feature.blob_doh`).

### Blob detection (LOG)

Detect blobs using Laplacian of Gaussian (LOG) method (implementation based on `skimage.feature.blob_log`).

### Blob detection (OpenCV)

Detect blobs using OpenCV implementation of `SimpleBlobDetector`.

## Show results

Show the results of all analyses performed on the selected images. This shows the same table as the one shown after having performed a computation.

## Plot results

Plot the results of analyses performed on the selected images, with user-defined X and Y axes (e.g. plot the contour circle radius as a function of the image number).

## 2.4.6 View options for Images

When the “Image Panel” is selected, the menus and toolbars are updated to provide image-related actions.

The “View” menu allows you to visualize the current image or group of images. It also allows you to show/hide titles, to show/hide the contrast panel, to refresh the visualization, and so on.

### View in a new window

Open a new window to visualize the selected images.

This option allows you to show the selected images in a separate window, in which you can visualize the data more comfortably (e.g., by maximizing the window) as well as to add or edit annotations.

When you click on the button “Annotations” in the toolbar of the new window, the annotation editing mode is activated (see the section “Edit annotations” below).

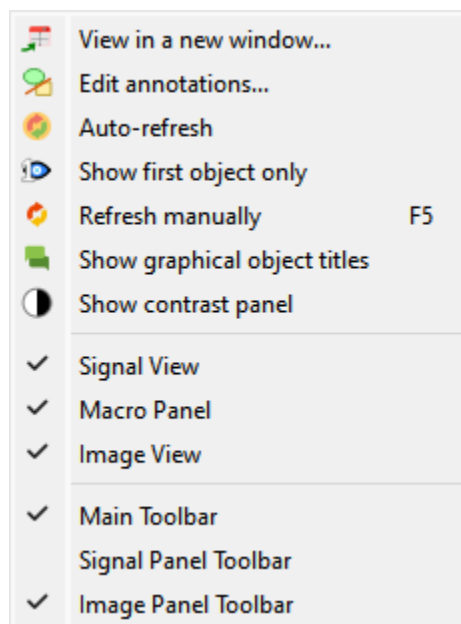


Fig. 46: Screenshot of the “View” menu.

## Edit annotations

Open a new window to edit annotations.

This option allows you to edit the annotations of the selected images in a separate window. This is equivalent to select the “View in a new window” option and then click on the “Annotations” button in the toolbar of the new window.

Annotations are used to add text, lines, rectangles, ellipses, and other geometrical shapes to the images. They are useful to highlight regions of interest, to add comments, to mark points, and so on.

---

**Note:** The annotations are saved in the metadata of the image, so they are persistent and will be displayed every time you visualize the image.

---

The typical workflow to edit annotations is as follows:

1. Select the “Edit annotations” option.
2. In the new window, add annotations by clicking on the corresponding buttons at the bottom of the window.
3. Eventually, customize the annotations by changing their properties (e.g., the text, the color, the position, etc.) using the “Parameters” option in the context menu of the annotations.
4. When you are done, click on the “OK” button to save the annotations. This will close the window and the annotations will be saved in the metadata of the image and will be displayed in the main window.

---

**Note:** Annotations may be copied from an image to another by using the “copy/paste metadata” features.

---

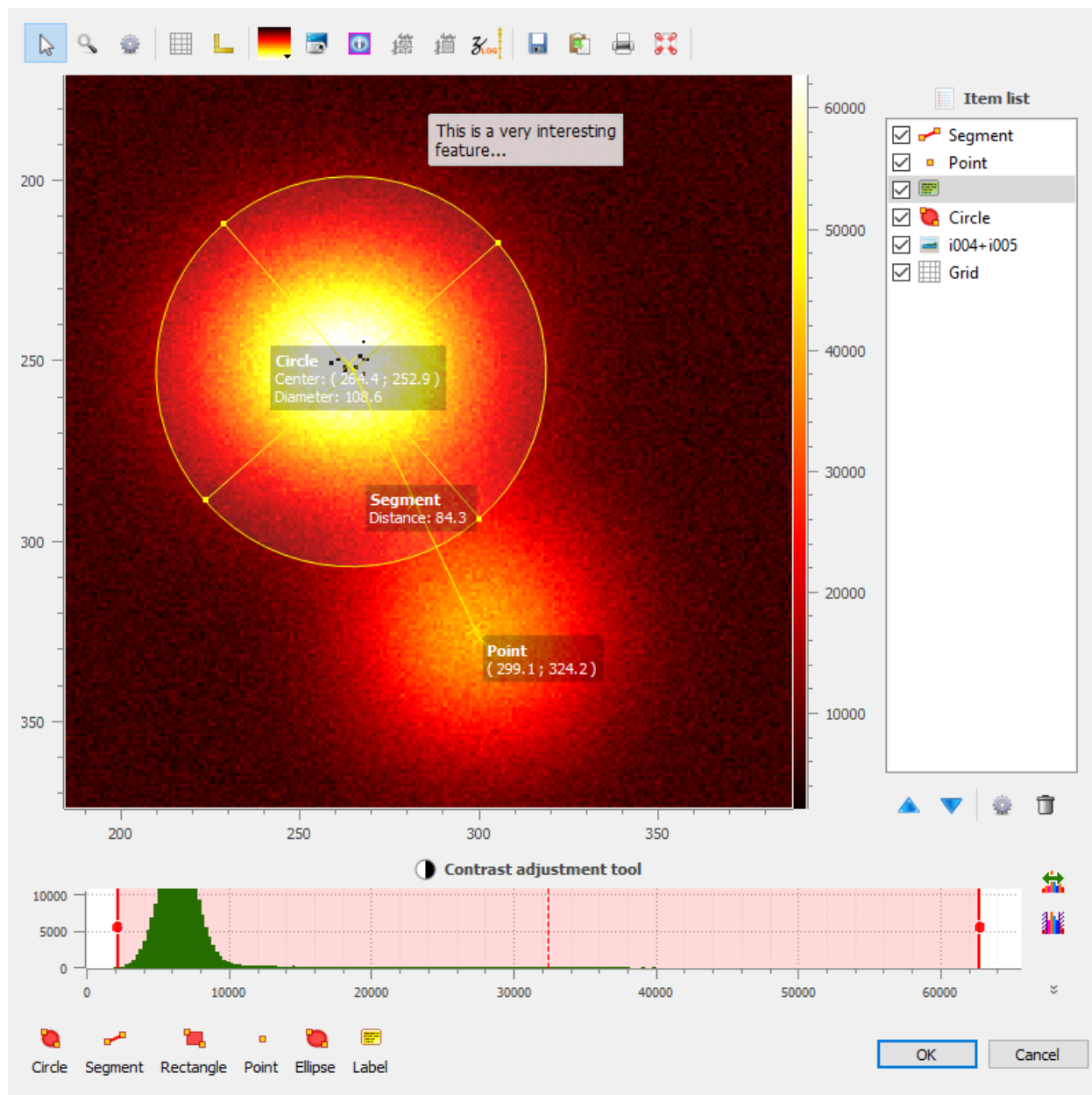


Fig. 47: Annotations may be added in the separate view.

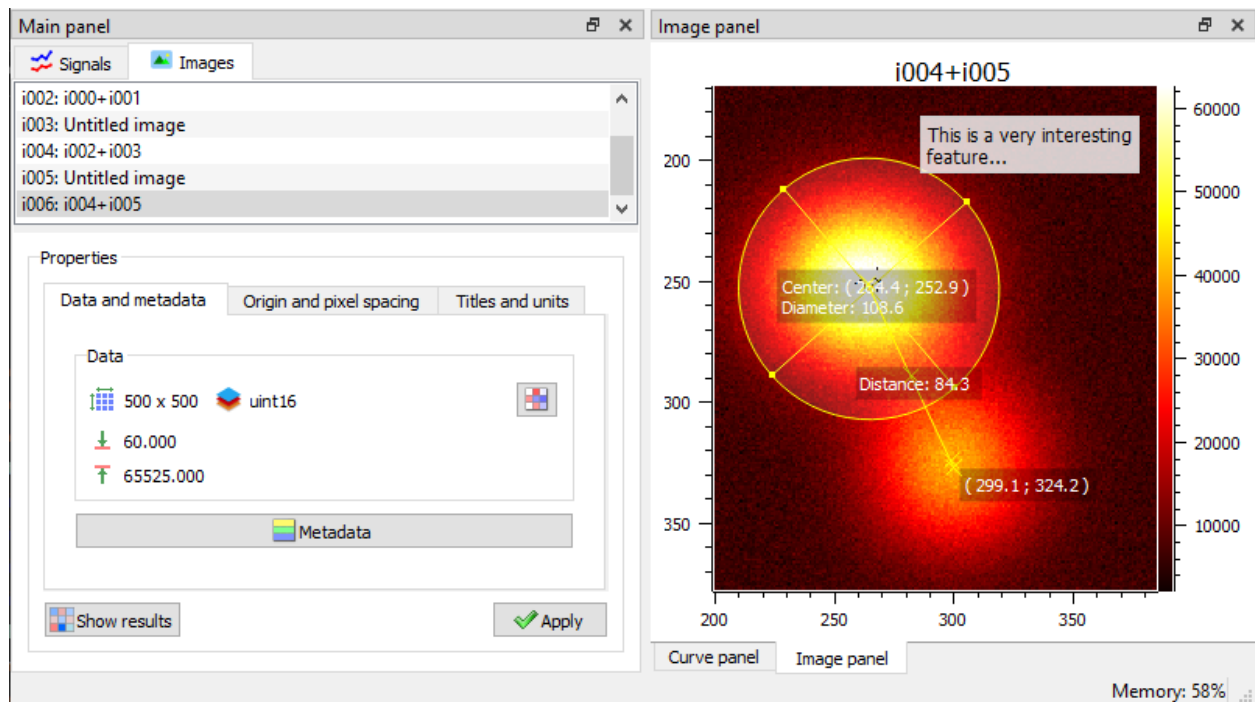


Fig. 48: Annotations are now part of the image metadata.

## Auto-refresh

Automatically refresh the visualization when the data changes:

- When enabled (default), the plot view is automatically refreshed when the data changes.
- When disabled, the plot view is not refreshed until you manually refresh it by using the “Refresh manually” menu entry.

Even though the refresh algorithm is optimized, it may still take some time to refresh the plot view when the data changes, especially when the data set is large. Therefore, you may want to disable the auto-refresh feature when you are working with large data sets, and enable it again when you are done. This will avoid unnecessary refreshes.

## Refresh manually

Refresh the visualization manually.

This triggers a refresh of the plot view. It is useful when the auto-refresh feature is disabled, or when you want to force a refresh of the plot view.



### Show contrast panel

Show/hide contrast adjustment panel.

### Show graphical object titles

Show/hide titles of analysis results or annotations.

This option allows you to show or hide the titles of the graphical objects (e.g., the titles of the analysis results or annotations). Hiding the titles can be useful when you want to visualize the data without any distractions, or if there are too many titles and they are overlapping.

## 2.4.7 2D Peak Detection

DataLab provides a “2D Peak Detection” feature which is based on a minimum-maximum filter algorithm.

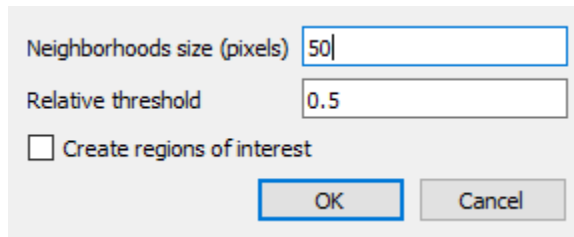


Fig. 49: 2D peak detection parameters.

#### How to use the feature:

- Create or open an image in DataLab workspace
- Select “2d peak detection” in “Analysis” menu
- Enter parameters “Neighborhoods size” and “Relative threshold”
- Check “Create regions of interest” if you want a ROI defined for each detected peak (this may become useful when using another computation afterwards on each area around peaks, e.g. contour detection)

#### Results are shown in a table:

- Each row is associated to a detected peak
- First column shows the ROI index (0 if no ROI is defined on input image)
- Second and third columns show peak coordinates

#### The 2d peak detection algorithm works in the following way:

- First, the minimum and maximum filtered images are computed using a sliding window algorithm with a user-defined size (implementation based on `scipy.ndimage.minimum_filter` and `scipy.ndimage.maximum_filter`)
- Then, the difference between the maximum and minimum filtered images is clipped at a user-defined threshold
- Resulting image features are labeled using `scipy.ndimage.label`
- Peak coordinates are then obtained from labels center
- Duplicates are eventually removed

Results - NumPy array (read only)			
	ROI	x	y
Peaks(i000)	0	1366	638
Peaks(i000)	0	1416	1069
Peaks(i000)	0	1018	1135
Peaks(i000)	0	828	1229

Format    Resize    ☒ Background color

Close

Fig. 50: 2d peak detection results (see test “peak2d\_app.py”)

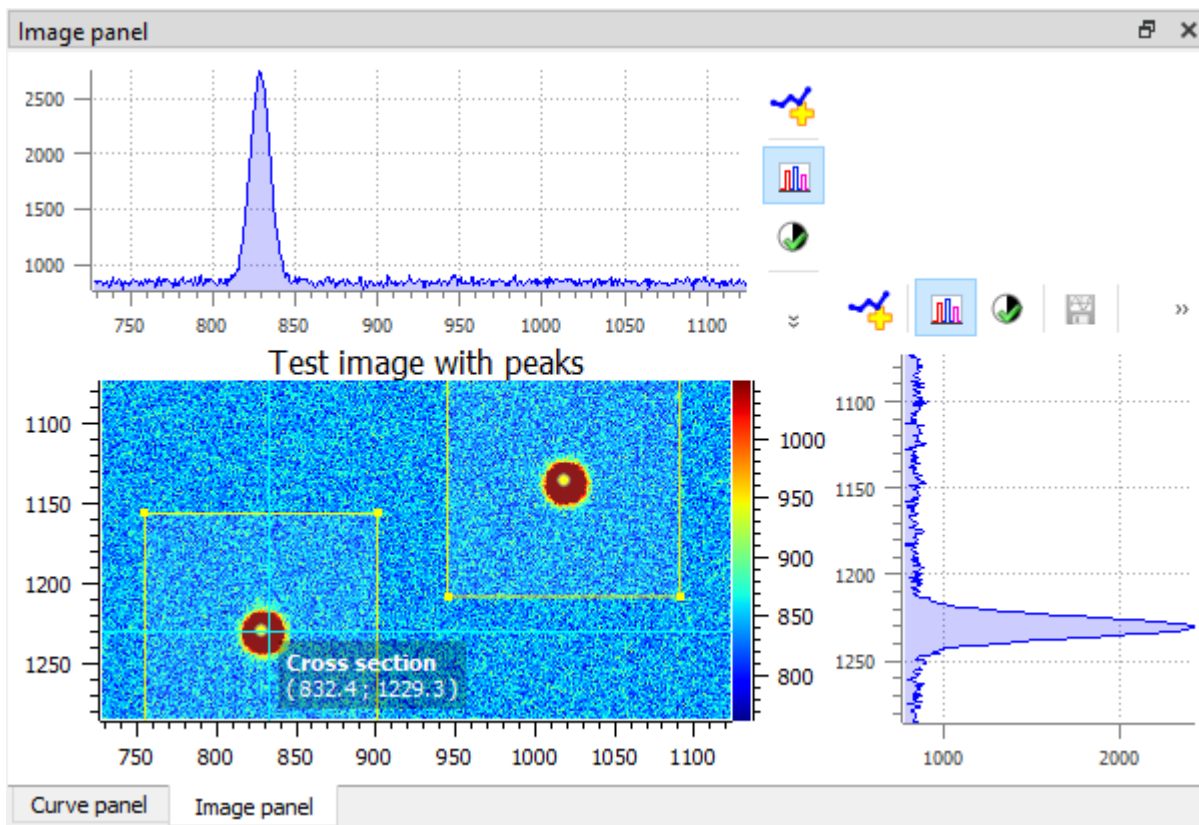


Fig. 51: Example of 2D peak detection.

The 2d peak detection parameters are the following:

- “Neighborhoods size”: size of the sliding window (see above)
- “Relative threshold”: detection threshold

Feature is based on `get_2d_peaks_coords` function from `cdl.algorithms` module:

```
def get_2d_peaks_coords(
    data: np.ndarray, size: int | None = None, level: float = 0.5
) -> np.ndarray:
    """Detect peaks in image data, return coordinates.

    If neighborhoods size is None, default value is the highest value
    between 50 pixels and the 1/40th of the smallest image dimension.

    Detection threshold level is relative to difference
    between data maximum and minimum values.

    Args:
        data: Input data
        size: Neighborhood size (default: None)
        level: Relative level (default: 0.5)

    Returns:
        Coordinates of peaks
    """
    if size is None:
        size = max(min(data.shape) // 40, 50)
    data_max = spi.maximum_filter(data, size)
    data_min = spi.minimum_filter(data, size)
    data_diff = data_max - data_min
    diff = (data_max - data_min) > get_absolute_level(data_diff, level)
    maxima = data == data_max
    maxima[diff == 0] = 0
    labeled, _num_objects = spi.label(maxima)
    slices = spi.find_objects(labeled)
    coords = []
    for dy, dx in slices:
        x_center = int(0.5 * (dx.start + dx.stop - 1))
        y_center = int(0.5 * (dy.start + dy.stop - 1))
        coords.append((x_center, y_center))
    if len(coords) > 1:
        # Eventually removing duplicates
        dist = distance_matrix(coords)
        for index in reversed(np.unique(np.where((dist < size) & (dist > 0))[1])):
            coords.pop(index)
    return np.array(coords)
```

## 2.4.8 Contour Detection

DataLab provides a “Contour Detection” feature which is based on the [marching cubes algorithm](#).

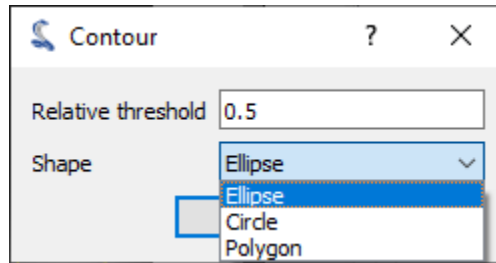


Fig. 52: Contour detection parameters.

### How to use the feature:

- Create or open an image in DataLab workspace
- Eventually create a ROI around the target area
- Select “Contour detection” in “Analysis” menu
- Enter parameter “Shape” (“Ellipse”, “Circle” or “Polygon”)

	ROI	x0	y0	x1	y1	x2
i001: contour_shape	0	110	150.125	109	150.375	108
i001: contour_shape	0	160.75	199	160	198.625	159

Format    Resize    ☒ Background color    Close

Fig. 53: Contour detection results (see test “contour\_app.py”)

### Results are shown in a table:

- Each row is associated to a contour
- First column shows the ROI index (0 if no ROI is defined on input image)
- Other columns show contour coordinates: 4 columns for circles (coordinates of diameter), 8 columns for ellipses (coordinates of diameters)

### The contour detection algorithm works in the following way:

- First, iso-valued contours are computed (implementation based on [skimage.measure.find\\_contours.find\\_contours](#))

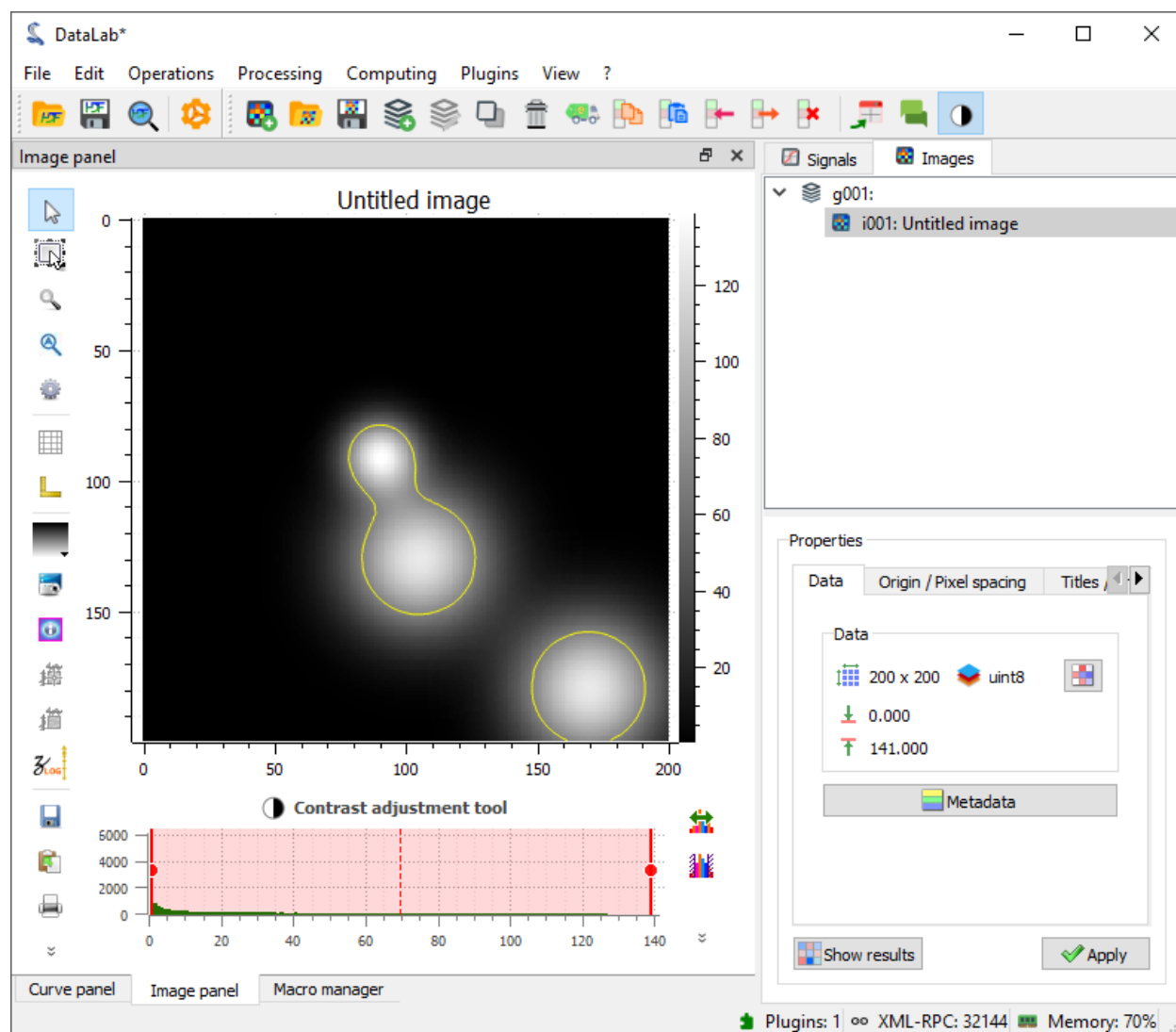


Fig. 54: Example of contour detection.

- Then, each contour is fitted to the closest ellipse (or circle)

Feature is based on `get_contour_shapes` function from `cdl.algorithms` module:

```
def get_contour_shapes(
    data: np.ndarray | ma.MaskedArray,
    shape: Literal["circle", "ellipse", "polygon"] = "ellipse",
    level: float = 0.5,
) -> np.ndarray:
    """Find iso-valued contours in a 2D array, above relative level (.5 means
    FWHM),
    then fit contours with shape ('ellipse' or 'circle')

    Args:
        data: Input data
        shape: Shape to fit. Default is 'ellipse'
        level: Relative level (default: 0.5)

    Returns:
        Coordinates of shapes
        """
    # pylint: disable=too-many-locals
    assert shape in ("circle", "ellipse", "polygon")
    contours = measure.find_contours(data, level=get_absolute_level(data,
    level))
    coords = []
    for contour in contours:
        # `contour` is a (N, 2) array (rows, cols): we need to check if all
        those
        # coordinates are masked: if so, we skip this contour
        if isinstance(data, ma.MaskedArray) and np.all(
            data.mask[contour[:, 0].astype(int), contour[:, 1].astype(int)]
        ):
            continue
        if shape == "circle":
            model = measure.CircleModel()
            if model.estimate(contour):
                yc, xc, r = model.params
                if r <= 1.0:
                    continue
                coords.append([xc, yc, r])
        elif shape == "ellipse":
            model = measure.EllipseModel()
            if model.estimate(contour):
                yc, xc, b, a, theta = model.params
                if a <= 1.0 or b <= 1.0:
                    continue
                coords.append([xc, yc, a, b, theta])
        elif shape == "polygon":
            # `contour` is a (N, 2) array (rows, cols): we need to convert it
            # to a list of x, y coordinates flattened in a single list
            coords.append(contour[:, :-1].flatten())
        else:
            raise NotImplementedError(f"Invalid contour model {model}")
```

(continues on next page)

(continued from previous page)

```
if shape == "polygon":
    # `coords` is a list of arrays of shape (N, 2) where N is the number of
    ↪points
    # that can vary from one array to another, so we need to padd with
    ↪NaNs each
    # array to get a regular array:
    max_len = max(coord.shape[0] for coord in coords)
    arr = np.full((len(coords), max_len), np.nan)
    for i_row, coord in enumerate(coords):
        arr[i_row, : coord.shape[0]] = coord
    return arr
return np.array(coords)
```





The public Application Programming Interface (API) of DataLab offers a set of functions to access the DataLab features. Those functions are available in various submodules of the *cdl* package. The following table lists the available submodules and their purpose:

Submodule	Purpose
<i>cdl.algorithms</i>	Algorithms for data analysis, which operates on NumPy arrays
<i>cdl.param</i>	Convenience module to access the DataLab sets of parameters (instances of <code>guidata.dataset.DataSet</code> objects)
<i>cdl.obj</i>	Convenience module to access the DataLab objects ( <i>cdl.obj.SignalObj</i> or <i>cdl.obj.ImageObj</i> ) and related functions
<i>cdl.computation</i>	Computation functions, which operate on DataLab objects ( <i>cdl.obj.SignalObj</i> or <i>cdl.obj.ImageObj</i> )
<i>cdl.proxy</i>	Proxy objects to access the DataLab interface from a Python script or a remote application

## 3.1 Algorithms (*cdl.algorithms*)

This package contains the algorithms used by the DataLab project. Those algorithms operate directly on NumPy arrays and are designed to be used in the DataLab pipeline, but can be used independently as well.

### See also:

The *cdl.algorithms* package is the main entry point for the DataLab algorithms when manipulating NumPy arrays. See the *cdl.computation* package for algorithms that operate directly on DataLab objects (i.e. *cdl.obj.SignalObj* and *cdl.obj.ImageObj*).

The algorithms are organized in subpackages according to their purpose. The following subpackages are available:

- *cdl.algorithms.signal*: Signal processing algorithms
- *cdl.algorithms.image*: Image processing algorithms
- *cdl.algorithms.datatypes*: Data type conversion algorithms
- *cdl.algorithms.coordinates*: Coordinate conversion algorithms

### 3.1.1 Signal Processing Algorithms

`cdl.algorithms.signal.normalize(yin: ndarray, parameter: Literal['maximum', 'amplitude', 'area', 'energy', 'rms'] = 'maximum') → ndarray`

Normalize input array to a given parameter.

**Parameters**

- **yin** – Input array
- **parameter** – Normalization parameter. Defaults to “maximum”

**Returns**

Normalized array

`cdl.algorithms.signal.zero_padding(x: ndarray, y: ndarray, n: int) → tuple[ndarray, ndarray]`

Append n zeros at the end of the signal.

**Parameters**

- **x** – X data
- **y** – Y data
- **n** – Number of zeros to append

**Returns**

X data, Y data (tuple)

`cdl.algorithms.signal.fft1d(x: ndarray, y: ndarray, shift: bool = True) → tuple[ndarray, ndarray]`

Compute FFT on X,Y data.

**Parameters**

- **x** – X data
- **y** – Y data
- **shift** – Shift the zero frequency to the center of the spectrum. Defaults to True.

**Returns**

X data, Y data (tuple)

`cdl.algorithms.signal.ifft1d(x: ndarray, y: ndarray, shift: bool = True) → tuple[ndarray, ndarray]`

Compute iFFT on X,Y data.

**Parameters**

- **x** – X data
- **y** – Y data
- **shift** – Shift the zero frequency to the center of the spectrum. Defaults to True.

**Returns**

X data, Y data (tuple)

`cdl.algorithms.signal.magnitude_spectrum(x: ndarray, y: ndarray, log_scale: bool = False) → tuple[ndarray, ndarray]`

Compute magnitude spectrum.

**Parameters**

- **x** – X data

- **y** – Y data
- **log\_scale** – Use log scale. Defaults to False.

**Returns**

Magnitude spectrum (X data, Y data)

`cdl.algorithms.signal.phase_spectrum(x: ndarray, y: ndarray) → tuple[ndarray, ndarray]`

Compute phase spectrum.

**Parameters**

- **x** – X data
- **y** – Y data

**Returns**

Phase spectrum in degrees (X data, Y data)

`cdl.algorithms.signal.psd(x: ndarray, y: ndarray, log_scale: bool = False) → tuple[ndarray, ndarray]`

Compute Power Spectral Density (PSD), using the Welch method.

**Parameters**

- **x** – X data
- **y** – Y data
- **log\_scale** – Use log scale. Defaults to False.

**Returns**

X data, Y data (tuple)

**Return type**

Power Spectral Density (PSD)

`cdl.algorithms.signal.sort_frequencies(x: ndarray, y: ndarray) → ndarray`

Sort from X,Y data by computing FFT(y).

**Parameters**

- **x** – X data
- **y** – Y data

**Returns**

Sorted frequencies in ascending order

`cdl.algorithms.signal.peak_indices(y, thres: float = 0.3, min_dist: int = 1, thres_abs: bool = False) → ndarray`

Peak detection routine.

Finds the numeric index of the peaks in y by taking its first order difference. By using *thres* and *min\_dist* parameters, it is possible to reduce the number of detected peaks. y must be signed.

**Parameters**

- **y** (*ndarray (signed)*) – 1D amplitude data to search for peaks.
- **thres** (*float between [0., 1.]*) – Normalized threshold. Only the peaks with amplitude higher than the threshold will be detected.
- **min\_dist** (*int*) – Minimum distance between each detected peak. The peak with the highest amplitude is preferred to satisfy this constraint.

- **thres\_abs** (*boolean*) – If True, the thres value will be interpreted as an absolute value, instead of a normalized threshold.

**Returns**

Array containing the numeric indices of the peaks that were detected

**Return type**

ndarray

`cdl.algorithms.signal.xpeak(x: ndarray, y: ndarray) → float`

Return default peak X-position (assuming a single peak).

**Parameters**

- **x** – X data
- **y** – Y data

**Returns**

Peak X-position

`cdl.algorithms.signal.interpolate(x: ndarray, y: ndarray, xnew: ndarray, method: Literal['linear', 'spline', 'quadratic', 'cubic', 'barycentric', 'pchip'], fill_value: float | None = None) → ndarray`

Interpolate data.

**Parameters**

- **x** – X data
- **y** – Y data
- **xnew** – New X data
- **method** – Interpolation method
- **fill\_value** – Fill value. Defaults to None. This value is used to fill in for requested points outside of the X data range. It is only used if the method argument is ‘linear’, ‘cubic’ or ‘pchip’.

**Returns**

Interpolated Y data

`cdl.algorithms.signal.windowing(y: ndarray, method: Literal['barthann', 'bartlett', 'blackman', 'blackman-harris', 'bohman', 'boxcar', 'cosine', 'exponential', 'flat-top', 'hamming', 'hanning', 'lanczos', 'nuttall', 'parzen', 'rectangular', 'taylor', 'tukey', 'kaiser', 'gaussian'] = 'hamming', alpha: float = 0.5, beta: float = 14.0, sigma: float = 7.0) → ndarray`

Apply windowing to the input data.

**Parameters**

- **x** – X data
- **y** – Y data
- **method** – Windowing function. Defaults to “hamming”.
- **alpha** – Tukey window parameter. Defaults to 0.5.
- **beta** – Kaiser window parameter. Defaults to 14.0.
- **sigma** – Gaussian window parameter. Defaults to 7.0.

**Returns**

Windowed Y data

**class** `cdl.algorithms.signal.FitModel`

Curve fitting model base class

**abstractmethod** `func(x, amp, sigma, x0, y0)`

Return fitting function

**classmethod** `get_amp_from_amplitude(amplitude, sigma)`

Return amp from function amplitude and sigma

**classmethod** `amplitude(amp, sigma)`

Return function amplitude

**abstractmethod** `fwhm(amp, sigma)`

Return function FWHM

**classmethod** `half_max_segment(amp, sigma, x0, y0)`

Return segment coordinates for y=half-maximum intersection

**class** `cdl.algorithms.signal.GaussianModel`

1-dimensional Gaussian fit model

**classmethod** `func(x, amp, sigma, x0, y0)`

Return fitting function

**classmethod** `get_amp_from_amplitude(amplitude, sigma)`

Return amp from function amplitude and sigma

**classmethod** `amplitude(amp, sigma)`

Return function amplitude

**classmethod** `fwhm(amp, sigma)`

Return function FWHM

**class** `cdl.algorithms.signal.LorentzianModel`

1-dimensional Lorentzian fit model

**classmethod** `func(x, amp, sigma, x0, y0)`

Return fitting function

**classmethod** `get_amp_from_amplitude(amplitude, sigma)`

Return amp from function amplitude and sigma

**classmethod** `amplitude(amp, sigma)`

Return function amplitude

**classmethod** `fwhm(amp, sigma)`

Return function FWHM

**class** `cdl.algorithms.signal.VoigtModel`

1-dimensional Voigt fit model

**classmethod** `func(x, amp, sigma, x0, y0)`

Return fitting function

**classmethod** `fwhm(amp, sigma)`

Return function FWHM

`cdl.algorithms.signal.find_zero_crossings(y: ndarray) → ndarray`

Find the left indices of the zero-crossing intervals in the given array.

**Parameters**

**y** – Input array.

**Returns**

An array of indices where zero-crossings occur.

`cdl.algorithms.signal.find_first_x_at_y_value(x: ndarray, y: ndarray, y_value: float) → float`

Find the first x value where the signal reaches a given y value (interpolated).

**Parameters**

- **x** – x signal data
- **y** – y signal data (possibly non-monotonic)
- **y\_value** – the y value to find the corresponding x value for

**Returns**

The first interpolated x value at the given y, or *nan* if not found

`cdl.algorithms.signal.find_y_at_x_value(x: ndarray, y: ndarray, x_value: float) → float`

Find the y value at a given x value using linear interpolation.

**Parameters**

- **x** – Monotonic X data
- **y** – Y data (may contain NaNs)
- **x\_value** – The x value to find the corresponding y value for

**Returns**

The interpolated y value at the given x, or *nan* if not computable

`cdl.algorithms.signal.find_x_at_value(x: ndarray, y: ndarray, value: float) → ndarray`

Find the x values where the y value is the closest to the given value using linear interpolation to deduce the precise x value.

**Parameters**

- **x** – X data
- **y** – Y data
- **value** – Value to find

**Returns**

An array of x values where the y value is the closest to the given value (empty array if no zero crossing is found)

`cdl.algorithms.signal.bandwidth(data: ndarray, level: float = 3.0) → tuple[float, float, float, float]`

Compute the bandwidth of the signal at a given level.

**Parameters**

- **data** – X,Y data
- **level** – Level in dB at which the bandwidth is computed. Defaults to 3.0.

**Returns**

segment coordinates

**Return type**

Bandwidth of the signal at the given level

`cdl.algorithms.signal.contrast(y: ndarray) → float`

Compute contrast

**Parameters**

**y** – Input array

**Returns**

Contrast

`cdl.algorithms.signal.sinusoidal_model(x: ndarray, a: float, f: float, phi: float, offset: float) → ndarray`

Sinusoidal model function.

`cdl.algorithms.signal.sinusoidal_fit(x: ndarray, y: ndarray) → tuple[tuple[float, float, float, float], float]`

Fit a sinusoidal model to the input data.

**Parameters**

- **x** – X data
- **y** – Y data

**Returns**

A tuple containing the fit parameters (amplitude, frequency, phase, offset) and the residuals

`cdl.algorithms.signal.sinus_frequency(x: ndarray, y: ndarray) → float`

Compute the frequency of a sinusoidal signal.

**Parameters**

- **x** – x signal data
- **y** – y signal data

**Returns**

Frequency of the sinusoidal signal

`cdl.algorithms.signal.enob(x: ndarray, y: ndarray, full_scale: float = 1.0) → float`

Compute Effective Number of Bits (ENOB).

**Parameters**

- **x** – x signal data
- **y** – y signal data
- **full\_scale** – Full scale(V). Defaults to 1.0.

**Returns**

Effective Number of Bits (ENOB)

`cdl.algorithms.signal.sinad(x: ndarray, y: ndarray, full_scale: float = 1.0, unit: Literal['dBc', 'dBFS'] = 'dBc') → float`

Compute Signal-to-Noise and Distortion Ratio (SINAD).

**Parameters**

- **x** – x signal data
- **y** – y signal data
- **full\_scale** – Full scale(V). Defaults to 1.0.

- **unit** – Unit of the input data. Valid values are ‘dBc’ and ‘dBFS’. Defaults to ‘dBc’.

**Returns**

Signal-to-Noise and Distortion Ratio (SINAD)

```
cdl.algorithms.signal.thd(x: ndarray, y: ndarray, full_scale: float = 1.0, unit: Literal['dBc', 'dBFS'] = 'dBc',  
                          nb_harm: int = 5) → float
```

Compute Total Harmonic Distortion (THD).

**Parameters**

- **x** – x signal data
- **y** – y signal data
- **full\_scale** – Full scale(V). Defaults to 1.0.
- **unit** – Unit of the input data. Valid values are ‘dBc’ and ‘dBFS’. Defaults to ‘dBc’.
- **nb\_harm** – Number of harmonics to consider. Defaults to 5.

**Returns**

Total Harmonic Distortion (THD)

```
cdl.algorithms.signal.sfdr(x: ndarray, y: ndarray, full_scale: float = 1.0, unit: Literal['dBc', 'dBFS'] =  
                           'dBc') → float
```

Compute Spurious-Free Dynamic Range (SFDR).

**Parameters**

- **x** – x signal data
- **y** – y signal data
- **full\_scale** – Full scale(V). Defaults to 1.0.
- **unit** – Unit of the input data. Valid values are ‘dBc’ and ‘dBFS’. Defaults to ‘dBc’.

**Returns**

Spurious-Free Dynamic Range (SFDR)

```
cdl.algorithms.signal.snr(x: ndarray, y: ndarray, full_scale: float = 1.0, unit: Literal['dBc', 'dBFS'] = 'dBc')  
                          → float
```

Compute Signal-to-Noise Ratio (SNR).

**Parameters**

- **x** – x signal data
- **y** – y signal data
- **full\_scale** – Full scale(V). Defaults to 1.0.
- **unit** – Unit of the input data. Valid values are ‘dBc’ and ‘dBFS’. Defaults to ‘dBc’.

**Returns**

Signal-to-Noise Ratio (SNR)

```
cdl.algorithms.signal.sampling_period(x: ndarray) → float
```

Compute sampling period

**Parameters**

**x** – X data

**Returns**

Sampling period



`cdl.algorithms.signal.sampling_rate(x: ndarray) → float`

Compute mean sampling rate

**Parameters**

**x** – X data

**Returns**

Sampling rate

`cdl.algorithms.signal.full_width_at_y(data: ndarray, level: float) → tuple[float, float, float, float]`

Compute the full width at a given y level using zero-crossing method.

**Parameters**

- **data** – X,Y data
- **level** – The Y level at which to compute the width

**Returns**

Full width segment coordinates

`cdl.algorithms.signal.fwhm(data: ndarray, method: Literal['zero-crossing', 'gauss', 'lorentz', 'voigt'] = 'zero-crossing', xmin: float | None = None, xmax: float | None = None) → tuple[float, float, float, float]`

Compute Full Width at Half Maximum (FWHM) of the input data

**Parameters**

- **data** – X,Y data
- **method** – Calculation method. Two types of methods are supported: a zero-crossing method and fitting methods (based on various models: Gauss, Lorentz, Voigt). Defaults to “zero-crossing”.
- **xmin** – Lower X bound for the fitting. Defaults to None (no lower bound, i.e. the fitting starts from the first point).
- **xmax** – Upper X bound for the fitting. Defaults to None (no upper bound, i.e. the fitting ends at the last point)

**Returns**

FWHM segment coordinates

`cdl.algorithms.signal.fw1e2(data: ndarray) → tuple[float, float, float, float]`

Compute Full Width at 1/e of the input data (using a Gaussian model fitting).

**Parameters**

**data** – X,Y data

**Returns**

FW at 1/e<sup>2</sup> segment coordinates

`cdl.algorithms.signal.allan_variance(x: ndarray, y: ndarray, tau_values: ndarray) → ndarray`

Calculate the Allan variance for given time and measurement values at specified tau values.

**Parameters**

- **x** – Time array
- **y** – Measured values array
- **tau\_values** – Allan deviation time values

**Returns**

Allan variance values

`cdl.algorithms.signal.allan_deviation(x: ndarray, y: ndarray, tau_values: ndarray) → ndarray`

Calculate the Allan deviation for given time and measurement values at specified tau values.

**Parameters**

- **x** – Time array
- **y** – Measured values array
- **tau\_values** – Allan deviation time values

**Returns**

Allan deviation values

`cdl.algorithms.signal.overlapping_allan_variance(x: ndarray, y: ndarray, tau_values: ndarray) → ndarray`

Calculate the Overlapping Allan variance for given time and measurement values.

**Parameters**

- **x** – Time array
- **y** – Measured values array
- **tau\_values** – Allan deviation time values

**Returns**

Overlapping Allan variance values

`cdl.algorithms.signal.modified_allan_variance(x: ndarray, y: ndarray, tau_values: ndarray) → ndarray`

Calculate the Modified Allan variance for given time and measurement values.

**Parameters**

- **x** – Time array
- **y** – Measured values array
- **tau\_values** – Modified Allan deviation time values

**Returns**

Modified Allan variance values

`cdl.algorithms.signal.hadamard_variance(x: ndarray, y: ndarray, tau_values: ndarray) → ndarray`

Calculate the Hadamard variance for given time and measurement values.

**Parameters**

- **x** – Time array
- **y** – Measured values array
- **tau\_values** – Hadamard deviation time values

**Returns**

Hadamard variance values

`cdl.algorithms.signal.total_variance(x: ndarray, y: ndarray, tau_values: ndarray) → ndarray`

Calculate the Total variance for given time and measurement values.

**Parameters**

- **x** – Time array

- **y** – Measured values array
- **tau\_values** – Total variance time values

**Returns**

Total variance values

`cdl.algorithms.signal.time_deviation(x: ndarray, y: ndarray, tau_values: ndarray) → ndarray`

Calculate the Time Deviation (TDEV) for given time and measurement values.

**Parameters**

- **x** – Time array
- **y** – Measured values array
- **tau\_values** – Time deviation time values

**Returns**

Time deviation values

### 3.1.2 Image Processing Algorithms

`cdl.algorithms.image.scale_data_to_min_max(data: ndarray, zmin: float | int, zmax: float | int) → ndarray`

Scale array *data* to fit [zmin, zmax] dynamic range

**Parameters**

- **data** – Input data
- **zmin** – Minimum value of output data
- **zmax** – Maximum value of output data

**Returns**

Scaled data

`cdl.algorithms.image.normalize(data: ndarray, parameter: Literal['maximum', 'amplitude', 'area', 'energy', 'rms'] = 'maximum') → ndarray`

Normalize input array to a given parameter.

**Parameters**

- **data** – Input data
- **parameter** – Normalization parameter (default: “maximum”)

**Returns**

Normalized array

`cdl.algorithms.image.zero_padding(image: ndarray, rows: int = 0, cols: int = 0, position: Literal['bottom-right', 'center'] = 'bottom-right') → ndarray`

Zero-pad a 2D image by adding rows and/or columns.

**Parameters**

- **image** – 2D input image (grayscale)
- **rows** – Number of rows to add in total (default: 0)
- **cols** – Number of columns to add in total (default: 0)
- **position** – Padding placement strategy: - “bottom-right”: all padding is added to the bottom and right - “center”: padding is split equally on top/bottom and left/right

**Returns**

The padded 2D image as a NumPy array.

**Raises**

**ValueError** – If the input is not a 2D array or if padding values are negative.

`cdl.algorithms.image.fft2d(z: ndarray, shift: bool = True) → ndarray`

Compute FFT of complex array *z*

**Parameters**

- **z** – Input data
- **shift** – Shift zero frequency to center (default: True)

**Returns**

FFT of input data

`cdl.algorithms.image.ifft2d(z: ndarray, shift: bool = True) → ndarray`

Compute inverse FFT of complex array *z*

**Parameters**

- **z** – Input data
- **shift** – Shift zero frequency to center (default: True)

**Returns**

Inverse FFT of input data

`cdl.algorithms.image.magnitude_spectrum(z: ndarray, log_scale: bool = False) → ndarray`

Compute magnitude spectrum of complex array *z*

**Parameters**

- **z** – Input data
- **log\_scale** – Use log scale (default: False)

**Returns**

Magnitude spectrum of input data

`cdl.algorithms.image.phase_spectrum(z: ndarray) → ndarray`

Compute phase spectrum of complex array *z*

**Parameters**

**z** – Input data

**Returns**

Phase spectrum of input data (in degrees)

`cdl.algorithms.image.psd(z: ndarray, log_scale: bool = False) → ndarray`

Compute power spectral density of complex array *z*

**Parameters**

- **z** – Input data
- **log\_scale** – Use log scale (default: False)

**Returns**

Power spectral density of input data

`cdl.algorithms.image.binning(data: ndarray, sx: int, sy: int, operation: Literal['sum', 'average', 'median', 'min', 'max'], dtype=None) → ndarray`

Perform image pixel binning

#### Parameters

- **data** – Input data
- **sx** – Binning size along x (number of pixels to bin together)
- **sy** – Binning size along y (number of pixels to bin together)
- **operation** – Binning operation
- **dtype** – Output data type (default: None, i.e. same as input)

#### Returns

Binned data

`cdl.algorithms.image.flatfield(rawdata: ndarray, flatdata: ndarray, threshold: float | None = None) → ndarray`

Compute flat-field correction

#### Parameters

- **rawdata** – Raw data
- **flatdata** – Flat-field data
- **threshold** – Threshold for flat-field correction (default: None)

#### Returns

Flat-field corrected data

`cdl.algorithms.image.get_centroid_fourier(data: ndarray) → tuple[float, float]`

Return image centroid using Fourier algorithm

#### Parameters

**data** – Input data

#### Returns

Centroid coordinates (row, col)

`cdl.algorithms.image.get_absolute_level(data: ndarray, level: float) → float`

Return absolute level

#### Parameters

- **data** – Input data
- **level** – Relative level (0.0 to 1.0)

#### Returns

Absolute level

`cdl.algorithms.image.get_enclosing_circle(data: ndarray, level: float = 0.5) → tuple[int, int, float]`

Return (x, y, radius) for the circle contour enclosing image values above threshold relative level (.5 means FWHM)

#### Parameters

- **data** – Input data
- **level** – Relative level (default: 0.5)

**Returns**

A tuple (x, y, radius)

**Raises**

**ValueError** – No contour was found

`cdl.algorithms.image.get_radial_profile(data: ndarray, center: tuple[int, int]) → tuple[ndarray, ndarray]`

Return radial profile of image data

**Parameters**

- **data** – Input data (2D array)
- **center** – Coordinates of the center of the profile (x, y)

**Returns**

Radial profile (X, Y) where X is the distance from the center (1D array) and Y is the average value of pixels at this distance (1D array)

`cdl.algorithms.image.distance_matrix(coords: list) → ndarray`

Return distance matrix from coords

**Parameters**

**coords** – List of coordinates

**Returns**

Distance matrix

`cdl.algorithms.image.get_2d_peaks_coords(data: ndarray, size: int | None = None, level: float = 0.5) → ndarray`

Detect peaks in image data, return coordinates.

If neighborhoods size is None, default value is the highest value between 50 pixels and the 1/40th of the smallest image dimension.

Detection threshold level is relative to difference between data maximum and minimum values.

**Parameters**

- **data** – Input data
- **size** – Neighborhood size (default: None)
- **level** – Relative level (default: 0.5)

**Returns**

Coordinates of peaks

`cdl.algorithms.image.get_contour_shapes(data: ndarray | MaskedArray, shape: Literal['circle', 'ellipse', 'polygon'] = 'ellipse', level: float = 0.5) → ndarray`

Find iso-valued contours in a 2D array, above relative level (.5 means FWHM), then fit contours with shape ('ellipse' or 'circle')

**Parameters**

- **data** – Input data
- **shape** – Shape to fit. Default is 'ellipse'
- **level** – Relative level (default: 0.5)

**Returns**

Coordinates of shapes

```
cdl.algorithms.image.get_hough_circle_peaks(data: ndarray, min_radius: float | None = None,
                                           max_radius: float | None = None, nb_radius: int | None =
                                           None, min_distance: int = 1) → ndarray
```

Detect peaks in image from circle Hough transform, return circle coordinates.

#### Parameters

- **data** – Input data
- **min\_radius** – Minimum radius (default: None)
- **max\_radius** – Maximum radius (default: None)
- **nb\_radius** – Number of radii (default: None)
- **min\_distance** – Minimum distance between circles (default: 1)

#### Returns

Coordinates of circles

```
cdl.algorithms.image.find_blobs_dog(data: ndarray, min_sigma: float = 1, max_sigma: float = 30, overlap:
float = 0.5, threshold_rel: float = 0.2, exclude_border: bool = True)
→ ndarray
```

Finds blobs in the given grayscale image using the Difference of Gaussians (DoG) method.

#### Parameters

- **data** – The grayscale input image.
- **min\_sigma** – The minimum blob radius in pixels.
- **max\_sigma** – The maximum blob radius in pixels.
- **overlap** – The minimum overlap between two blobs in pixels. For instance, if two blobs are detected with radii of 10 and 12 respectively, and the **overlap** is set to 0.5, then the area of the smaller blob will be ignored and only the area of the larger blob will be returned.
- **threshold\_rel** – The absolute lower bound for scale space maxima. Local maxima smaller than **threshold\_rel** are ignored. Reduce this to detect blobs with less intensities.
- **exclude\_border** – If **True**, exclude blobs from detection if they are too close to the border of the image. Border size is **min\_sigma**.

#### Returns

Coordinates of blobs

```
cdl.algorithms.image.find_blobs_doh(data: ndarray, min_sigma: float = 1, max_sigma: float = 30, overlap:
float = 0.5, log_scale: bool = False, threshold_rel: float = 0.2) →
ndarray
```

Finds blobs in the given grayscale image using the Determinant of Hessian (DoH) method.

#### Parameters

- **data** – The grayscale input image.
- **min\_sigma** – The minimum blob radius in pixels.
- **max\_sigma** – The maximum blob radius in pixels.
- **overlap** – The minimum overlap between two blobs in pixels. For instance, if two blobs are detected with radii of 10 and 12 respectively, and the **overlap** is set to 0.5, then the area of the smaller blob will be ignored and only the area of the larger blob will be returned.
- **log\_scale** – If **True**, the radius of each blob is returned as `sqrt(sigma)` for each detected blob.

- **threshold\_rel** – The absolute lower bound for scale space maxima. Local maxima smaller than **threshold\_rel** are ignored. Reduce this to detect blobs with less intensities.

**Returns**

Coordinates of blobs

```
cdl.algorithms.image.find_blobs_log(data: ndarray, min_sigma: float = 1, max_sigma: float = 30, overlap: float = 0.5, log_scale: bool = False, threshold_rel: float = 0.2, exclude_border: bool = True) → ndarray
```

Finds blobs in the given grayscale image using the Laplacian of Gaussian (LoG) method.

**Parameters**

- **data** – The grayscale input image.
- **min\_sigma** – The minimum blob radius in pixels.
- **max\_sigma** – The maximum blob radius in pixels.
- **overlap** – The minimum overlap between two blobs in pixels. For instance, if two blobs are detected with radii of 10 and 12 respectively, and the **overlap** is set to 0.5, then the area of the smaller blob will be ignored and only the area of the larger blob will be returned.
- **log\_scale** – If True, the radius of each blob is returned as `sqrt(sigma)` for each detected blob.
- **threshold\_rel** – The absolute lower bound for scale space maxima. Local maxima smaller than **threshold\_rel** are ignored. Reduce this to detect blobs with less intensities.
- **exclude\_border** – If True, exclude blobs from detection if they are too close to the border of the image. Border size is **min\_sigma**.

**Returns**

Coordinates of blobs

```
cdl.algorithms.image.remove_overlapping_disks(coords: ndarray) → ndarray
```

Remove overlapping disks among coordinates

**Parameters**

**coords** – The coordinates of the disks

**Returns**

The coordinates of the disks with overlapping disks removed

```
cdl.algorithms.image.find_blobs_opencv(data: ndarray, min_threshold: float | None = None, max_threshold: float | None = None, min_repeatability: int | None = None, min_dist_between_blobs: float | None = None, filter_by_color: bool | None = None, blob_color: int | None = None, filter_by_area: bool | None = None, min_area: float | None = None, max_area: float | None = None, filter_by_circularity: bool | None = None, min_circularity: float | None = None, max_circularity: float | None = None, filter_by_inertia: bool | None = None, min_inertia_ratio: float | None = None, max_inertia_ratio: float | None = None, filter_by_convexity: bool | None = None, min_convexity: float | None = None, max_convexity: float | None = None) → ndarray
```

Finds blobs in the given grayscale image using OpenCV's SimpleBlobDetector.

**Parameters**

- **data** – The grayscale input image.



- **min\_threshold** – The minimum blob intensity.
- **max\_threshold** – The maximum blob intensity.
- **min\_repeatability** – The minimum number of times a blob is detected before it is reported.
- **min\_dist\_between\_blobs** – The minimum distance between blobs.
- **filter\_by\_color** – If True, blobs are filtered by color.
- **blob\_color** – The color of the blobs to filter by.
- **filter\_by\_area** – If True, blobs are filtered by area.
- **min\_area** – The minimum blob area.
- **max\_area** – The maximum blob area.
- **filter\_by\_circularity** – If True, blobs are filtered by circularity.
- **min\_circularity** – The minimum blob circularity.
- **max\_circularity** – The maximum blob circularity.
- **filter\_by\_inertia** – If True, blobs are filtered by inertia.
- **min\_inertia\_ratio** – The minimum blob inertia ratio.
- **max\_inertia\_ratio** – The maximum blob inertia ratio.
- **filter\_by\_convexity** – If True, blobs are filtered by convexity.
- **min\_convexity** – The minimum blob convexity.
- **max\_convexity** – The maximum blob convexity.

**Returns**

Coordinates of blobs

### 3.1.3 Data Type Conversion Algorithms

`cdl.algorithms.datatypes.is_integer_dtype(dtype: dtype) → bool`

Return True if data type is an integer type

**Parameters****dtype** – Data type to check**Returns**

True if data type is an integer type

`cdl.algorithms.datatypes.is_complex_dtype(dtype: dtype) → bool`

Return True if data type is a complex type

**Parameters****dtype** – Data type to check**Returns**

True if data type is a complex type

`cdl.algorithms.datatypes.clip_astype(data: ndarray, dtype: dtype) → ndarray`

Convert array to a new data type, after having clipped values to the new data type's range if it is an integer type. If data type is not integer, this is equivalent to `data.astype(dtype)`.

**Parameters**

- **data** – Array to convert
- **dtype** – Data type to convert to

**Returns**

Array converted to new data type

### 3.1.4 Coordinate Conversion Algorithms

`cdl.algorithms.coordinates.circle_to_diameter(xc: float, yc: float, r: float) → tuple[float, float, float, float]`

Convert circle center and radius to X diameter coordinates

**Parameters**

- **xc** – Circle center X coordinate
- **yc** – Circle center Y coordinate
- **r** – Circle radius

**Returns**

Circle X diameter coordinates

**Return type**

`tuple`

`cdl.algorithms.coordinates.array_circle_to_diameter(data: ndarray) → ndarray`

Convert circle center and radius to X diameter coordinates (array version)

**Parameters**

**data** – Circle center and radius, in the form of a 2D array (N, 3)

**Returns**

Circle X diameter coordinates, in the form of a 2D array (N, 4)

`cdl.algorithms.coordinates.circle_to_center_radius(x0: float, y0: float, x1: float, y1: float) → tuple[float, float, float]`

Convert circle X diameter coordinates to center and radius

**Parameters**

- **x0** – Diameter start X coordinate
- **y0** – Diameter start Y coordinate
- **x1** – Diameter end X coordinate
- **y1** – Diameter end Y coordinate

**Returns**

Circle center and radius

**Return type**

`tuple`

`cdl.algorithms.coordinates.array_circle_to_center_radius(data: ndarray) → ndarray`

Convert circle X diameter coordinates to center and radius (array version)

**Parameters**

**data** – Circle X diameter coordinates, in the form of a 2D array (N, 4)

**Returns**

Circle center and radius, in the form of a 2D array (N, 3)

`cdl.algorithms.coordinates.ellipse_to_diameters(xc: float, yc: float, a: float, b: float, theta: float) → tuple[float, float, float, float, float, float, float, float]`

Convert ellipse center, axes and angle to X/Y diameters coordinates

**Parameters**

- **xc** – Ellipse center X coordinate
- **yc** – Ellipse center Y coordinate
- **a** – Ellipse half larger axis
- **b** – Ellipse half smaller axis
- **theta** – Ellipse angle

**Returns**

Ellipse X/Y diameters (major/minor axes) coordinates

`cdl.algorithms.coordinates.array_ellipse_to_diameters(data: ndarray) → ndarray`

Convert ellipse center, axes and angle to X/Y diameters coordinates (array version)

**Parameters**

**data** – Ellipse center, axes and angle, in the form of a 2D array (N, 5)

**Returns**

**Ellipse X/Y diameters (major/minor axes) coordinates,**  
in the form of a 2D array (N, 8)

`cdl.algorithms.coordinates.ellipse_to_center_axes_angle(x0: float, y0: float, x1: float, y1: float, x2: float, y2: float, x3: float, y3: float) → tuple[float, float, float, float, float]`

Convert ellipse X/Y diameters coordinates to center, axes and angle

**Parameters**

- **x0** – major axis start X coordinate
- **y0** – major axis start Y coordinate
- **x1** – major axis end X coordinate
- **y1** – major axis end Y coordinate
- **x2** – minor axis start X coordinate
- **y2** – minor axis start Y coordinate
- **x3** – minor axis end X coordinate
- **y3** – minor axis end Y coordinate

**Returns**

Ellipse center, axes and angle

`cdl.algorithms.coordinates.array_ellipse_to_center_axes_angle(data: ndarray) → ndarray`

Convert ellipse X/Y diameters coordinates to center, axes and angle (array version)

**Parameters**

**data** – Ellipse X/Y diameters coordinates, in the form of a 2D array (N, 8)

**Returns**

Ellipse center, axes and angle, in the form of a 2D array (N, 5)

```
cdl.algorithms.coordinates.cartesian2polar(x: ndarray, y: ndarray, unit: Literal['rad', 'deg'] = 'rad') →  
tuple[ndarray, ndarray]
```

Convert Cartesian coordinates to polar coordinates.

**Parameters**

- **x** – Cartesian x-coordinate.
- **y** – Cartesian y-coordinate.
- **unit** – Unit of the angle ('rad' or 'deg').

**Returns**

Polar coordinates (r, theta) where r is the radius and theta is the angle.

```
cdl.algorithms.coordinates.polar2cartesian(r: ndarray, theta: ndarray, unit: Literal['rad', 'deg'] = 'rad')  
→ tuple[ndarray, ndarray]
```

Convert polar coordinates to Cartesian coordinates.

**Parameters**

- **r** – Polar radius.
- **theta** – Polar angle.
- **unit** – Unit of the angle ('rad' or 'deg').

**Returns**

Cartesian coordinates (x, y) where x is the x-coordinate and y is the y-coordinate.

---

**Note:** Negative radius values are not supported. They will be set to 0.

---

## 3.2 Parameters (cdl.param)

The `cdl.param` module aims at providing all the dataset parameters that are used by the `cdl.computation` and `cdl.core.gui.processor` packages.

Those datasets are defined in other modules:

- `cdl.computation.base`
- `cdl.computation.image`
- `cdl.computation.signal`

The `cdl.param` module is thus a convenient way to import all the sets of parameters at once.

As a matter of fact, the following import statement is equivalent to the previous one:

```
# Original import statement  
from cdl.computation.base import MovingAverageParam  
from cdl.computation.signal import PolynomialFitParam  
from cdl.computation.image.exposure import EqualizeHistParam  
  
# Equivalent import statement  
from cdl.param import MovingAverageParam, PolynomialFitParam, EqualizeHistParam
```

### 3.2.1 Introduction to *DataSet* parameters

The datasets listed in the following sections are used to define the parameters necessary for the various computations and processing operations available in DataLab.

Each dataset is a subclass of `guidata.dataset.datatypes.DataSet` and thus needs to be instantiated before being used.

Here is a complete example of how to instantiate a dataset and access its parameters with the `cdl.param.BinningParam` dataset:

```
class cdl.param.BinningParam
    Binning parameters

    sx
        Cluster size (X). Number of adjacent pixels to be combined together along x-axis. Integer higher
        than 2. Default: 2.
        Type
            guidata.dataset.dataitems.IntItem

    sy
        Cluster size (Y). Number of adjacent pixels to be combined together along y-axis. Integer higher
        than 2. Default: 2.
        Type
            guidata.dataset.dataitems.IntItem

    operation
        Single choice from: 'sum', 'average', 'median', 'min', 'max'. Default: 'sum'.
        Type
            guidata.dataset.dataitems.ChoiceItem

    dtype_str
        Data type. Output image data type. Single choice from: 'dtype', 'float32', 'float64', 'com-
        plex128', 'int32', 'int16', 'uint16', 'uint8'. Default: 'dtype'.
        Type
            guidata.dataset.dataitems.ChoiceItem

    change_pixel_size
        Change pixel size so that overall image size remains the same. Default: False.
        Type
            guidata.dataset.dataitems.BoolItem

    classmethod create(sx: int, sy: int, operation: str, dtype_str: str, change_pixel_size: bool) →
        cdl.computation.image.BinningParam

        Returns a new instance of BinningParam with the fields set to the given values.

    Parameters
        • sx (int) – Cluster size (X). Number of adjacent pixels to be combined together
          along x-axis. Integer higher than 2. Default: 2.
        • sy (int) – Cluster size (Y). Number of adjacent pixels to be combined together
          along y-axis. Integer higher than 2. Default: 2.
        • operation (str) – Single choice from: 'sum', 'average', 'median', 'min', 'max'.
          Default: 'sum'.
        • dtype_str (str) – Data type. Output image data type. Single choice from:
          'dtype', 'float32', 'float64', 'complex128', 'int32', 'int16', 'uint16', 'uint8'. De-
          fault: 'dtype'.
        • change_pixel_size (bool) – Change pixel size so that overall image size re-
          mains the same. Default: False.
```

**Returns**

New instance of `BinningParam`.

---

**Note:** To instantiate a new `BinningParam` dataset, you can use the classmethod `BinningParam.create()` like this:

```
BinningParam.create(sx=2, sy=2, operation='sum', dtype_str='dtype', change_
↪ pixel_size=False)
```

You can also first instantiate a default `BinningParam` and then set the fields like this:

```
param = BinningParam()
param.sx = 2
param.sy = 2
param.operation = 'sum'
param.dtype_str = 'dtype'
param.change_pixel_size = False
```

---

### 3.2.2 Common parameters

**class** `cdl.param.ArithmeticParam`

Arithmetic parameters

**operator**

Single choice from: '+', '-', '×', '/'. Default: '+'.  
**Type**  
`guidata.dataset.dataitems.ChoiceItem`

**factor**

Default: 1.0.  
**Type**  
`guidata.dataset.dataitems.FloatItem`

**constant**

Default: 0.0.  
**Type**  
`guidata.dataset.dataitems.FloatItem`

**operation**

Default: ‘.’.  
**Type**  
`guidata.dataset.dataitems.StringItem`

**restore\_dtype**

Result. Default: True.  
**Type**  
`guidata.dataset.dataitems.BoolItem`

**classmethod create**(*operator: str, factor: float, constant: float, operation: str, restore\_dtype: bool*) → *cdl.computation.base.ArithmeticParam*

Returns a new instance of `ArithmeticParam` with the fields set to the given values.

#### Parameters

- **operator** (*str*) – Single choice from: '+', '-', '×', '/'. Default: '+'.
- **factor** (*float*) – Default: 1.0.
- **constant** (*float*) – Default: 0.0.
- **operation** (*str*) – Default: ''.
- **restore\_dtype** (*bool*) – Result. Default: True.

#### Returns

New instance of `ArithmeticParam`.

**get\_operation**() → *str*

Return the operation string

**update\_operation**(*\_item, \_value*)

Update the operation item

**class** `cdl.param.ClipParam`

Data clipping parameters

#### **lower**

Lower clipping value. Default: None.

#### Type

*guidata.dataset.dataitems.FloatItem*

#### **upper**

Upper clipping value. Default: None.

#### Type

*guidata.dataset.dataitems.FloatItem*

**classmethod create**(*lower: float, upper: float*) → *cdl.computation.base.ClipParam*

Returns a new instance of `ClipParam` with the fields set to the given values.

#### Parameters

- **lower** (*float*) – Lower clipping value. Default: None.
- **upper** (*float*) – Upper clipping value. Default: None.

#### Returns

New instance of `ClipParam`.

**class** `cdl.param.ConstantParam`

Parameter used to set a constant value to used in operations

#### **value**

Constant value. Default: None.

#### Type

*guidata.dataset.dataitems.FloatItem*

**classmethod** **create**(*value: float*) → *cdl.computation.base.ConstantParam*

Returns a new instance of `ConstantParam` with the fields set to the given values.

**Parameters**

**value** (*float*) – Constant value. Default: None.

**Returns**

New instance of `ConstantParam`.

**class** `cdl.param.FFTParam`

FFT parameters

**shift**

Shift zero frequency to center. Default: None.

**Type**

`guidata.dataset.dataitems.BoolItem`

**classmethod** **create**(*shift: bool*) → *cdl.computation.base.FFTParam*

Returns a new instance of `FFTParam` with the fields set to the given values.

**Parameters**

**shift** (*bool*) – Shift zero frequency to center. Default: None.

**Returns**

New instance of `FFTParam`.

**class** `cdl.param.GaussianParam`

Gaussian filter parameters

**sigma**

. Default: 1.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**classmethod** **create**(*sigma: float*) → *cdl.computation.base.GaussianParam*

Returns a new instance of `GaussianParam` with the fields set to the given values.

**Parameters**

**sigma** (*float*) – . Default: 1.0.

**Returns**

New instance of `GaussianParam`.

**class** `cdl.param.HistogramParam`

Histogram parameters

**bins**

Number of bins. Integer higher than 1. Default: 256.

**Type**

`guidata.dataset.dataitems.IntItem`

**lower**

Lower limit. Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`



**upper**

Upper limit. Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`

**classmethod** `create(bins: int, lower: float, upper: float) → cdl.computation.base.HistogramParam`

Returns a new instance of `HistogramParam` with the fields set to the given values.

**Parameters**

- **bins** (*int*) – Number of bins. Integer higher than 1. Default: 256.
- **lower** (*float*) – Lower limit. Default: None.
- **upper** (*float*) – Upper limit. Default: None.

**Returns**

New instance of `HistogramParam`.

**get\_suffix**(*data: ndarray*) → *str*

Return suffix for the histogram computation

**Parameters**

**data** – data array

**class** `cdl.param.MovingAverageParam`

Moving average parameters

**n**

Size of the moving window. Integer higher than 1. Default: 3.

**Type**

`guidata.dataset.dataitems.IntItem`

**mode**

Mode of the filter: - 'reflect': reflect the data at the boundary - 'constant': pad with a constant value - 'nearest': pad with the nearest value - 'mirror': reflect the data at the boundary with the data itself - 'wrap': circular boundary. Single choice from: 'reflect', 'constant', 'nearest', 'mirror', 'wrap'. Default: 'reflect'.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**classmethod** `create(n: int, mode: str) → cdl.computation.base.MovingAverageParam`

Returns a new instance of `MovingAverageParam` with the fields set to the given values.

**Parameters**

- **n** (*int*) – Size of the moving window. Integer higher than 1. Default: 3.
- **mode** (*str*) – Mode of the filter: - 'reflect': reflect the data at the boundary - 'constant': pad with a constant value - 'nearest': pad with the nearest value - 'mirror': reflect the data at the boundary with the data itself - 'wrap': circular boundary. Single choice from: 'reflect', 'constant', 'nearest', 'mirror', 'wrap'. Default: 'reflect'.

**Returns**

New instance of `MovingAverageParam`.

**class** `cdl.param.MovingMedianParam`

Moving median parameters

**n**

Size of the moving window. Integer higher than 1, odd. Default: 3.

**Type**

`guidata.dataset.dataitems.IntItem`

**mode**

Mode of the filter: - 'reflect': reflect the data at the boundary - 'constant': pad with a constant value - 'nearest': pad with the nearest value - 'mirror': reflect the data at the boundary with the data itself - 'wrap': circular boundary. Single choice from: 'reflect', 'constant', 'nearest', 'mirror', 'wrap'. Default: 'nearest'.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**classmethod** `create(n: int, mode: str) → cdl.computation.base.MovingMedianParam`

Returns a new instance of `MovingMedianParam` with the fields set to the given values.

**Parameters**

- **n** (*int*) – Size of the moving window. Integer higher than 1, odd. Default: 3.
- **mode** (*str*) – Mode of the filter: - 'reflect': reflect the data at the boundary - 'constant': pad with a constant value - 'nearest': pad with the nearest value - 'mirror': reflect the data at the boundary with the data itself - 'wrap': circular boundary. Single choice from: 'reflect', 'constant', 'nearest', 'mirror', 'wrap'. Default: 'nearest'.

**Returns**

New instance of `MovingMedianParam`.

**class** `cdl.param.NormalizeParam`

Normalize parameters

**method**

Normalize with respect to. Single choice from: 'maximum', 'amplitude', 'area', 'energy', 'rms'. Default: 'maximum'.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**classmethod** `create(method: str) → cdl.computation.base.NormalizeParam`

Returns a new instance of `NormalizeParam` with the fields set to the given values.

**Parameters**

**method** (*str*) – Normalize with respect to. Single choice from: 'maximum', 'amplitude', 'area', 'energy', 'rms'. Default: 'maximum'.

**Returns**

New instance of `NormalizeParam`.

**class** `cdl.param.SpectrumParam`

Spectrum parameters

**log**

Default: False.

**Type**

`guidata.dataset.dataitems.BoolItem`

**classmethod** `create(log: bool) → cdl.computation.base.SpectrumParam`

Returns a new instance of `SpectrumParam` with the fields set to the given values.

**Parameters****log** (*bool*) – Default: False.**Returns**New instance of `SpectrumParam`.

### 3.2.3 Signal parameters

**class** `cdl.param.AllanVarianceParam`

Allan variance parameters

**max\_tau**

Max . Integer higher than 1, unit: pts. Default: 100.

**Type**`guidata.dataset.dataitems.IntItem`**classmethod** `create(max_tau: int) → cdl.computation.signal.AllanVarianceParam`Returns a new instance of `AllanVarianceParam` with the fields set to the given values.**Parameters****max\_tau** (*int*) – Max . Integer higher than 1, unit: pts. Default: 100.**Returns**New instance of `AllanVarianceParam`.**class** `cdl.param.AngleUnitParam`

Choice of angle unit.

**unit**

Angle unit. Single choice from: 'rad', 'deg'. Default: 'rad'.

**Type**`guidata.dataset.dataitems.ChoiceItem`**classmethod** `create(unit: str) → cdl.computation.signal.AngleUnitParam`Returns a new instance of `AngleUnitParam` with the fields set to the given values.**Parameters****unit** (*str*) – Angle unit. Single choice from: 'rad', 'deg'. Default: 'rad'.**Returns**New instance of `AngleUnitParam`.**class** `cdl.param.BandPassFilterParam`

Band-pass filter parameters

**method**

Filter method. Single choice from: 'bessel', 'butter', 'cheby1', 'cheby2', 'ellip'. Default: 'bessel'.

**Type**`guidata.dataset.dataitems.ChoiceItem`**order**

Filter order. Integer higher than 1. Default: 3.

**Type**`guidata.dataset.dataitems.IntItem`

**f\_cut0**

Low cutoff frequency. Float higher than 0, non zero, unit: hz. Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`

**f\_cut1**

High cutoff frequency. Float higher than 0, non zero, unit: hz. Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`

**rp**

Passband ripple. Float higher than 0, non zero, unit: db. Default: 1.

**Type**

`guidata.dataset.dataitems.FloatItem`

**rs**

Stopband attenuation. Float higher than 0, non zero, unit: db. Default: 1.

**Type**

`guidata.dataset.dataitems.FloatItem`

**classmethod** **create**(*method: str, order: int, f\_cut0: float, f\_cut1: float, rp: float, rs: float*) → *cdl.computation.signal.BandPassFilterParam*

Returns a new instance of `BandPassFilterParam` with the fields set to the given values.

**Parameters**

- **method** (*str*) – Filter method. Single choice from: ‘bessel’, ‘butter’, ‘cheby1’, ‘cheby2’, ‘ellip’. Default: ‘bessel’.
- **order** (*int*) – Filter order. Integer higher than 1. Default: 3.
- **f\_cut0** (*float*) – Low cutoff frequency. Float higher than 0, non zero, unit: hz. Default: None.
- **f\_cut1** (*float*) – High cutoff frequency. Float higher than 0, non zero, unit: hz. Default: None.
- **rp** (*float*) – Passband ripple. Float higher than 0, non zero, unit: db. Default: 1.
- **rs** (*float*) – Stopband attenuation. Float higher than 0, non zero, unit: db. Default: 1.

**Returns**

New instance of `BandPassFilterParam`.

**class** `cdl.param.BandStopFilterParam`

Band-stop filter parameters

**method**

Filter method. Single choice from: ‘bessel’, ‘butter’, ‘cheby1’, ‘cheby2’, ‘ellip’. Default: ‘bessel’.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**order**

Filter order. Integer higher than 1. Default: 3.

**Type**

`guidata.dataset.dataitems.IntItem`

**f\_cut0**

Low cutoff frequency. Float higher than 0, non zero, unit: hz. Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`

**f\_cut1**

High cutoff frequency. Float higher than 0, non zero, unit: hz. Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`

**rp**

Passband ripple. Float higher than 0, non zero, unit: db. Default: 1.

**Type**

`guidata.dataset.dataitems.FloatItem`

**rs**

Stopband attenuation. Float higher than 0, non zero, unit: db. Default: 1.

**Type**

`guidata.dataset.dataitems.FloatItem`

**classmethod** `create(method: str, order: int, f_cut0: float, f_cut1: float, rp: float, rs: float) → cdl.computation.signal.BandStopFilterParam`

Returns a new instance of `BandStopFilterParam` with the fields set to the given values.

**Parameters**

- **method** (*str*) – Filter method. Single choice from: ‘bessel’, ‘butter’, ‘cheby1’, ‘cheby2’, ‘ellip’. Default: ‘bessel’.
- **order** (*int*) – Filter order. Integer higher than 1. Default: 3.
- **f\_cut0** (*float*) – Low cutoff frequency. Float higher than 0, non zero, unit: hz. Default: None.
- **f\_cut1** (*float*) – High cutoff frequency. Float higher than 0, non zero, unit: hz. Default: None.
- **rp** (*float*) – Passband ripple. Float higher than 0, non zero, unit: db. Default: 1.
- **rs** (*float*) – Stopband attenuation. Float higher than 0, non zero, unit: db. Default: 1.

**Returns**

New instance of `BandStopFilterParam`.

**class** `cdl.param.DataTypeSParam`

Convert signal data type parameters

**dtype\_str**

Destination data type. Output image data type. Single choice from: ‘float32’, ‘float64’, ‘complex128’. Default: ‘float32’.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**classmethod** `create(dtype_str: str) → cdl.computation.signal.DataTypeSParam`

Returns a new instance of `DataTypeSParam` with the fields set to the given values.

**Parameters**

**dtype\_str** (*str*) – Destination data type. Output image data type. Single choice from: ‘float32’, ‘float64’, ‘complex128’. Default: ‘float32’.

**Returns**

New instance of `DataTypeSParam`.

**class** `cdl.param.DetrendingParam`

Detrending parameters

**method**

Detrending method. Single choice from: ‘linear’, ‘constant’. Default: ‘linear’.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**classmethod** `create`(*method: str*) → `cdl.computation.signal.DetrendingParam`

Returns a new instance of `DetrendingParam` with the fields set to the given values.

**Parameters**

**method** (*str*) – Detrending method. Single choice from: ‘linear’, ‘constant’. Default: ‘linear’.

**Returns**

New instance of `DetrendingParam`.

**class** `cdl.param.DynamicParam`

Parameters for dynamic range computation (ENOB, SNR, SINAD, THD, SFDR)

**full\_scale**

Float higher than 0.0, unit: v. Default: 0.16.

**Type**

`guidata.dataset.dataitems.FloatItem`

**unit**

Unit for sinad. Single choice from: ‘dBc’, ‘dBFS’. Default: ‘dBc’.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**nb\_harm**

Number of harmonics. Number of harmonics to consider for thd. Integer higher than 1. Default: 5.

**Type**

`guidata.dataset.dataitems.IntItem`

**classmethod** `create`(*full\_scale: float, unit: str, nb\_harm: int*) → `cdl.computation.signal.DynamicParam`

Returns a new instance of `DynamicParam` with the fields set to the given values.

**Parameters**

- **full\_scale** (*float*) – Float higher than 0.0, unit: v. Default: 0.16.
- **unit** (*str*) – Unit for sinad. Single choice from: ‘dBc’, ‘dBFS’. Default: ‘dBc’.
- **nb\_harm** (*int*) – Number of harmonics. Number of harmonics to consider for thd. Integer higher than 1. Default: 5.

**Returns**

New instance of `DynamicParam`.

**class** `cdl.param.AbscissaParam`

Abscissa parameter

**x**

Abscissa. Default: 0.

**Type**`guidata.dataset.dataitems.FloatItem`**classmethod** `create(x: float) → cdl.computation.signal.AbscissaParam`Returns a new instance of `AbscissaParam` with the fields set to the given values.**Parameters****x** (*float*) – Abscissa. Default: 0.**Returns**New instance of `AbscissaParam`.**class** `cdl.param.OrdinateParam`

Ordinate parameter

**y**

Ordinate. Default: 0.

**Type**`guidata.dataset.dataitems.FloatItem`**classmethod** `create(y: float) → cdl.computation.signal.OrdinateParam`Returns a new instance of `OrdinateParam` with the fields set to the given values.**Parameters****y** (*float*) – Ordinate. Default: 0.**Returns**New instance of `OrdinateParam`.**class** `cdl.param.FWHMParam`

FWHM parameters

**method**

Single choice from: ‘zero-crossing’, ‘gauss’, ‘lorentz’, ‘voigt’. Default: ‘zero-crossing’.

**Type**`guidata.dataset.dataitems.ChoiceItem`**xmin** $X_{\text{MIN}}$ . Lower x boundary (empty for no limit, i.e. Start of the signal). Default: None.**Type**`guidata.dataset.dataitems.FloatItem`**xmax** $X_{\text{MAX}}$ . Upper x boundary (empty for no limit, i.e. End of the signal). Default: None.**Type**`guidata.dataset.dataitems.FloatItem`**classmethod** `create(method: str, xmin: float, xmax: float) → cdl.computation.signal.FWHMParam`Returns a new instance of `FWHMParam` with the fields set to the given values.**Parameters**

- **method** (*str*) – Single choice from: ‘zero-crossing’, ‘gauss’, ‘lorentz’, ‘voigt’. Default: ‘zero-crossing’.
- **xmin** (*float*) –  $X_{\text{MIN}}$ . Lower x boundary (empty for no limit, i.e. Start of the signal). Default: None.
- **xmax** (*float*) –  $X_{\text{MAX}}$ . Upper x boundary (empty for no limit, i.e. End of the signal). Default: None.

**Returns**

New instance of FWHMParam.

**class** `cdl.param.HighPassFilterParam`

High-pass filter parameters

**method**

Filter method. Single choice from: ‘bessel’, ‘butter’, ‘cheby1’, ‘cheby2’, ‘ellip’. Default: ‘bessel’.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**order**

Filter order. Integer higher than 1. Default: 3.

**Type**

`guidata.dataset.dataitems.IntItem`

**f\_cut0**

Low cutoff frequency. Float higher than 0, non zero, unit: hz. Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`

**f\_cut1**

High cutoff frequency. Float higher than 0, non zero, unit: hz. Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`

**rp**

Passband ripple. Float higher than 0, non zero, unit: db. Default: 1.

**Type**

`guidata.dataset.dataitems.FloatItem`

**rs**

Stopband attenuation. Float higher than 0, non zero, unit: db. Default: 1.

**Type**

`guidata.dataset.dataitems.FloatItem`

**classmethod** `create`(*method: str, order: int, f\_cut0: float, f\_cut1: float, rp: float, rs: float*) → `cdl.computation.signal.HighPassFilterParam`

Returns a new instance of `HighPassFilterParam` with the fields set to the given values.

**Parameters**

- **method** (*str*) – Filter method. Single choice from: ‘bessel’, ‘butter’, ‘cheby1’, ‘cheby2’, ‘ellip’. Default: ‘bessel’.
- **order** (*int*) – Filter order. Integer higher than 1. Default: 3.



- **f\_cut0** (*float*) – Low cutoff frequency. Float higher than 0, non zero, unit: hz. Default: None.
- **f\_cut1** (*float*) – High cutoff frequency. Float higher than 0, non zero, unit: hz. Default: None.
- **rp** (*float*) – Passband ripple. Float higher than 0, non zero, unit: db. Default: 1.
- **rs** (*float*) – Stopband attenuation. Float higher than 0, non zero, unit: db. Default: 1.

**Returns**

New instance of HighPassFilterParam.

**class** cdl.param.InterpolationParam

Interpolation parameters

**method**

Interpolation method. Single choice from: 'linear', 'spline', 'quadratic', 'cubic', 'barycentric', 'pchip'. Default: 'linear'.

**Type**

guidata.dataset.dataitems.ChoiceItem

**fill\_value**

Value to use for points outside the interpolation domain (used only with linear, cubic and pchip methods). Default: None.

**Type**

guidata.dataset.dataitems.FloatItem

**classmethod** **create**(*method*: str, *fill\_value*: float) → cdl.computation.signal.InterpolationParam

Returns a new instance of InterpolationParam with the fields set to the given values.

**Parameters**

- **method** (*str*) – Interpolation method. Single choice from: 'linear', 'spline', 'quadratic', 'cubic', 'barycentric', 'pchip'. Default: 'linear'.
- **fill\_value** (*float*) – Value to use for points outside the interpolation domain (used only with linear, cubic and pchip methods). Default: None.

**Returns**

New instance of InterpolationParam.

**class** cdl.param.LowPassFilterParam

Low-pass filter parameters

**method**

Filter method. Single choice from: 'bessel', 'butter', 'cheby1', 'cheby2', 'ellip'. Default: 'bessel'.

**Type**

guidata.dataset.dataitems.ChoiceItem

**order**

Filter order. Integer higher than 1. Default: 3.

**Type**

guidata.dataset.dataitems.IntItem

**f\_cut0**

Low cutoff frequency. Float higher than 0, non zero, unit: hz. Default: None.

**Type**`guidata.dataset.dataitems.FloatItem`**f\_cut1**

High cutoff frequency. Float higher than 0, non zero, unit: hz. Default: None.

**Type**`guidata.dataset.dataitems.FloatItem`**rp**

Passband ripple. Float higher than 0, non zero, unit: db. Default: 1.

**Type**`guidata.dataset.dataitems.FloatItem`**rs**

Stopband attenuation. Float higher than 0, non zero, unit: db. Default: 1.

**Type**`guidata.dataset.dataitems.FloatItem`

**classmethod create**(*method: str, order: int, f\_cut0: float, f\_cut1: float, rp: float, rs: float*) → *cdl.computation.signal.LowPassFilterParam*

Returns a new instance of `LowPassFilterParam` with the fields set to the given values.

**Parameters**

- **method** (*str*) – Filter method. Single choice from: ‘bessel’, ‘butter’, ‘cheby1’, ‘cheby2’, ‘ellip’. Default: ‘bessel’.
- **order** (*int*) – Filter order. Integer higher than 1. Default: 3.
- **f\_cut0** (*float*) – Low cutoff frequency. Float higher than 0, non zero, unit: hz. Default: None.
- **f\_cut1** (*float*) – High cutoff frequency. Float higher than 0, non zero, unit: hz. Default: None.
- **rp** (*float*) – Passband ripple. Float higher than 0, non zero, unit: db. Default: 1.
- **rs** (*float*) – Stopband attenuation. Float higher than 0, non zero, unit: db. Default: 1.

**Returns**

New instance of `LowPassFilterParam`.

**class** `cdl.param.PeakDetectionParam`

Peak detection parameters

**threshold**

Integer between 0 and 100, unit: %. Default: 30.

**Type**`guidata.dataset.dataitems.IntItem`**min\_dist**

Minimum distance. Integer higher than 1, unit: points. Default: 1.

**Type**`guidata.dataset.dataitems.IntItem`

**classmethod** `create(threshold: int, min_dist: int) → cdl.computation.signal.PeakDetectionParam`

Returns a new instance of `PeakDetectionParam` with the fields set to the given values.

**Parameters**

- **threshold** (*int*) – Integer between 0 and 100, unit: %. Default: 30.
- **min\_dist** (*int*) – Minimum distance. Integer higher than 1, unit: points. Default: 1.

**Returns**

New instance of `PeakDetectionParam`.

**class** `cdl.param.PolynomialFitParam`

Polynomial fitting parameters

**degree**

Integer between 1 and 10. Default: 3.

**Type**

`guidata.dataset.dataitems.IntItem`

**classmethod** `create(degree: int) → cdl.computation.signal.PolynomialFitParam`

Returns a new instance of `PolynomialFitParam` with the fields set to the given values.

**Parameters**

**degree** (*int*) – Integer between 1 and 10. Default: 3.

**Returns**

New instance of `PolynomialFitParam`.

**class** `cdl.param.PowerParam`

Power parameters

**power**

Default: 2.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**classmethod** `create(power: float) → cdl.computation.signal.PowerParam`

Returns a new instance of `PowerParam` with the fields set to the given values.

**Parameters**

**power** (*float*) – Default: 2.0.

**Returns**

New instance of `PowerParam`.

**class** `cdl.param.ResamplingParam`

Resample parameters

**method**

Interpolation method. Single choice from: 'linear', 'spline', 'quadratic', 'cubic', 'barycentric', 'pchip'. Default: 'linear'.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**fill\_value**

Value to use for points outside the interpolation domain (used only with linear, cubic and pchip methods). Default: None.

**Type**`guidata.dataset.dataitems.FloatItem`**xmin** $X_{\min}$ . Default: None.**Type**`guidata.dataset.dataitems.FloatItem`**xmax** $X_{\max}$ . Default: None.**Type**`guidata.dataset.dataitems.FloatItem`**mode**

Single choice from: 'dx', 'nbpts'. Default: 'nbpts'.

**Type**`guidata.dataset.dataitems.ChoiceItem`**dx**

X. Default: None.

**Type**`guidata.dataset.dataitems.FloatItem`**nbpts**

Number of points. Default: None.

**Type**`guidata.dataset.dataitems.IntItem`

**classmethod create**(*method: str, fill\_value: float, xmin: float, xmax: float, mode: str, dx: float, nbpts: int*)  
→ `cdl.computation.signal.ResamplingParam`

Returns a new instance of `ResamplingParam` with the fields set to the given values.**Parameters**

- **method** (*str*) – Interpolation method. Single choice from: 'linear', 'spline', 'quadratic', 'cubic', 'barycentric', 'pchip'. Default: 'linear'.
- **fill\_value** (*float*) – Value to use for points outside the interpolation domain (used only with linear, cubic and pchip methods). Default: None.
- **xmin** (*float*) –  $X_{\min}$ . Default: None.
- **xmax** (*float*) –  $X_{\max}$ . Default: None.
- **mode** (*str*) – Single choice from: 'dx', 'nbpts'. Default: 'nbpts'.
- **dx** (*float*) – X. Default: None.
- **nbpts** (*int*) – Number of points. Default: None.

**Returns**New instance of `ResamplingParam`.**class** `cdl.param.WindowingParam`

Windowing parameters

**method**

Single choice from: 'barthann', 'bartlett', 'blackman', 'blackman-harris', 'bohman', 'boxcar', 'cosine', 'exponential', 'flat-top', 'gaussian', 'hamming', 'hanning', 'kaiser', 'lanczos', 'nuttall', 'parzen', 'rectangular', 'taylor', 'tukey'. Default: 'hamming'.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**alpha**

Shape parameter of the tukey windowing function. Default: 0.5.

**Type**

`guidata.dataset.dataitems.FloatItem`

**beta**

Shape parameter of the kaiser windowing function. Default: 14.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**sigma**

Shape parameter of the gaussian windowing function. Default: 0.5.

**Type**

`guidata.dataset.dataitems.FloatItem`

**classmethod** `create(method: str, alpha: float, beta: float, sigma: float) →`  
`cdl.computation.signal.WindowingParam`

Returns a new instance of `WindowingParam` with the fields set to the given values.

**Parameters**

- **method** (*str*) – Single choice from: 'barthann', 'bartlett', 'blackman', 'blackman-harris', 'bohman', 'boxcar', 'cosine', 'exponential', 'flat-top', 'gaussian', 'hamming', 'hanning', 'kaiser', 'lanczos', 'nuttall', 'parzen', 'rectangular', 'taylor', 'tukey'. Default: 'hamming'.
- **alpha** (*float*) – Shape parameter of the tukey windowing function. Default: 0.5.
- **beta** (*float*) – Shape parameter of the kaiser windowing function. Default: 14.0.
- **sigma** (*float*) – Shape parameter of the gaussian windowing function. Default: 0.5.

**Returns**

New instance of `WindowingParam`.

**class** `cdl.param.XYCalibrateParam`

Signal calibration parameters

**axis**

Calibrate. Single choice from: 'x', 'y'. Default: 'y'.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**a**

Default: 1.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**b**

Default: 0.0.

**Type**`guidata.dataset.dataitems.FloatItem`**classmethod** `create(axis: str, a: float, b: float) → cdl.computation.signal.XYCalibrateParam`Returns a new instance of `XYCalibrateParam` with the fields set to the given values.**Parameters**

- **axis** (`str`) – Calibrate. Single choice from: ‘x’, ‘y’. Default: ‘y’.
- **a** (`float`) – Default: 1.0.
- **b** (`float`) – Default: 0.0.

**Returns**New instance of `XYCalibrateParam`.**class** `cdl.param.ZeroPadding1DParam`

Zero padding parameters

**strategy**

Single choice from: ‘next\_pow2’, ‘double’, ‘triple’, ‘custom’. Default: ‘custom’.

**Type**`guidata.dataset.dataitems.ChoiceItem`**n**

Number of points. Number of points to add. Integer higher than 1. Default: None.

**Type**`guidata.dataset.dataitems.IntItem`**classmethod** `create(strategy: str, n: int) → cdl.computation.signal.ZeroPadding1DParam`Returns a new instance of `ZeroPadding1DParam` with the fields set to the given values.**Parameters**

- **strategy** (`str`) – Single choice from: ‘next\_pow2’, ‘double’, ‘triple’, ‘custom’. Default: ‘custom’.
- **n** (`int`) – Number of points. Number of points to add. Integer higher than 1. Default: None.

**Returns**New instance of `ZeroPadding1DParam`.**update\_from\_signal**(*obj*: `SignalObj`) → `None`

Update parameters from signal

**choice\_callback**(*item*, *value*)

Callback for choice item

### 3.2.4 Image parameters

#### Base image parameters

**class** `cdl.param.AverageProfileParam`

Average horizontal or vertical profile parameters

**direction**

Single choice from: 'horizontal', 'vertical'. Default: 'horizontal'.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**row1**

Integer higher than 0. Default: 0.

**Type**

`guidata.dataset.dataitems.IntItem`

**row2**

Integer higher than -1. Default: -1.

**Type**

`guidata.dataset.dataitems.IntItem`

**col1**

Column 1. Integer higher than 0. Default: 0.

**Type**

`guidata.dataset.dataitems.IntItem`

**col2**

Column 2. Integer higher than -1. Default: -1.

**Type**

`guidata.dataset.dataitems.IntItem`

**classmethod** `create(direction: str, row1: int, row2: int, col1: int, col2: int) →`  
`cdl.computation.image.AverageProfileParam`

Returns a new instance of `AverageProfileParam` with the fields set to the given values.

**Parameters**

- **direction** (*str*) – Single choice from: 'horizontal', 'vertical'. Default: 'horizontal'.
- **row1** (*int*) – Integer higher than 0. Default: 0.
- **row2** (*int*) – Integer higher than -1. Default: -1.
- **col1** (*int*) – Column 1. Integer higher than 0. Default: 0.
- **col2** (*int*) – Column 2. Integer higher than -1. Default: -1.

**Returns**

New instance of `AverageProfileParam`.

**class** `cdl.param.BinningParam`

Binning parameters

**sx**

Cluster size (X). Number of adjacent pixels to be combined together along x-axis. Integer higher than 2. Default: 2.

**Type**

`guidata.dataset.dataitems.IntItem`

**sy**

Cluster size (Y). Number of adjacent pixels to be combined together along y-axis. Integer higher than 2. Default: 2.

**Type**

`guidata.dataset.dataitems.IntItem`

**operation**

Single choice from: 'sum', 'average', 'median', 'min', 'max'. Default: 'sum'.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**dtype\_str**

Data type. Output image data type. Single choice from: 'dtype', 'float32', 'float64', 'complex128', 'int32', 'int16', 'uint16', 'uint8'. Default: 'dtype'.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**change\_pixel\_size**

Change pixel size so that overall image size remains the same. Default: False.

**Type**

`guidata.dataset.dataitems.BoolItem`

**classmethod** `create(sx: int, sy: int, operation: str, dtype_str: str, change_pixel_size: bool) → cdl.computation.image.BinningParam`

Returns a new instance of `BinningParam` with the fields set to the given values.

**Parameters**

- **sx** (*int*) – Cluster size (X). Number of adjacent pixels to be combined together along x-axis. Integer higher than 2. Default: 2.
- **sy** (*int*) – Cluster size (Y). Number of adjacent pixels to be combined together along y-axis. Integer higher than 2. Default: 2.
- **operation** (*str*) – Single choice from: 'sum', 'average', 'median', 'min', 'max'. Default: 'sum'.
- **dtype\_str** (*str*) – Data type. Output image data type. Single choice from: 'dtype', 'float32', 'float64', 'complex128', 'int32', 'int16', 'uint16', 'uint8'. Default: 'dtype'.
- **change\_pixel\_size** (*bool*) – Change pixel size so that overall image size remains the same. Default: False.

**Returns**

New instance of `BinningParam`.

**class** `cdl.param.ButterworthParam`

Butterworth filter parameters



**cut\_off**

Cut-off frequency ratio. Cut-off frequency ratio. Float between 0.0 and 0.5. Default: 0.005.

**Type**

`guidata.dataset.dataitems.FloatItem`

**high\_pass**

If true, apply high-pass filter instead of low-pass. Default: False.

**Type**

`guidata.dataset.dataitems.BoolItem`

**order**

Order of the butterworth filter. Integer higher than 1. Default: 2.

**Type**

`guidata.dataset.dataitems.IntItem`

**classmethod** `create(cut_off: float, high_pass: bool, order: int) → cdl.computation.image.ButterworthParam`

Returns a new instance of `ButterworthParam` with the fields set to the given values.

**Parameters**

- **cut\_off** (*float*) – Cut-off frequency ratio. Cut-off frequency ratio. Float between 0.0 and 0.5. Default: 0.005.
- **high\_pass** (*bool*) – If true, apply high-pass filter instead of low-pass. Default: False.
- **order** (*int*) – Order of the butterworth filter. Integer higher than 1. Default: 2.

**Returns**

New instance of `ButterworthParam`.

**class** `cdl.param.DataTypeIParam`

Convert image data type parameters

**dtype\_str**

Destination data type. Output image data type. Single choice from: 'float32', 'float64', 'complex128', 'int32', 'int16', 'uint16', 'uint8'. Default: 'float32'.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**classmethod** `create(dtype_str: str) → cdl.computation.image.DataTypeIParam`

Returns a new instance of `DataTypeIParam` with the fields set to the given values.

**Parameters**

**dtype\_str** (*str*) – Destination data type. Output image data type. Single choice from: 'float32', 'float64', 'complex128', 'int32', 'int16', 'uint16', 'uint8'. Default: 'float32'.

**Returns**

New instance of `DataTypeIParam`.

**class** `cdl.param.FlatFieldParam`

Flat-field parameters

**threshold**

Default: 0.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**classmethod** `create(threshold: float) → cdl.computation.image.FlatFieldParam`

Returns a new instance of `FlatFieldParam` with the fields set to the given values.

**Parameters**

**threshold** (*float*) – Default: 0.0.

**Returns**

New instance of `FlatFieldParam`.

**class** `cdl.param.GridParam`

Grid parameters

**direction**

Distribute over. Single choice from: ‘col’, ‘row’. Default: ‘col’.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**cols**

Columns. Integer, non zero. Default: 1.

**Type**

`guidata.dataset.dataitems.IntItem`

**rows**

Integer, non zero. Default: 1.

**Type**

`guidata.dataset.dataitems.IntItem`

**colspac**

Column spacing. Float higher than 0.0. Default: 0.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**rowspac**

Row spacing. Float higher than 0.0. Default: 0.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**classmethod** `create(direction: str, cols: int, rows: int, colspac: float, rowspac: float) → cdl.computation.image.GridParam`

Returns a new instance of `GridParam` with the fields set to the given values.

**Parameters**

- **direction** (*str*) – Distribute over. Single choice from: ‘col’, ‘row’. Default: ‘col’.
- **cols** (*int*) – Columns. Integer, non zero. Default: 1.
- **rows** (*int*) – Integer, non zero. Default: 1.
- **colspac** (*float*) – Column spacing. Float higher than 0.0. Default: 0.0.
- **rowspac** (*float*) – Row spacing. Float higher than 0.0. Default: 0.0.

**Returns**

New instance of `GridParam`.

**class** `cdl.param.HoughCircleParam`

Circle Hough transform parameters

**min\_radius**Radius<sub>min</sub>. Integer higher than 0, non zero, unit: pixels. Default: None.**Type**`guidata.dataset.dataitems.IntItem`**max\_radius**Radius<sub>max</sub>. Integer higher than 0, non zero, unit: pixels. Default: None.**Type**`guidata.dataset.dataitems.IntItem`**min\_distance**

Minimal distance. Integer higher than 0. Default: None.

**Type**`guidata.dataset.dataitems.IntItem`

**classmethod** `create(min_radius: int, max_radius: int, min_distance: int) →`  
`cdl.computation.image.HoughCircleParam`

Returns a new instance of `HoughCircleParam` with the fields set to the given values.**Parameters**

- **min\_radius** (*int*) – Radius<sub>min</sub>. Integer higher than 0, non zero, unit: pixels. Default: None.
- **max\_radius** (*int*) – Radius<sub>max</sub>. Integer higher than 0, non zero, unit: pixels. Default: None.
- **min\_distance** (*int*) – Minimal distance. Integer higher than 0. Default: None.

**Returns**New instance of `HoughCircleParam`.**class** `cdl.param.LineProfileParam`

Horizontal or vertical profile parameters

**direction**

Single choice from: 'horizontal', 'vertical'. Default: 'horizontal'.

**Type**`guidata.dataset.dataitems.ChoiceItem`**row**

Integer higher than 0. Default: 0.

**Type**`guidata.dataset.dataitems.IntItem`**col**

Column. Integer higher than 0. Default: 0.

**Type**`guidata.dataset.dataitems.IntItem`

**classmethod** `create(direction: str, row: int, col: int) → cdl.computation.image.LineProfileParam`

Returns a new instance of `LineProfileParam` with the fields set to the given values.

**Parameters**

- **direction** (*str*) – Single choice from: ‘horizontal’, ‘vertical’. Default: ‘horizontal’.
- **row** (*int*) – Integer higher than 0. Default: 0.
- **col** (*int*) – Column. Integer higher than 0. Default: 0.

**Returns**

New instance of `LineProfileParam`.

**class** `cdl.param.LogP1Param`

Log10 parameters

**n**

Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`

**classmethod** `create(n: float) → cdl.computation.image.LogP1Param`

Returns a new instance of `LogP1Param` with the fields set to the given values.

**Parameters**

**n** (*float*) – Default: None.

**Returns**

New instance of `LogP1Param`.

**class** `cdl.param.RadialProfileParam`

Radial profile parameters

**center**

Center position. Single choice from: ‘centroid’, ‘center’, ‘user’. Default: ‘centroid’.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**x0**

$X_{\text{Center}}$ . Float, unit: pixel. Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`

**y0**

$X_{\text{Center}}$ . Float, unit: pixel. Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`

**classmethod** `create(center: str, x0: float, y0: float) → cdl.computation.image.RadialProfileParam`

Returns a new instance of `RadialProfileParam` with the fields set to the given values.

**Parameters**

- **center** (*str*) – Center position. Single choice from: ‘centroid’, ‘center’, ‘user’. Default: ‘centroid’.
- **x0** (*float*) –  $X_{\text{Center}}$ . Float, unit: pixel. Default: None.

- **y0** (*float*) –  $X_{\text{Center}}$ . Float, unit: pixel. Default: None.

**Returns**

New instance of RadialProfileParam.

**update\_from\_image**(*obj*: ImageObj) → None

Update parameters from image

**choice\_callback**(*item*, *value*)

Callback for choice item

**class** cdl.param.ResizeParam

Resize parameters

**zoom**

Default: None.

**Type**

guidata.dataset.dataitems.FloatItem

**mode**

Single choice from: 'constant', 'nearest', 'reflect', 'wrap'. Default: 'constant'.

**Type**

guidata.dataset.dataitems.ChoiceItem

**cval**

Value used for points outside the boundaries of the input if mode is 'constant'. Default: 0.0.

**Type**

guidata.dataset.dataitems.FloatItem

**prefilter**

Default: True.

**Type**

guidata.dataset.dataitems.BoolItem

**order**

Spline interpolation order. Integer between 0 and 5. Default: 3.

**Type**

guidata.dataset.dataitems.IntItem

**classmethod** **create**(*zoom*: float, *mode*: str, *cval*: float, *prefilter*: bool, *order*: int) →  
cdl.computation.image.ResizeParam

Returns a new instance of ResizeParam with the fields set to the given values.

**Parameters**

- **zoom** (*float*) – Default: None.
- **mode** (*str*) – Single choice from: 'constant', 'nearest', 'reflect', 'wrap'. Default: 'constant'.
- **cval** (*float*) – Value used for points outside the boundaries of the input if mode is 'constant'. Default: 0.0.
- **prefilter** (*bool*) – Default: True.
- **order** (*int*) – Spline interpolation order. Integer between 0 and 5. Default: 3.

**Returns**

New instance of ResizeParam.

**class** `cdl.param.RotateParam`

Rotate parameters

**angle**

Angle (°). Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`

**mode**

Single choice from: 'constant', 'nearest', 'reflect', 'wrap'. Default: 'constant'.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**cval**

Value used for points outside the boundaries of the input if mode is 'constant'. Default: 0.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**reshape**

Reshape the output array so that the input array is contained completely in the output. Default: False.

**Type**

`guidata.dataset.dataitems.BoolItem`

**prefilter**

Default: True.

**Type**

`guidata.dataset.dataitems.BoolItem`

**order**

Spline interpolation order. Integer between 0 and 5. Default: 3.

**Type**

`guidata.dataset.dataitems.IntItem`

**classmethod** `create`(*angle: float, mode: str, cval: float, reshape: bool, prefilter: bool, order: int*) → `cdl.computation.image.RotateParam`

Returns a new instance of `RotateParam` with the fields set to the given values.

**Parameters**

- **angle** (*float*) – Angle (°). Default: None.
- **mode** (*str*) – Single choice from: 'constant', 'nearest', 'reflect', 'wrap'. Default: 'constant'.
- **cval** (*float*) – Value used for points outside the boundaries of the input if mode is 'constant'. Default: 0.0.
- **reshape** (*bool*) – Reshape the output array so that the input array is contained completely in the output. Default: False.
- **prefilter** (*bool*) – Default: True.
- **order** (*int*) – Spline interpolation order. Integer between 0 and 5. Default: 3.

**Returns**

New instance of `RotateParam`.

**class** `cdl.param.SegmentProfileParam`

Segment profile parameters

**row1**

Start row. Integer higher than 0. Default: 0.

**Type**`guidata.dataset.dataitems.IntItem`**col1**

Start column. Integer higher than 0. Default: 0.

**Type**`guidata.dataset.dataitems.IntItem`**row2**

End row. Integer higher than 0. Default: 0.

**Type**`guidata.dataset.dataitems.IntItem`**col2**

End column. Integer higher than 0. Default: 0.

**Type**`guidata.dataset.dataitems.IntItem`

**classmethod** **create**(*row1: int, col1: int, row2: int, col2: int*) →  
`cdl.computation.image.SegmentProfileParam`

Returns a new instance of `SegmentProfileParam` with the fields set to the given values.**Parameters**

- **row1** (*int*) – Start row. Integer higher than 0. Default: 0.
- **col1** (*int*) – Start column. Integer higher than 0. Default: 0.
- **row2** (*int*) – End row. Integer higher than 0. Default: 0.
- **col2** (*int*) – End column. Integer higher than 0. Default: 0.

**Returns**New instance of `SegmentProfileParam`.**class** `cdl.param.ZCalibrateParam`

Image linear calibration parameters

**a**

Default: 1.0.

**Type**`guidata.dataset.dataitems.FloatItem`**b**

Default: 0.0.

**Type**`guidata.dataset.dataitems.FloatItem`

**classmethod** `create(a: float, b: float) → cdl.computation.image.ZCalibrateParam`

Returns a new instance of `ZCalibrateParam` with the fields set to the given values.

**Parameters**

- **a** (*float*) – Default: 1.0.
- **b** (*float*) – Default: 0.0.

**Returns**

New instance of `ZCalibrateParam`.

**class** `cdl.param.ZeroPadding2DParam`

Zero padding parameters for 2D images

**strategy**

Padding strategy. Single choice from: ‘next\_pow2’, ‘multiple\_of\_64’, ‘custom’. Default: ‘custom’.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**rows**

Rows to add. Integer higher than 0. Default: None.

**Type**

`guidata.dataset.dataitems.IntItem`

**cols**

Columns to add. Integer higher than 0. Default: None.

**Type**

`guidata.dataset.dataitems.IntItem`

**position**

Padding position. Single choice from: ‘bottom-right’, ‘center’. Default: ‘bottom-right’.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**classmethod** `create(strategy: str, rows: int, cols: int, position: str) → cdl.computation.image.ZeroPadding2DParam`

Returns a new instance of `ZeroPadding2DParam` with the fields set to the given values.

**Parameters**

- **strategy** (*str*) – Padding strategy. Single choice from: ‘next\_pow2’, ‘multiple\_of\_64’, ‘custom’. Default: ‘custom’.
- **rows** (*int*) – Rows to add. Integer higher than 0. Default: None.
- **cols** (*int*) – Columns to add. Integer higher than 0. Default: None.
- **position** (*str*) – Padding position. Single choice from: ‘bottom-right’, ‘center’. Default: ‘bottom-right’.

**Returns**

New instance of `ZeroPadding2DParam`.

**update\_from\_image**(*obj*: `ImageObj`) → `None`

Update parameters from image

**choice\_callback**(*item*, *value*)

Callback to update padding values



## Detection parameters

### class `cdl.param.BlobDOGParam`

Blob detection using Difference of Gaussian method

#### **min\_sigma**

min. The minimum standard deviation for gaussian kernel. Keep this low to detect smaller blobs. Float higher than 0, non zero, unit: pixels. Default: 1.0.

#### Type

`guidata.dataset.dataitems.FloatItem`

#### **max\_sigma**

max. The maximum standard deviation for gaussian kernel. Keep this high to detect larger blobs. Float higher than 0, non zero, unit: pixels. Default: 30.0.

#### Type

`guidata.dataset.dataitems.FloatItem`

#### **threshold\_rel**

Relative threshold. Minimum intensity of blobs. Float between 0.0 and 1.0. Default: 0.2.

#### Type

`guidata.dataset.dataitems.FloatItem`

#### **overlap**

If two blobs overlap by a fraction greater than this value, the smaller blob is eliminated. Float between 0.0 and 1.0. Default: 0.5.

#### Type

`guidata.dataset.dataitems.FloatItem`

#### **exclude\_border**

If true, exclude blobs from the border of the image. Default: True.

#### Type

`guidata.dataset.dataitems.BoolItem`

**classmethod** `create(min_sigma: float, max_sigma: float, threshold_rel: float, overlap: float, exclude_border: bool) → cdl.computation.image.detection.BlobDOGParam`

Returns a new instance of `BlobDOGParam` with the fields set to the given values.

#### Parameters

- **min\_sigma** (*float*) – min. The minimum standard deviation for gaussian kernel. Keep this low to detect smaller blobs. Float higher than 0, non zero, unit: pixels. Default: 1.0.
- **max\_sigma** (*float*) – max. The maximum standard deviation for gaussian kernel. Keep this high to detect larger blobs. Float higher than 0, non zero, unit: pixels. Default: 30.0.
- **threshold\_rel** (*float*) – Relative threshold. Minimum intensity of blobs. Float between 0.0 and 1.0. Default: 0.2.
- **overlap** (*float*) – If two blobs overlap by a fraction greater than this value, the smaller blob is eliminated. Float between 0.0 and 1.0. Default: 0.5.
- **exclude\_border** (*bool*) – If true, exclude blobs from the border of the image. Default: True.

#### Returns

New instance of `BlobDOGParam`.

**class** `cdl.param.BlobDOHParam`

Blob detection using Determinant of Hessian method

**min\_sigma**

min. The minimum standard deviation for gaussian kernel. Keep this low to detect smaller blobs. Float higher than 0, non zero, unit: pixels. Default: 1.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**max\_sigma**

max. The maximum standard deviation for gaussian kernel. Keep this high to detect larger blobs. Float higher than 0, non zero, unit: pixels. Default: 30.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**threshold\_rel**

Relative threshold. Minimum intensity of blobs. Float between 0.0 and 1.0. Default: 0.2.

**Type**

`guidata.dataset.dataitems.FloatItem`

**overlap**

If two blobs overlap by a fraction greater than this value, the smaller blob is eliminated. Float between 0.0 and 1.0. Default: 0.5.

**Type**

`guidata.dataset.dataitems.FloatItem`

**log\_scale**

If set intermediate values of standard deviations are interpolated using a logarithmic scale to the base 10. If not, linear interpolation is used. Default: False.

**Type**

`guidata.dataset.dataitems.BoolItem`

**classmethod** `create(min_sigma: float, max_sigma: float, threshold_rel: float, overlap: float, log_scale: bool) → cdl.computation.image.detection.BlobDOHParam`

Returns a new instance of `BlobDOHParam` with the fields set to the given values.

**Parameters**

- **min\_sigma** (*float*) – min. The minimum standard deviation for gaussian kernel. Keep this low to detect smaller blobs. Float higher than 0, non zero, unit: pixels. Default: 1.0.
- **max\_sigma** (*float*) – max. The maximum standard deviation for gaussian kernel. Keep this high to detect larger blobs. Float higher than 0, non zero, unit: pixels. Default: 30.0.
- **threshold\_rel** (*float*) – Relative threshold. Minimum intensity of blobs. Float between 0.0 and 1.0. Default: 0.2.
- **overlap** (*float*) – If two blobs overlap by a fraction greater than this value, the smaller blob is eliminated. Float between 0.0 and 1.0. Default: 0.5.
- **log\_scale** (*bool*) – If set intermediate values of standard deviations are interpolated using a logarithmic scale to the base 10. If not, linear interpolation is used. Default: False.

**Returns**

New instance of `BlobDOHParam`.

**class** `cdl.param.BlobLOGParam`

Blob detection using Laplacian of Gaussian method

**min\_sigma**

min. The minimum standard deviation for gaussian kernel. Keep this low to detect smaller blobs. Float higher than 0, non zero, unit: pixels. Default: 1.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**max\_sigma**

max. The maximum standard deviation for gaussian kernel. Keep this high to detect larger blobs. Float higher than 0, non zero, unit: pixels. Default: 30.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**threshold\_rel**

Relative threshold. Minimum intensity of blobs. Float between 0.0 and 1.0. Default: 0.2.

**Type**

`guidata.dataset.dataitems.FloatItem`

**overlap**

If two blobs overlap by a fraction greater than this value, the smaller blob is eliminated. Float between 0.0 and 1.0. Default: 0.5.

**Type**

`guidata.dataset.dataitems.FloatItem`

**log\_scale**

If set intermediate values of standard deviations are interpolated using a logarithmic scale to the base 10. If not, linear interpolation is used. Default: False.

**Type**

`guidata.dataset.dataitems.BoolItem`

**exclude\_border**

If true, exclude blobs from the border of the image. Default: True.

**Type**

`guidata.dataset.dataitems.BoolItem`

**classmethod** `create(min_sigma: float, max_sigma: float, threshold_rel: float, overlap: float, log_scale: bool, exclude_border: bool) → cdl.computation.image.detection.BlobLOGParam`

Returns a new instance of `BlobLOGParam` with the fields set to the given values.

**Parameters**

- **min\_sigma** (*float*) – min. The minimum standard deviation for gaussian kernel. Keep this low to detect smaller blobs. Float higher than 0, non zero, unit: pixels. Default: 1.0.
- **max\_sigma** (*float*) – max. The maximum standard deviation for gaussian kernel. Keep this high to detect larger blobs. Float higher than 0, non zero, unit: pixels. Default: 30.0.
- **threshold\_rel** (*float*) – Relative threshold. Minimum intensity of blobs. Float between 0.0 and 1.0. Default: 0.2.
- **overlap** (*float*) – If two blobs overlap by a fraction greater than this value, the smaller blob is eliminated. Float between 0.0 and 1.0. Default: 0.5.

- **log\_scale** (*bool*) – If set intermediate values of standard deviations are interpolated using a logarithmic scale to the base 10. If not, linear interpolation is used. Default: False.
- **exclude\_border** (*bool*) – If true, exclude blobs from the border of the image. Default: True.

**Returns**

New instance of BlobLOGParam.

**class** `cdl.param.BlobOpenCVParam`

Blob detection using OpenCV

**min\_threshold**

Min. threshold. The minimum threshold between local maxima and minima. This parameter does not affect the quality of the blobs, only the quantity. Lower thresholds result in larger numbers of blobs. Float higher than 0.0. Default: 10.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**max\_threshold**

Max. threshold. The maximum threshold between local maxima and minima. This parameter does not affect the quality of the blobs, only the quantity. Lower thresholds result in larger numbers of blobs. Float higher than 0.0. Default: 200.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**min\_repeatability**

Min. repeatability. The minimum number of times a blob needs to be detected in a sequence of images to be considered valid. Integer higher than 1. Default: 2.

**Type**

`guidata.dataset.dataitems.IntItem`

**min\_dist\_between\_blobs**

Min. distance between blobs. The minimum distance between two blobs. If blobs are found closer together than this distance, the smaller blob is removed. Float higher than 0.0, non zero. Default: 10.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**filter\_by\_color**

If true, the image is filtered by color instead of intensity. Default: True.

**Type**

`guidata.dataset.dataitems.BoolItem`

**blob\_color**

The color of the blobs to detect (0 for dark blobs, 255 for light blobs). Default: 0.

**Type**

`guidata.dataset.dataitems.IntItem`

**filter\_by\_area**

If true, the image is filtered by blob area. Default: True.

**Type**

`guidata.dataset.dataitems.BoolItem`

**min\_area**

Min. area. The minimum blob area. Float higher than 0.0. Default: 25.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**max\_area**

Max. area. The maximum blob area. Float higher than 0.0. Default: 500.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**filter\_by\_circularity**

If true, the image is filtered by blob circularity. Default: False.

**Type**

`guidata.dataset.dataitems.BoolItem`

**min\_circularity**

Min. circularity. The minimum circularity of the blobs. Float between 0.0 and 1.0. Default: 0.8.

**Type**

`guidata.dataset.dataitems.FloatItem`

**max\_circularity**

Max. circularity. The maximum circularity of the blobs. Float between 0.0 and 1.0. Default: 1.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**filter\_by\_inertia**

If true, the image is filtered by blob inertia. Default: False.

**Type**

`guidata.dataset.dataitems.BoolItem`

**min\_inertia\_ratio**

Min. inertia ratio. The minimum inertia ratio of the blobs. Float between 0.0 and 1.0. Default: 0.6.

**Type**

`guidata.dataset.dataitems.FloatItem`

**max\_inertia\_ratio**

Max. inertia ratio. The maximum inertia ratio of the blobs. Float between 0.0 and 1.0. Default: 1.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**filter\_by\_convexity**

If true, the image is filtered by blob convexity. Default: False.

**Type**

`guidata.dataset.dataitems.BoolItem`

**min\_convexity**

Min. convexity. The minimum convexity of the blobs. Float between 0.0 and 1.0. Default: 0.8.

**Type**

`guidata.dataset.dataitems.FloatItem`

**max\_convexity**

Max. convexity. The maximum convexity of the blobs. Float between 0.0 and 1.0. Default: 1.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**classmethod create**(*min\_threshold: float, max\_threshold: float, min\_repeatability: int, min\_dist\_between\_blobs: float, filter\_by\_color: bool, blob\_color: int, filter\_by\_area: bool, min\_area: float, max\_area: float, filter\_by\_circularity: bool, min\_circularity: float, max\_circularity: float, filter\_by\_inertia: bool, min\_inertia\_ratio: float, max\_inertia\_ratio: float, filter\_by\_convexity: bool, min\_convexity: float, max\_convexity: float*) → `cdl.computation.image.detection.BlobOpenCVParam`

Returns a new instance of `BlobOpenCVParam` with the fields set to the given values.

**Parameters**

- **min\_threshold** (*float*) – Min. threshold. The minimum threshold between local maxima and minima. This parameter does not affect the quality of the blobs, only the quantity. Lower thresholds result in larger numbers of blobs. Float higher than 0.0. Default: 10.0.
- **max\_threshold** (*float*) – Max. threshold. The maximum threshold between local maxima and minima. This parameter does not affect the quality of the blobs, only the quantity. Lower thresholds result in larger numbers of blobs. Float higher than 0.0. Default: 200.0.
- **min\_repeatability** (*int*) – Min. repeatability. The minimum number of times a blob needs to be detected in a sequence of images to be considered valid. Integer higher than 1. Default: 2.
- **min\_dist\_between\_blobs** (*float*) – Min. distance between blobs. The minimum distance between two blobs. If blobs are found closer together than this distance, the smaller blob is removed. Float higher than 0.0, non zero. Default: 10.0.
- **filter\_by\_color** (*bool*) – If true, the image is filtered by color instead of intensity. Default: True.
- **blob\_color** (*int*) – The color of the blobs to detect (0 for dark blobs, 255 for light blobs). Default: 0.
- **filter\_by\_area** (*bool*) – If true, the image is filtered by blob area. Default: True.
- **min\_area** (*float*) – Min. area. The minimum blob area. Float higher than 0.0. Default: 25.0.
- **max\_area** (*float*) – Max. area. The maximum blob area. Float higher than 0.0. Default: 500.0.
- **filter\_by\_circularity** (*bool*) – If true, the image is filtered by blob circularity. Default: False.
- **min\_circularity** (*float*) – Min. circularity. The minimum circularity of the blobs. Float between 0.0 and 1.0. Default: 0.8.
- **max\_circularity** (*float*) – Max. circularity. The maximum circularity of the blobs. Float between 0.0 and 1.0. Default: 1.0.
- **filter\_by\_inertia** (*bool*) – If true, the image is filtered by blob inertia. Default: False.
- **min\_inertia\_ratio** (*float*) – Min. inertia ratio. The minimum inertia ratio of the blobs. Float between 0.0 and 1.0. Default: 0.6.
- **max\_inertia\_ratio** (*float*) – Max. inertia ratio. The maximum inertia ratio of the blobs. Float between 0.0 and 1.0. Default: 1.0.

- **filter\_by\_convexity** (*bool*) – If true, the image is filtered by blob convexity. Default: False.
- **min\_convexity** (*float*) – Min. convexity. The minimum convexity of the blobs. Float between 0.0 and 1.0. Default: 0.8.
- **max\_convexity** (*float*) – Max. convexity. The maximum convexity of the blobs. Float between 0.0 and 1.0. Default: 1.0.

**Returns**

New instance of BlobOpenCVParam.

**class** `cdl.param.ContourShapeParam`

Contour shape parameters

**threshold**

Relative threshold. Detection threshold, relative to difference between data maximum and minimum. Float between 0.1 and 0.9. Default: 0.5.

**Type**

`guidata.dataset.dataitems.FloatItem`

**shape**

Single choice from: 'ellipse', 'circle', 'polygon'. Default: 'ellipse'.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**classmethod** `create(threshold: float, shape: str) → cdl.computation.image.detection.ContourShapeParam`

Returns a new instance of ContourShapeParam with the fields set to the given values.

**Parameters**

- **threshold** (*float*) – Relative threshold. Detection threshold, relative to difference between data maximum and minimum. Float between 0.1 and 0.9. Default: 0.5.
- **shape** (*str*) – Single choice from: 'ellipse', 'circle', 'polygon'. Default: 'ellipse'.

**Returns**

New instance of ContourShapeParam.

**class** `cdl.param.Peak2DDetectionParam`

Peak detection parameters

**threshold**

Relative threshold. Detection threshold, relative to difference between data maximum and minimum. Float between 0.1 and 0.9. Default: 0.5.

**Type**

`guidata.dataset.dataitems.FloatItem`

**size**

Neighborhoods size. Size of the sliding window used in maximum/minimum filtering algorithm. Integer higher than 1, unit: pixels. Default: 10.

**Type**

`guidata.dataset.dataitems.IntItem`

**create\_rois**

Default: True.

**Type**`guidata.dataset.dataitems.BoolItem`

**classmethod** **create**(*threshold: float, size: int, create\_rois: bool*) →  
*cdl.computation.image.detection.Peak2DDetectionParam*

Returns a new instance of `Peak2DDetectionParam` with the fields set to the given values.

**Parameters**

- **threshold** (*float*) – Relative threshold. Detection threshold, relative to difference between data maximum and minimum. Float between 0.1 and 0.9. Default: 0.5.
- **size** (*int*) – Neighborhoods size. Size of the sliding window used in maximum/minimum filtering algorithm. Integer higher than 1, unit: pixels. Default: 10.
- **create\_rois** (*bool*) – Default: True.

**Returns**

New instance of `Peak2DDetectionParam`.

**Edge detection parameters**

**class** `cdl.param.CannyParam`

Canny filter parameters

**sigma**

Standard deviation of the gaussian filter. Float higher than 0, non zero, unit: pixels. Default: 1.0.

**Type**`guidata.dataset.dataitems.FloatItem`**low\_threshold**

Lower bound for hysteresis thresholding (linking edges). Float higher than 0. Default: 0.1.

**Type**`guidata.dataset.dataitems.FloatItem`**high\_threshold**

Upper bound for hysteresis thresholding (linking edges). Float higher than 0. Default: 0.9.

**Type**`guidata.dataset.dataitems.FloatItem`**use\_quantiles**

If true then treat `low_threshold` and `high_threshold` as quantiles of the edge magnitude image, rather than absolute edge magnitude values. If true then the thresholds must be in the range [0, 1]. Default: True.

**Type**`guidata.dataset.dataitems.BoolItem`**mode**

Single choice from: 'reflect', 'constant', 'nearest', 'mirror', 'wrap'. Default: 'constant'.

**Type**`guidata.dataset.dataitems.ChoiceItem`**cval**

Value to fill past edges of input if mode is constant. Default: 0.0.



**Type**

guidata.dataset.dataitems.FloatItem

**classmethod create**(*sigma: float, low\_threshold: float, high\_threshold: float, use\_quantiles: bool, mode: str, cval: float*) → *cdl.computation.image.edges.CannyParam*

Returns a new instance of CannyParam with the fields set to the given values.

**Parameters**

- **sigma** (*float*) – Standard deviation of the gaussian filter. Float higher than 0, non zero, unit: pixels. Default: 1.0.
- **low\_threshold** (*float*) – Lower bound for hysteresis thresholding (linking edges). Float higher than 0. Default: 0.1.
- **high\_threshold** (*float*) – Upper bound for hysteresis thresholding (linking edges). Float higher than 0. Default: 0.9.
- **use\_quantiles** (*bool*) – If true then treat low\_threshold and high\_threshold as quantiles of the edge magnitude image, rather than absolute edge magnitude values. If true then the thresholds must be in the range [0, 1]. Default: True.
- **mode** (*str*) – Single choice from: ‘reflect’, ‘constant’, ‘nearest’, ‘mirror’, ‘wrap’. Default: ‘constant’.
- **cval** (*float*) – Value to fill past edges of input if mode is constant. Default: 0.0.

**Returns**

New instance of CannyParam.

**Exposure correction parameters**

**class** *cdl.param.AdjustGammaParam*

Gamma adjustment parameters

**gamma**

Gamma correction factor (higher values give more contrast). Float higher than 0.0. Default: 1.0.

**Type**

guidata.dataset.dataitems.FloatItem

**gain**

Gain factor (higher values give more contrast). Float higher than 0.0. Default: 1.0.

**Type**

guidata.dataset.dataitems.FloatItem

**classmethod create**(*gamma: float, gain: float*) → *cdl.computation.image.exposure.AdjustGammaParam*

Returns a new instance of AdjustGammaParam with the fields set to the given values.

**Parameters**

- **gamma** (*float*) – Gamma correction factor (higher values give more contrast). Float higher than 0.0. Default: 1.0.
- **gain** (*float*) – Gain factor (higher values give more contrast). Float higher than 0.0. Default: 1.0.

**Returns**

New instance of AdjustGammaParam.

**class** `cdl.param.AdjustLogParam`

Logarithmic adjustment parameters

**gain**

Gain factor (higher values give more contrast). Float higher than 0.0. Default: 1.0.

**Type**`guidata.dataset.dataitems.FloatItem`**inv**

If true, apply inverse logarithmic transformation. Default: False.

**Type**`guidata.dataset.dataitems.BoolItem`**classmethod** `create(gain: float, inv: bool) → cdl.computation.image.exposure.AdjustLogParam`Returns a new instance of `AdjustLogParam` with the fields set to the given values.**Parameters**

- **gain** (*float*) – Gain factor (higher values give more contrast). Float higher than 0.0. Default: 1.0.
- **inv** (*bool*) – If true, apply inverse logarithmic transformation. Default: False.

**Returns**New instance of `AdjustLogParam`.**class** `cdl.param.AdjustSigmoidParam`

Sigmoid adjustment parameters

**cutoff**

Cutoff value (higher values give more contrast). Float between 0.0 and 1.0. Default: 0.5.

**Type**`guidata.dataset.dataitems.FloatItem`**gain**

Gain factor (higher values give more contrast). Float higher than 0.0. Default: 10.0.

**Type**`guidata.dataset.dataitems.FloatItem`**inv**

If true, apply inverse sigmoid transformation. Default: False.

**Type**`guidata.dataset.dataitems.BoolItem`**classmethod** `create(cutoff: float, gain: float, inv: bool) → cdl.computation.image.exposure.AdjustSigmoidParam`Returns a new instance of `AdjustSigmoidParam` with the fields set to the given values.**Parameters**

- **cutoff** (*float*) – Cutoff value (higher values give more contrast). Float between 0.0 and 1.0. Default: 0.5.
- **gain** (*float*) – Gain factor (higher values give more contrast). Float higher than 0.0. Default: 10.0.
- **inv** (*bool*) – If true, apply inverse sigmoid transformation. Default: False.

**Returns**

New instance of `AdjustSigmoidParam`.

**class** `cdl.param.EqualizeAdaptHistParam`

Adaptive histogram equalization parameters

**nbins**

Number of bins. Number of bins for image histogram. Integer higher than 1. Default: 256.

**Type**

`guidata.dataset.dataitems.IntItem`

**clip\_limit**

Clipping limit. Clipping limit (higher values give more contrast). Float between 0.0 and 1.0. Default: 0.01.

**Type**

`guidata.dataset.dataitems.FloatItem`

**classmethod** `create`(*nbins*: *int*, *clip\_limit*: *float*) →

`cdl.computation.image.exposure.EqualizeAdaptHistParam`

Returns a new instance of `EqualizeAdaptHistParam` with the fields set to the given values.

**Parameters**

- **nbins** (*int*) – Number of bins. Number of bins for image histogram. Integer higher than 1. Default: 256.
- **clip\_limit** (*float*) – Clipping limit. Clipping limit (higher values give more contrast). Float between 0.0 and 1.0. Default: 0.01.

**Returns**

New instance of `EqualizeAdaptHistParam`.

**class** `cdl.param.EqualizeHistParam`

Histogram equalization parameters

**nbins**

Number of bins. Number of bins for image histogram. Integer higher than 1. Default: 256.

**Type**

`guidata.dataset.dataitems.IntItem`

**classmethod** `create`(*nbins*: *int*) → `cdl.computation.image.exposure.EqualizeHistParam`

Returns a new instance of `EqualizeHistParam` with the fields set to the given values.

**Parameters**

- **nbins** (*int*) – Number of bins. Number of bins for image histogram. Integer higher than 1. Default: 256.

**Returns**

New instance of `EqualizeHistParam`.

**class** `cdl.param.RescaleIntensityParam`

Intensity rescaling parameters

**in\_range**

Input range. Min and max intensity values of input image ('image' refers to input image min/max levels, 'dtype' refers to input image data type range). Single choice from: 'image', 'dtype', 'float32', 'float64', 'complex128', 'int32', 'int16', 'uint16', 'uint8'. Default: 'image'.

**Type**`guidata.dataset.dataitems.ChoiceItem`**out\_range**

Output range. Min and max intensity values of output image ('image' refers to input image min/max levels, 'dtype' refers to input image data type range).. Single choice from: 'image', 'dtype', 'float32', 'float64', 'complex128', 'int32', 'int16', 'uint16', 'uint8'. Default: 'dtype'.

**Type**`guidata.dataset.dataitems.ChoiceItem`

**classmethod** **create**(*in\_range: str*, *out\_range: str*) →

*cdl.computation.image.exposure.RescaleIntensityParam*

Returns a new instance of `RescaleIntensityParam` with the fields set to the given values.

**Parameters**

- **in\_range** (*str*) – Input range. Min and max intensity values of input image ('image' refers to input image min/max levels, 'dtype' refers to input image data type range). Single choice from: 'image', 'dtype', 'float32', 'float64', 'complex128', 'int32', 'int16', 'uint16', 'uint8'. Default: 'image'.
- **out\_range** (*str*) – Output range. Min and max intensity values of output image ('image' refers to input image min/max levels, 'dtype' refers to input image data type range).. Single choice from: 'image', 'dtype', 'float32', 'float64', 'complex128', 'int32', 'int16', 'uint16', 'uint8'. Default: 'dtype'.

**Returns**

New instance of `RescaleIntensityParam`.

## Morphological parameters

**class** `cdl.param.MorphologyParam`

White Top-Hat parameters

**radius**

Footprint (disk) radius. Integer higher than 1. Default: 1.

**Type**`guidata.dataset.dataitems.IntItem`

**classmethod** **create**(*radius: int*) → *cdl.computation.image.morphology.MorphologyParam*

Returns a new instance of `MorphologyParam` with the fields set to the given values.

**Parameters**

**radius** (*int*) – Footprint (disk) radius. Integer higher than 1. Default: 1.

**Returns**

New instance of `MorphologyParam`.

## Restoration parameters

### **class** `cdl.param.DenoiseBilateralParam`

Bilateral filter denoising parameters

#### **sigma\_spatial**

spatial. Standard deviation for range distance. A larger value results in averaging of pixels with larger spatial differences. Float higher than 0, non zero, unit: pixels. Default: 1.0.

#### **Type**

`guidata.dataset.dataitems.FloatItem`

#### **mode**

Single choice from: 'constant', 'edge', 'symmetric', 'reflect', 'wrap'. Default: 'constant'.

#### **Type**

`guidata.dataset.dataitems.ChoiceItem`

#### **cval**

Used in conjunction with mode 'constant', the value outside the image boundaries. Default: 0.

#### **Type**

`guidata.dataset.dataitems.FloatItem`

**classmethod** `create(sigma_spatial: float, mode: str, cval: float) →`

`cdl.computation.image.restoration.DenoiseBilateralParam`

Returns a new instance of `DenoiseBilateralParam` with the fields set to the given values.

#### **Parameters**

- **sigma\_spatial** (*float*) – spatial. Standard deviation for range distance. A larger value results in averaging of pixels with larger spatial differences. Float higher than 0, non zero, unit: pixels. Default: 1.0.
- **mode** (*str*) – Single choice from: 'constant', 'edge', 'symmetric', 'reflect', 'wrap'. Default: 'constant'.
- **cval** (*float*) – Used in conjunction with mode 'constant', the value outside the image boundaries. Default: 0.

#### **Returns**

New instance of `DenoiseBilateralParam`.

### **class** `cdl.param.DenoiseTVParam`

Total Variation denoising parameters

#### **weight**

Denoising weight. The greater weight, the more denoising (at the expense of fidelity to input). Float higher than 0, non zero. Default: 0.1.

#### **Type**

`guidata.dataset.dataitems.FloatItem`

#### **eps**

Epsilon. Relative difference of the value of the cost function that determines the stop criterion. The algorithm stops when:  $(e_{(n-1)} - e_n) < \text{eps} * e_0$ . Float higher than 0, non zero. Default: 0.0002.

#### **Type**

`guidata.dataset.dataitems.FloatItem`

**max\_num\_iter**

Max. iterations. Maximal number of iterations used for the optimization. Integer higher than 0, non zero. Default: 200.

**Type**

[guidata.dataset.dataitems.IntItem](#)

**classmethod create**(*weight: float, eps: float, max\_num\_iter: int*) →  
[cdl.computation.image.restoration.DenoiseTVParam](#)

Returns a new instance of `DenoiseTVParam` with the fields set to the given values.

**Parameters**

- **weight** (*float*) – Denoising weight. The greater weight, the more denoising (at the expense of fidelity to input). Float higher than 0, non zero. Default: 0.1.
- **eps** (*float*) – Epsilon. Relative difference of the value of the cost function that determines the stop criterion. The algorithm stops when:  $(e_{(n-1)} - e_n) < \text{eps} * e_0$ . Float higher than 0, non zero. Default: 0.0002.
- **max\_num\_iter** (*int*) – Max. iterations. Maximal number of iterations used for the optimization. Integer higher than 0, non zero. Default: 200.

**Returns**

New instance of `DenoiseTVParam`.

**class** `cdl.param.DenoiseWaveletParam`

Wavelet denoising parameters

**wavelet**

Single choice from: 'bior1.1', 'bior1.3', 'bior1.5', 'bior2.2', 'bior2.4', 'bior2.6', 'bior2.8', 'bior3.1', 'bior3.3', 'bior3.5', 'bior3.7', 'bior3.9', 'bior4.4', 'bior5.5', 'bior6.8', 'cgau1', 'cgau2', 'cgau3', 'cgau4', 'cgau5', 'cgau6', 'cgau7', 'cgau8', 'cmor', 'coif1', 'coif2', 'coif3', 'coif4', 'coif5', 'coif6', 'coif7', 'coif8', 'coif9', 'coif10', 'coif11', 'coif12', 'coif13', 'coif14', 'coif15', 'coif16', 'coif17', 'db1', 'db2', 'db3', 'db4', 'db5', 'db6', 'db7', 'db8', 'db9', 'db10', 'db11', 'db12', 'db13', 'db14', 'db15', 'db16', 'db17', 'db18', 'db19', 'db20', 'db21', 'db22', 'db23', 'db24', 'db25', 'db26', 'db27', 'db28', 'db29', 'db30', 'db31', 'db32', 'db33', 'db34', 'db35', 'db36', 'db37', 'db38', 'dmey', 'fbsp', 'gaus1', 'gaus2', 'gaus3', 'gaus4', 'gaus5', 'gaus6', 'gaus7', 'gaus8', 'haar', 'mexh', 'morl', 'rbio1.1', 'rbio1.3', 'rbio1.5', 'rbio2.2', 'rbio2.4', 'rbio2.6', 'rbio2.8', 'rbio3.1', 'rbio3.3', 'rbio3.5', 'rbio3.7', 'rbio3.9', 'rbio4.4', 'rbio5.5', 'rbio6.8', 'shan', 'sym2', 'sym3', 'sym4', 'sym5', 'sym6', 'sym7', 'sym8', 'sym9', 'sym10', 'sym11', 'sym12', 'sym13', 'sym14', 'sym15', 'sym16', 'sym17', 'sym18', 'sym19', 'sym20'. Default: 'sym9'.

**Type**

[guidata.dataset.dataitems.ChoiceItem](#)

**mode**

Single choice from: 'soft', 'hard'. Default: 'soft'.

**Type**

[guidata.dataset.dataitems.ChoiceItem](#)

**method**

Single choice from: 'BayesShrink', 'VisuShrink'. Default: 'VisuShrink'.

**Type**

[guidata.dataset.dataitems.ChoiceItem](#)

**classmethod create**(*wavelet: str, mode: str, method: str*) →  
[cdl.computation.image.restoration.DenoiseWaveletParam](#)

Returns a new instance of `DenoiseWaveletParam` with the fields set to the given values.

**Parameters**

- **wavelet** (*str*) – Single choice from: 'bior1.1', 'bior1.3', 'bior1.5', 'bior2.2', 'bior2.4', 'bior2.6', 'bior2.8', 'bior3.1', 'bior3.3', 'bior3.5', 'bior3.7', 'bior3.9', 'bior4.4', 'bior5.5', 'bior6.8', 'cgau1', 'cgau2', 'cgau3', 'cgau4', 'cgau5', 'cgau6', 'cgau7', 'cgau8', 'cmor', 'coif1', 'coif2', 'coif3', 'coif4', 'coif5', 'coif6', 'coif7', 'coif8', 'coif9', 'coif10', 'coif11', 'coif12', 'coif13', 'coif14', 'coif15', 'coif16', 'coif17', 'db1', 'db2', 'db3', 'db4', 'db5', 'db6', 'db7', 'db8', 'db9', 'db10', 'db11', 'db12', 'db13', 'db14', 'db15', 'db16', 'db17', 'db18', 'db19', 'db20', 'db21', 'db22', 'db23', 'db24', 'db25', 'db26', 'db27', 'db28', 'db29', 'db30', 'db31', 'db32', 'db33', 'db34', 'db35', 'db36', 'db37', 'db38', 'dmey', 'fbsp', 'gaus1', 'gaus2', 'gaus3', 'gaus4', 'gaus5', 'gaus6', 'gaus7', 'gaus8', 'haar', 'mexh', 'morl', 'rbio1.1', 'rbio1.3', 'rbio1.5', 'rbio2.2', 'rbio2.4', 'rbio2.6', 'rbio2.8', 'rbio3.1', 'rbio3.3', 'rbio3.5', 'rbio3.7', 'rbio3.9', 'rbio4.4', 'rbio5.5', 'rbio6.8', 'shan', 'sym2', 'sym3', 'sym4', 'sym5', 'sym6', 'sym7', 'sym8', 'sym9', 'sym10', 'sym11', 'sym12', 'sym13', 'sym14', 'sym15', 'sym16', 'sym17', 'sym18', 'sym19', 'sym20'. Default: 'sym9'.
- **mode** (*str*) – Single choice from: 'soft', 'hard'. Default: 'soft'.
- **method** (*str*) – Single choice from: 'BayesShrink', 'VisuShrink'. Default: 'VisuShrink'.

**Returns**

New instance of `DenoiseWaveletParam`.

**Threshold parameters**

**class** `cdl.param.ThresholdParam`

Histogram threshold parameters

**method**

Threshold method. Single choice from: 'manual', 'isodata', 'li', 'mean', 'minimum', 'otsu', 'triangle', 'yen'. Default: 'manual'.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**bins**

Number of bins. Integer higher than 1. Default: 256.

**Type**

`guidata.dataset.dataitems.IntItem`

**value**

Threshold value. Default: 0.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**operation**

Single choice from: '>', '<'. Default: '>'.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**classmethod** `create`(*method: str, bins: int, value: float, operation: str*) → `cdl.computation.image.threshold.ThresholdParam`

Returns a new instance of `ThresholdParam` with the fields set to the given values.

**Parameters**

- **method** (*str*) – Threshold method. Single choice from: ‘manual’, ‘isodata’, ‘li’, ‘mean’, ‘minimum’, ‘otsu’, ‘triangle’, ‘yen’. Default: ‘manual’.
- **bins** (*int*) – Number of bins. Integer higher than 1. Default: 256.
- **value** (*float*) – Threshold value. Default: 0.0.
- **operation** (*str*) – Single choice from: ‘>’, ‘<’. Default: ‘>’.

**Returns**

New instance of ThresholdParam.

### 3.3 Object model (cdl.obj)

The *cdl.obj* module aims at providing all the necessary classes and functions to create and manipulate DataLab signal and image objects.

Those classes and functions are defined in other modules:

- `cdl.core.model.base`
- `cdl.core.model.image`
- `cdl.core.model.signal`
- `cdl.core.io`

The *cdl.obj* module is thus a convenient way to import all the objects at once. As a matter of fact, the following import statement is equivalent to the previous one:

```
# Original import statement
from cdl.core.model.signal import SignalObj
from cdl.core.model.image import ImageObj

# Equivalent import statement
from cdl.obj import SignalObj, ImageObj
```

#### 3.3.1 Common objects

**class** `cdl.obj.ResultProperties` (*title: str, array: ndarray, labels: list[str] | None, item\_json: str = ""*)

Object representing properties serializable in signal/image metadata.

Result *array* is a NumPy 2-D array: each row is a list of properties, optionnally associated to a ROI (first column value).

ROI index is starting at 0 (or is simply 0 if there is no ROI).

**Parameters**

- **title** – properties title
- **array** – properties array
- **labels** – properties labels (one label per column of result array)
- **item\_json** – JSON string of label item associated to this obj

**Note:** The *array* argument can be a list of lists or a NumPy array. For instance, the following are equivalent:



- `array = [[1, 2], [3, 4]]`
- `array = np.array([[1, 2], [3, 4]])`

Or for only one line (one single result), the following are equivalent:

- `array = [1, 2]`
- `array = [[1, 2]]`
- `array = np.array([[1, 2]])`

**property category:** `str`

Return result category

**property headers:** `list[str] | None`

Return result headers (one header per column of result array)

**property shown\_array:** `ndarray`

Return array of shown results, i.e. including complementary array (if any)

**Returns**

Array of shown results

**update\_obj\_metadata\_from\_item**(*obj*: *BaseObj*, *item*: *LabelItem* | *None*) → *None*

Update object metadata with label item

**Parameters**

- **obj** – object (signal/image)
- **item** – label item

**property label\_contents:** `tuple[tuple[int, str], ...]`

Return label contents, i.e. a tuple of couples of (index, text) where index is the column of *raw\_data* and text is the associated label format string

**create\_label\_item**(*obj*: *BaseObj*) → *LabelItem* | *None*

Create label item

**Parameters**

**obj** – object (signal/image)

**Returns**

Label item

**Note:** The signal or image object is required as argument to create the label item because the label text may contain format strings that need to be filled with the object properties. For instance, the label text may contain the signal or image units.

**get\_label\_item**(*obj*: *BaseObj*) → *LabelItem* | *None*

Return label item associated to this result

**Parameters**

**obj** – object (signal/image)

**Returns**

Label item

---

**Note:** The signal or image object is required as argument to eventually create the label item if it has not been created yet. See [create\\_label\\_item\(\)](#).

---

**class** `cdl.obj.ResultShape`(*title*: *str*, *array*: *ndarray*, *shape*: *Literal*['rectangle', 'circle', 'ellipse', 'segment', 'marker', 'point', 'polygon'], *item\_json*: *str* = "", *add\_label*: *bool* = *False*)

Object representing a geometrical shape serializable in signal/image metadata.

Result *array* is a NumPy 2-D array: each row is a result, optionnally associated to a ROI (first column value).

ROI index is starting at 0 (or is simply 0 if there is no ROI).

#### Parameters

- **title** – result shape title
- **array** – shape coordinates (multiple shapes: one shape per row), first column is ROI index (0 if there is no ROI)
- **shape** – shape kind
- **item\_json** – JSON string of label item associated to this obj
- **add\_label** – if True, add a label item (and the geometrical shape) to plot (default to False)

#### Raises

**AssertionError** – invalid argument

---

**Note:** The *array* argument can be a list of lists or a NumPy array. For instance, the following are equivalent:

- `array = [[1, 2], [3, 4]]`
- `array = np.array([[1, 2], [3, 4]])`

Or for only one line (one single result), the following are equivalent:

- `array = [1, 2]`
  - `array = [[1, 2]]`
  - `array = np.array([[1, 2]])`
- 

**property category:** *str*

Return result category

**check\_array()** → *None*

Check if array attribute is valid

#### Raises

**AssertionError** – invalid array

**property headers:** *list[str] | None*

Return result headers (one header per column of result array)

**property shown\_array:** *ndarray*

Return array of shown results, i.e. including complementary array (if any)

#### Returns

Array of shown results

**property label\_contents:** `tuple[tuple[int, str], ...]`

Return label contents, i.e. a tuple of couples of (index, text) where index is the column of raw\_data and text is the associated label format string

**create\_label\_item**(*obj*: *BaseObj*) → *LabelItem* | *None*

Create label item

**Returns**

Label item

**merge\_with**(*obj*: *BaseObj*, *other\_metadata*: *dict[str, Any]*) → *None*

Merge object resultshape with another's metadata (obj ← other obj's metadata)

**Parameters**

- **obj** – object (signal/image)
- **other\_metadata** – other object metadata

**transform\_coordinates**(*func*: *Callable[[ndarray], None]*) → *None*

Transform shape coordinates.

**Parameters**

**func** – function to transform coordinates

**iterate\_plot\_items**(*fmt*: *str*, *lbl*: *bool*, *option*: *Literal['s', 'i']*) → *Iterable*

Iterate over metadata shape plot items.

**Parameters**

- **fmt** – numeric format (e.g. “%.3f”)
- **lbl** – if True, show shape labels
- **option** – shape style option (“s” for signal, “i” for image)

**Yields**

Plot item

**create\_shape\_item**(*coords*: *np.ndarray*, *fmt*: *str*, *lbl*: *bool*, *option*: *Literal['s', 'i']*) → *AnnotatedPoint* | *Marker* | *AnnotatedRectangle* | *AnnotatedCircle* | *AnnotatedSegment* | *AnnotatedEllipse* | *PolygonShape* | *None*

Make geometrical shape plot item adapted to the shape type.

**Parameters**

- **coords** – shape data
- **fmt** – numeric format (e.g. “%.3f”)
- **lbl** – if True, show shape labels
- **option** – shape style option (“s” for signal, “i” for image)

**Returns**

Plot item

**class** `cdl.obj.ShapeTypes`(*value*, *names*=*None*, \*, *module*=*None*, *qualname*=*None*, *type*=*None*, *start*=*1*, *boundary*=*None*)

Shape types for image metadata

**RECTANGLE** = `'_rec_'`

Rectangle shape

```
CIRCLE = '_cir_'
    Circle shape
ELLIPSE = '_ell_'
    Ellipse shape
SEGMENT = '_seg_'
    Segment shape
MARKER = '_mar_'
    Marker shape
POINT = '_poi_'
    Point shape
POLYGON = '_pol_'
    Polygon shape
class cdl.obj.UniformRandomParam(title=None, comment=None, icon="")
    Uniform-law random signal/image parameters
    apply_integer_range(vmin, vmax)
        Do something in case of integer min-max range
class cdl.obj.NormalRandomParam(title=None, comment=None, icon="")
    Normal-law random signal/image parameters
    apply_integer_range(vmin, vmax)
        Do something in case of integer min-max range
class cdl.obj.BaseProcParam(title=None, comment=None, icon="")
    Base class for processing parameters
    apply_integer_range(vmin, vmax)
        Do something in case of integer min-max range
    apply_float_range(vmin, vmax)
        Do something in case of float min-max range
    set_from_datatype(dtype)
        Set min/max range from NumPy datatype
```

### 3.3.2 Signal model

```
class cdl.obj.SignalObj
    Signal object
    uuid
        Default: None.
        Type
            guidata.dataset.dataitems.StringItem
    xydata
        Default: None.
        Type
            guidata.dataset.dataitems.FloatArrayItem
```

**metadata**

Default: {}.

**Type**

`guidata.dataset.dataitems.DictItem`

**title**

Signal title. Default: 'Untitled'.

**Type**

`guidata.dataset.dataitems.StringItem`

**xlabel**

Title. Default: ''.

**Type**

`guidata.dataset.dataitems.StringItem`

**xunit**

Default: ''.

**Type**

`guidata.dataset.dataitems.StringItem`

**ylabel**

Title. Default: ''.

**Type**

`guidata.dataset.dataitems.StringItem`

**yunit**

Default: ''.

**Type**

`guidata.dataset.dataitems.StringItem`

**autoscale**

Default: True.

**Type**

`guidata.dataset.dataitems.BoolItem`

**xscalelog**

Default: False.

**Type**

`guidata.dataset.dataitems.BoolItem`

**xscalemin**

Lower bound. Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`

**xscalemax**

Upper bound. Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`

**yscalelog**

Default: False.

**Type**

`guidata.dataset.dataitems.BoolItem`

**yscalemin**

Lower bound. Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`

**yscalemax**

Upper bound. Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`

**classmethod** **create**(*uuid*: *str*, *xydata*: *numpy.ndarray*, *metadata*: *dict*, *title*: *str*, *xlabel*: *str*, *xunit*: *str*, *ylabel*: *str*, *yunit*: *str*, *autoscale*: *bool*, *yscalelog*: *bool*, *yscalemin*: *float*, *yscalemax*: *float*, *yscalelog*: *bool*, *yscalemin*: *float*, *yscalemax*: *float*) → *cdl.core.model.signal.SignalObj*

Returns a new instance of *SignalObj* with the fields set to the given values.

**Parameters**

- **uuid** (*str*) – Default: None.
- **xydata** (*numpy.ndarray*) – Default: None.
- **metadata** (*dict*) – Default: {}.
- **title** (*str*) – Signal title. Default: 'Untitled'.
- **xlabel** (*str*) – Title. Default: ''.
- **xunit** (*str*) – Default: ''.
- **ylabel** (*str*) – Title. Default: ''.
- **yunit** (*str*) – Default: ''.
- **autoscale** (*bool*) – Default: True.
- **yscalelog** (*bool*) – Default: False.
- **yscalemin** (*float*) – Lower bound. Default: None.
- **yscalemax** (*float*) – Upper bound. Default: None.
- **yscalelog** (*bool*) – Default: False.
- **yscalemin** (*float*) – Lower bound. Default: None.
- **yscalemax** (*float*) – Upper bound. Default: None.

**Returns**

New instance of *SignalObj*.

**static** **get\_roi\_class**() → *Type[SignalROI]*

Return ROI class

**regenerate\_uuid()**

Regenerate UUID

This method is used to regenerate UUID after loading the object from a file. This is required to avoid UUID conflicts when loading objects from file without clearing the workspace first.

**copy**(title: *str* | *None* = *None*, dtype: *dtype* | *None* = *None*) → *SignalObj*

Copy object.

**Parameters**

- **title** – title
- **dtype** – data type

**Returns**

Copied object

**set\_data\_type**(dtype: *dtype*) → *None*

Change data type.

**Parameters**

**type** (*Data*) –

**set\_xydata**(x: *ndarray* | *list*, y: *ndarray* | *list*, dx: *ndarray* | *list* | *None* = *None*, dy: *ndarray* | *list* | *None* = *None*) → *None*

Set xy data

**Parameters**

- **x** – x data
- **y** – y data
- **dx** – dx data (optional: error bars)
- **dy** – dy data (optional: error bars)

**property x:** *ndarray* | *None*

Get x data

**property y:** *ndarray* | *None*

Get y data

**property data:** *ndarray* | *None*

Get y data

**property dx:** *ndarray* | *None*

Get dx data

**property dy:** *ndarray* | *None*

Get dy data

**get\_data**(roi\_index: *int* | *None* = *None*) → *tuple*[*ndarray*, *ndarray*]

Return original data (if ROI is not defined or *roi\_index* is *None*), or ROI data (if both ROI and *roi\_index* are defined).

**Parameters**

**roi\_index** – ROI index

**Returns**

Data

**update\_plot\_item\_parameters**(*item*: *CurveItem*) → *None*

Update plot item parameters from object data/metadata

Takes into account a subset of plot item parameters. Those parameters may have been overridden by object metadata entries or other object data. The goal is to update the plot item accordingly.

This is *almost* the inverse operation of *update\_metadata\_from\_plot\_item*.

**Parameters**

**item** – plot item

**update\_metadata\_from\_plot\_item**(*item*: *CurveItem*) → *None*

Update metadata from plot item.

Takes into account a subset of plot item parameters. Those parameters may have been modified by the user through the plot item GUI. The goal is to update the metadata accordingly.

This is *almost* the inverse operation of *update\_plot\_item\_parameters*.

**Parameters**

**item** – plot item

**make\_item**(*update\_from*: *CurveItem* | *None* = *None*) → *CurveItem*

Make plot item from data.

**Parameters**

**update\_from** – plot item to update from

**Returns**

Plot item

**update\_item**(*item*: *CurveItem*, *data\_changed*: *bool* = *True*) → *None*

Update plot item from data.

**Parameters**

- **item** – plot item
- **data\_changed** – if *True*, data has changed

**physical\_to\_indices**(*coords*: *list[float]* | *ndarray*) → *ndarray*

Convert coordinates from physical (real world) to (array) indices (pixel)

**Parameters**

**coords** – coordinates

**Returns**

Indices

**indices\_to\_physical**(*indices*: *list[int]* | *ndarray*) → *ndarray*

Convert coordinates from (array) indices to physical (real world)

**Parameters**

**indices** – indices

**Returns**

Coordinates

**add\_label\_with\_title**(*title*: *str* | *None* = *None*) → *None*

Add label with title annotation

**Parameters**

**title** – title (if *None*, use signal title)



**accept**(*vis*: *object*) → *None*

Helper function that passes the visitor to the accept methods of all the items in this dataset

**Parameters**

**vis** (*object*) – visitor object

**add\_annotations\_from\_file**(*filename*: *str*) → *None*

Add object annotations from file (JSON).

**Parameters**

**filename** – filename

**add\_annotations\_from\_items**(*items*: *list*) → *None*

Add object annotations (annotation plot items).

**Parameters**

**items** – annotation plot items

**property annotations**: *str*

Get object annotations (JSON string describing annotation plot items)

**check**() → *list*[*str*]

Check the dataset item values

**Returns**

list of errors

**Return type**

*list*[*str*]

**check\_data**()

Check if data is valid, raise an exception if that's not the case

**Raises**

**TypeError** – if data type is not supported

**delete\_results**() → *None*

Delete all object results (shapes and properties)

**deserialize**(*reader*: *HDF5Reader* | *JSONReader* | *INIReader*) → *None*

Deserialize the dataset

**Parameters**

**reader** (*HDF5Reader* | *JSONReader* | *INIReader*) – reader object

**edit**(*parent*: *QWidget* | *None* = *None*, *apply*: *Callable* | *None* = *None*, *wordwrap*: *bool* = *True*, *size*: *QSize* | *tuple*[*int*, *int*] | *None* = *None*) → *DataSetEditDialog*

Open a dialog box to edit data set

**Parameters**

- **parent** – parent widget (default is *None*, meaning no parent)
- **apply** – apply callback (default is *None*)
- **wordwrap** – if *True*, comment text is wordwrapped
- **size** – dialog size (*QSize* object or integer tuple (width, height))

**get\_comment()** → `str` | `None`

Return data set comment

**Returns**

comment

**Return type**

`str` | `None`

**get\_icon()** → `str` | `None`

Return data set icon

**Returns**

icon

**Return type**

`str` | `None`

**get\_items**(*copy=False*) → `list`[`DataItem`]

Returns all the `DataItem` objects from the `DataSet` instance. Ignore private items that have a name starting with an underscore (e.g. `'_private_item = ...'`)

**Parameters**

- **copy** – If `True`, deepcopy the `DataItem` list, else return the original.
- **False.** (*Defaults to*) –

**Returns**

`_description_`

**get\_masked\_view()** → `MaskedArray`

Return masked view for data

**Returns**

Masked view

**get\_metadata\_option**(*name: str*) → `Any`

Return metadata option value

A metadata option is a metadata entry starting with an underscore. It is a way to store application-specific options in object metadata.

**Parameters**

**name** – option name

**Returns**

Option value

**Valid option names:**

`'format'`: format string `'showlabel'`: show label

**get\_title()** → `str`

Return data set title

**Returns**

title

**Return type**

`str`

**classmethod** `get_valid_dtypenames()` → `list[str]`  
 Get valid data type names

**Returns**  
 Valid data type names supported by this class

**invalidate\_maskdata\_cache()** → `None`  
 Invalidate mask data cache: force to rebuild it

**iterate\_resultproperties()** → `Iterable[ResultProperties]`  
 Iterate over object result properties.

**Yields**  
 Result properties

**iterate\_resultshapes()** → `Iterable[ResultShape]`  
 Iterate over object result shapes.

**Yields**  
 Result shape

**iterate\_roi\_indices()** → `Generator[int | None, None, None]`  
 Iterate over object ROI indices (if there is no ROI, yield None)

**iterate\_shape\_items**(*editable: bool = False*)  
 Iterate over shape items encoded in metadata (if any).

**Parameters**  
**editable** – if True, annotations are editable

**Yields**  
 Plot item

**property maskdata:** `ndarray`  
 Return masked data (areas outside defined regions of interest)

**Returns**  
 Masked data

**property number:** `int`  
 Return object number (used for short ID)

**read\_config**(*conf: UserConfig, section: str, option: str*) → `None`  
 Read configuration from a UserConfig instance

**Parameters**

- **conf** (*UserConfig*) – UserConfig instance
- **section** (*str*) – section name
- **option** (*str*) – option name

**remove\_all\_shapes()** → `None`  
 Remove metadata shapes and ROIs

**reset\_metadata\_to\_defaults()** → `None`  
 Reset metadata to default values

**restore\_attr\_from\_metadata**(*attrname: str, default: Any*) → *None*

Restore attribute from metadata

**Parameters**

- **attrname** – attribute name
- **default** – default value

**property roi:** *TypeROI | None*

Return object regions of interest object.

**Returns**

Regions of interest object

**roi\_has\_changed**() → *bool*

Return True if ROI has changed since last call to this method.

The first call to this method will return True if ROI has not yet been set, or if ROI has been set and has changed since the last call to this method. The next call to this method will always return False if ROI has not changed in the meantime.

**Returns**

True if ROI has changed

**save\_attr\_to\_metadata**(*attrname: str, new\_value: Any*) → *None*

Save attribute to metadata

**Parameters**

- **attrname** – attribute name
- **new\_value** – new value

**serialize**(*writer: HDF5Writer | JSONWriter | INIWriter*) → *None*

Serialize the dataset

**Parameters**

**writer** (*HDF5Writer | JSONWriter | INIWriter*) – writer object

**set\_defaults**() → *None*

Set default values

**classmethod set\_global\_prop**(*realm: str, \*\*kwargs*) → *None*

Set global properties for all data items in the dataset

**Parameters**

- **realm** (*str*) – realm name
- **kwargs** (*dict*) – properties to set

**set\_metadata\_option**(*name: str, value: Any*) → *None*

Set metadata option value

A metadata option is a metadata entry starting with an underscore. It is a way to store application-specific options in object metadata.

**Parameters**

- **name** – option name
- **value** – option value

**Valid option names:**

‘format’: format string ‘showlabel’: show label

**property short\_id:** `str`

Short object ID

**text\_edit()** → `None`

Edit data set with text input only

**to\_string**(*debug*: `bool` | `None` = `False`, *indent*: `str` | `None` = `None`, *align*: `bool` | `None` = `False`, *show\_hidden*: `bool` | `None` = `True`) → `str`

Return readable string representation of the data set If debug is True, add more details on data items

**Parameters**

- **debug** (`bool`) – if True, add more details on data items
- **indent** (`str`) – indentation string (default is None, meaning no indentation)
- **align** (`bool`) – if True, align data items (default is False)
- **show\_hidden** (`bool`) – if True, show hidden data items (default is True)

**Returns**

string representation of the data set

**Return type**

`str`

**transform\_shapes**(*orig*, *func*, *param*=`None`)

Apply transform function to result shape / annotations coordinates.

**Parameters**

- **orig** – original object
- **func** – transform function
- **param** – transform function parameter

**update\_metadata\_from**(*other\_metadata*: `dict[str, Any]`) → `None`

Update metadata from another object’s metadata (merge result shapes and annotations, and update the rest of the metadata).

**Parameters**

**other\_metadata** – other object metadata

**update\_metadata\_view\_settings**() → `None`

Update metadata view settings from Conf.view

**update\_resultshapes\_from**(*other*: `TypeObj`) → `None`

Update geometric shape from another object (merge metadata).

**Parameters**

**other** – other object, from which to update this object

**view**(*parent*: `QWidget` | `None` = `None`, *wordwrap*: `bool` = `True`, *size*: `QSize` | `tuple[int, int]` | `None` = `None`) → `None`

Open a dialog box to view data set

**Parameters**

- **parent** – parent widget (default is None, meaning no parent)

- **wordwrap** – if True, comment text is wordwrapped
- **size** – dialog size (QSize object or integer tuple (width, height))

**write\_config**(*conf*: *UserConfig*, *section*: *str*, *option*: *str*) → *None*

Write configuration to a UserConfig instance

**Parameters**

- **conf** (*UserConfig*) – UserConfig instance
- **section** (*str*) – section name
- **option** (*str*) – option name

**cdl.obj.read\_signal**(*filename*: *str*) → *SignalObj*

Read a signal from a file.

**Parameters**

**filename** – File name.

**Returns**

Signal.

**cdl.obj.read\_signals**(*filename*: *str*) → *list[SignalObj]*

Read a list of signals from a file.

**Parameters**

**filename** – File name.

**Returns**

List of signals.

**cdl.obj.create\_signal\_roi**(*coords*: *ndarray* | *list[float, float]* | *list[list[float, float]]*, *indices*: *bool* = *False*, *singleobj*: *bool* | *None* = *None*, *inverse*: *bool* = *False*, *title*: *str* = "") → *SignalROI*

Create Signal Regions of Interest (ROI) object. More ROIs can be added to the object after creation, using the *add\_roi* method.

**Parameters**

- **coords** – single ROI coordinates [*xmin*, *xmax*], or multiple ROIs coordinates [[*xmin1*, *xmax1*], [*xmin2*, *xmax2*], ...] (lists or NumPy arrays)
- **indices** – if True, coordinates are indices, if False, they are physical values (default to False for signals)
- **singleobj** – if True, when extracting data defined by ROIs, only one object is created (default to True). If False, one object is created per single ROI. If None, the value is get from the user configuration
- **inverse** – if True, ROI is outside the region
- **title** – title

**Returns**

Regions of Interest (ROI) object

**Raises**

**ValueError** – if the number of coordinates is not even

**cdl.obj.create\_signal**(*title*: *str*, *x*: *ndarray* | *None* = *None*, *y*: *ndarray* | *None* = *None*, *dx*: *ndarray* | *None* = *None*, *dy*: *ndarray* | *None* = *None*, *metadata*: *dict* | *None* = *None*, *units*: *tuple[str, str]* | *None* = *None*, *labels*: *tuple[str, str]* | *None* = *None*) → *SignalObj*

Create a new Signal object.

**Parameters**

- **title** – signal title
- **x** – X data
- **y** – Y data
- **dx** – dX data (optional: error bars)
- **dy** – dY data (optional: error bars)
- **metadata** – signal metadata
- **units** – X, Y units (tuple of strings)
- **labels** – X, Y labels (tuple of strings)

**Returns**

Signal object

`cdl.obj.create_signal_from_param(newparam: NewSignalParam, addparam: DataSet | None = None, edit: bool = False, parent: QWidget | None = None) → SignalObj | None`

Create a new Signal object from a dialog box.

**Parameters**

- **newparam** – new signal parameters
- **addparam** – additional parameters
- **edit** – Open a dialog box to edit parameters (default: False)
- **parent** – parent widget

**Returns**

Signal object or None if canceled

`cdl.obj.new_signal_param(title: str | None = None, stype: str | None = None, xmin: float | None = None, xmax: float | None = None, size: int | None = None) → NewSignalParam`

Create a new Signal dataset instance.

**Parameters**

- **title** – dataset title (default: None, uses default title)
- **stype** – signal type (default: None, uses default type)
- **xmin** – X min (default: None, uses default value)
- **xmax** – X max (default: None, uses default value)
- **size** – signal size (default: None, uses default value)

**Returns**

new signal dataset instance

**Return type**

*NewSignalParam*

`class cdl.obj.SignalTypes(value, names=None, *, module=None, qualname=None, type=None, start=1, boundary=None)`

Signal types

**ZEROS** = 'zeros'

Signal filled with zeros

**GAUSS = 'gaussian'**

Gaussian function

**LORENTZ = 'lorentzian'**

Lorentzian function

**VOIGT = 'Voigt'**

Voigt function

**UNIFORMRANDOM = 'random (uniform law)'**

Random signal (uniform law)

**NORMALRANDOM = 'random (normal law)'**

Random signal (normal law)

**SINUS = 'sinus'**

Sinusoid

**COSINUS = 'cosinus'**

Cosinusoid

**SAWTOOTH = 'sawtooth'**

Sawtooth function

**TRIANGLE = 'triangle'**

Triangle function

**SQUARE = 'square'**

Square function

**SINC = 'cardinal sine'**

Cardinal sine

**STEP = 'step'**

Step function

**EXPONENTIAL = 'exponential'**

Exponential function

**PULSE = 'pulse'**

Pulse function

**POLYNOMIAL = 'polynomial'**

Polynomial function

**EXPERIMENTAL = 'experimental'**

Experimental function

**class cdl.obj.NewSignalParam**

New signal dataset

**title**

Default: None.

**Type**

`guidata.dataset.dataitems.StringItem`



**xmin**

Default: -10.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**xmax**

Default: 10.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**size**

Signal size (total number of points). Integer higher than 1. Default: 500.

**Type**

`guidata.dataset.dataitems.IntItem`

**stype**

Single choice from: `SignalTypes.ZEROS`, `SignalTypes.GAUSS`, `SignalTypes.LORENTZ`, `SignalTypes.VOIGT`, `SignalTypes.UNIFORMRANDOM`, `SignalTypes.NORMALRANDOM`, `SignalTypes.SINUS`, `SignalTypes.COSINUS`, `SignalTypes.SAWTOOTH`, `SignalTypes.TRIANGLE`, `SignalTypes.SQUARE`, `SignalTypes.SINC`, `SignalTypes.STEP`, `SignalTypes.EXPONENTIAL`, `SignalTypes.PULSE`, `SignalTypes.POLYNOMIAL`, `SignalTypes.EXPERIMENTAL`. Default: `SignalTypes.ZEROS`.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**classmethod** `create`(*title*: *str*, *xmin*: *float*, *xmax*: *float*, *size*: *int*, *stype*: `cdl.core.model.signal.SignalTypes`)  
→ `cdl.core.model.signal.NewSignalParam`

Returns a new instance of `NewSignalParam` with the fields set to the given values.

**Parameters**

- **title** (*str*) – Default: None.
- **xmin** (*float*) – Default: -10.0.
- **xmax** (*float*) – Default: 10.0.
- **size** (*int*) – Signal size (total number of points). Integer higher than 1. Default: 500.
- **stype** (`cdl.core.model.signal.SignalTypes`) – Single choice from: `SignalTypes.ZEROS`, `SignalTypes.GAUSS`, `SignalTypes.LORENTZ`, `SignalTypes.VOIGT`, `SignalTypes.UNIFORMRANDOM`, `SignalTypes.NORMALRANDOM`, `SignalTypes.SINUS`, `SignalTypes.COSINUS`, `SignalTypes.SAWTOOTH`, `SignalTypes.TRIANGLE`, `SignalTypes.SQUARE`, `SignalTypes.SINC`, `SignalTypes.STEP`, `SignalTypes.EXPONENTIAL`, `SignalTypes.PULSE`, `SignalTypes.POLYNOMIAL`, `SignalTypes.EXPERIMENTAL`. Default: `SignalTypes.ZEROS`.

**Returns**

New instance of `NewSignalParam`.

**class** `cdl.obj.GaussLorentzVoigtParam`

Parameters for Gaussian and Lorentzian functions

**a**

Default: 1.0.

**Type**`guidata.dataset.dataitems.FloatItem`**ymin**

Default: 0.0.

**Type**`guidata.dataset.dataitems.FloatItem`**sigma**

. Default: 1.0.

**Type**`guidata.dataset.dataitems.FloatItem`**mu**

. Default: 0.0.

**Type**`guidata.dataset.dataitems.FloatItem`

**classmethod** `create(a: float, ymin: float, sigma: float, mu: float) →`  
`cdl.core.model.signal.GaussLorentzVoigtParam`

Returns a new instance of `GaussLorentzVoigtParam` with the fields set to the given values.

**Parameters**

- **a** (`float`) – Default: 1.0.
- **ymin** (`float`) – Default: 0.0.
- **sigma** (`float`) – . Default: 1.0.
- **mu** (`float`) – . Default: 0.0.

**Returns**

New instance of `GaussLorentzVoigtParam`.

**class** `cdl.obj.StepParam`

Parameters for step function

**a1**

Default: 0.0.

**Type**`guidata.dataset.dataitems.FloatItem`**a2**

Default: 1.0.

**Type**`guidata.dataset.dataitems.FloatItem`**x0**

Default: 0.0.

**Type**`guidata.dataset.dataitems.FloatItem`

**classmethod** `create(a1: float, a2: float, x0: float) → cdl.core.model.signal.StepParam`

Returns a new instance of `StepParam` with the fields set to the given values.

**Parameters**

- **a1** (*float*) – Default: 0.0.
- **a2** (*float*) – Default: 1.0.
- **x0** (*float*) – Default: 0.0.

**Returns**

New instance of *StepParam*.

**class** `cdl.obj.PeriodicParam`

Parameters for periodic functions

**a**

Default: 1.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**ymin**

Default: 0.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**freq**

Frequency. Default: 1.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**freq\_unit**

Single choice from: `FreqUnits.HZ`, `FreqUnits.KHZ`, `FreqUnits.MHZ`, `FreqUnits.GHZ`. Default: `FreqUnits.HZ`.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**phase**

Float, unit: °. Default: 0.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**classmethod** `create(a: float, ymin: float, freq: float, freq_unit: cdl.core.model.signal.FreqUnits, phase: float) → cdl.core.model.signal.PeriodicParam`

Returns a new instance of *PeriodicParam* with the fields set to the given values.

**Parameters**

- **a** (*float*) – Default: 1.0.
- **ymin** (*float*) – Default: 0.0.
- **freq** (*float*) – Frequency. Default: 1.0.
- **freq\_unit** (`cdl.core.model.signal.FreqUnits`) – Single choice from: `FreqUnits.HZ`, `FreqUnits.KHZ`, `FreqUnits.MHZ`, `FreqUnits.GHZ`. Default: `FreqUnits.HZ`.
- **phase** (*float*) – Float, unit: °. Default: 0.0.

**Returns**

New instance of *PeriodicParam*.

**get\_frequency\_in\_hz()**

Return frequency in Hz

**class** `cdl.obj.ROI1DParam`

Signal ROI parameters

**xmin**

First point coordinate. Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`

**xmax**

Last point coordinate. Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`

**classmethod** `create(xmin: float, xmax: float) → cdl.core.model.signal.ROI1DParam`

Returns a new instance of `ROI1DParam` with the fields set to the given values.

**Parameters**

- **xmin** (*float*) – First point coordinate. Default: None.
- **xmax** (*float*) – Last point coordinate. Default: None.

**Returns**

New instance of `ROI1DParam`.

**to\_single\_roi(obj: SignalObj, title: str = "") → SegmentROI**

Convert parameters to single ROI

**Parameters**

- **obj** – signal object
- **title** – ROI title

**Returns**

Single ROI

**get\_data(obj: SignalObj) → ndarray**

Get signal data in ROI

**Parameters**

**obj** – signal object

**Returns**

Data in ROI

**class** `cdl.obj.SignalROI(singleobj: bool | None = None, inverse: bool = False)`

Signal Regions of Interest

**Parameters**

- **singleobj** – if True, when extracting data defined by ROIs, only one object is created (default to True). If False, one object is created per single ROI. If None, the value is get from the user configuration
- **inverse** – if True, ROI is outside the region

**static get\_compatible\_single\_roi\_classes()** → list[Type[SegmentROI]]

Return compatible single ROI classes

**to\_mask(obj: SignalObj)** → ndarray[bool]

Create mask from ROI

**Parameters**

**obj** – signal object

**Returns**

Mask (boolean array where True values are inside the ROI)

### 3.3.3 Image model

**class cdl.obj.ImageObj**

Image object

**uuid**

Default: None.

**Type**

guidata.dataset.dataitems.StringItem

**data**

Default: None.

**Type**

guidata.dataset.dataitems.FloatArrayItem

**metadata**

Default: {}.

**Type**

guidata.dataset.dataitems.DictItem

**x0**

X<sub>0</sub>. Default: 0.0.

**Type**

guidata.dataset.dataitems.FloatItem

**y0**

Y<sub>0</sub>. Default: 0.0.

**Type**

guidata.dataset.dataitems.FloatItem

**dx**

x. Float, non zero. Default: 1.0.

**Type**

guidata.dataset.dataitems.FloatItem

**dy**

y. Float, non zero. Default: 1.0.

**Type**

guidata.dataset.dataitems.FloatItem

**title**

Image title. Default: 'Untitled'.

**Type**

`guidata.dataset.dataitems.StringItem`

**xlabel**

Title. Default: ''.

**Type**

`guidata.dataset.dataitems.StringItem`

**xunit**

Default: ''.

**Type**

`guidata.dataset.dataitems.StringItem`

**ylabel**

Title. Default: ''.

**Type**

`guidata.dataset.dataitems.StringItem`

**yunit**

Default: ''.

**Type**

`guidata.dataset.dataitems.StringItem`

**zlabel**

Title. Default: ''.

**Type**

`guidata.dataset.dataitems.StringItem`

**zunit**

Default: ''.

**Type**

`guidata.dataset.dataitems.StringItem`

**autoscale**

Default: True.

**Type**

`guidata.dataset.dataitems.BoolItem`

**xscalelog**

Default: False.

**Type**

`guidata.dataset.dataitems.BoolItem`

**xscalemin**

Lower bound. Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`

**xscalemax**

Upper bound. Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`

**yscalelog**

Default: False.

**Type**

`guidata.dataset.dataitems.BoolItem`

**yscalemin**

Lower bound. Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`

**yscalemax**

Upper bound. Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`

**zscalemin**

Lower bound. Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`

**zscalemax**

Upper bound. Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`

**classmethod create**(*uuid: str, data: numpy.ndarray, metadata: dict, x0: float, y0: float, dx: float, dy: float, title: str, xlabel: str, xunit: str, ylabel: str, yunit: str, zlabel: str, zunit: str, autoscale: bool, xscalelog: bool, xscalemin: float, xscalemax: float, yscalelog: bool, yscalemin: float, yscalemax: float, zscalemin: float, zscalemax: float*) → *cdl.core.model.image.ImageObj*

Returns a new instance of *ImageObj* with the fields set to the given values.

**Parameters**

- **uuid** (*str*) – Default: None.
- **data** (*numpy.ndarray*) – Default: None.
- **metadata** (*dict*) – Default: {}.
- **x0** (*float*) –  $X_0$ . Default: 0.0.
- **y0** (*float*) –  $Y_0$ . Default: 0.0.
- **dx** (*float*) – x. Float, non zero. Default: 1.0.
- **dy** (*float*) – y. Float, non zero. Default: 1.0.
- **title** (*str*) – Image title. Default: 'Untitled'.
- **xlabel** (*str*) – Title. Default: ''.
- **xunit** (*str*) – Default: ''.

- **ylabel** (*str*) – Title. Default: “.
- **yunit** (*str*) – Default: “.
- **zlabel** (*str*) – Title. Default: “.
- **zunit** (*str*) – Default: “.
- **autoscale** (*bool*) – Default: True.
- **xscalelog** (*bool*) – Default: False.
- **xscalemmin** (*float*) – Lower bound. Default: None.
- **xscalemmax** (*float*) – Upper bound. Default: None.
- **yscalelog** (*bool*) – Default: False.
- **yscalemmin** (*float*) – Lower bound. Default: None.
- **yscalemmax** (*float*) – Upper bound. Default: None.
- **zscalemmin** (*float*) – Lower bound. Default: None.
- **zscalemmax** (*float*) – Upper bound. Default: None.

**Returns**

New instance of *ImageObj*.

**static** **get\_roi\_class**() → *Type[ImageROI]*

Return ROI class

**regenerate\_uuid**()

Regenerate UUID

This method is used to regenerate UUID after loading the object from a file. This is required to avoid UUID conflicts when loading objects from file without clearing the workspace first.

**set\_metadata\_from**(*obj: Mapping | dict*) → *None*

Set metadata from object: dict-like (only string keys are considered) or any other object (iterating over supported attributes)

**Parameters**

**obj** – object

**property** **dicom\_template**

Get DICOM template

**property** **width**: *float*

Return image width, i.e. number of columns multiplied by pixel size

**property** **height**: *float*

Return image height, i.e. number of rows multiplied by pixel size

**property** **xc**: *float*

Return image center X-axis coordinate

**property** **yc**: *float*

Return image center Y-axis coordinate

**get\_data**(*roi\_index: int | None = None*) → *ndarray*

Return original data (if ROI is not defined or *roi\_index* is None), or ROI data (if both ROI and *roi\_index* are defined).



**Parameters****roi\_index** – ROI index**Returns**

Masked data

**copy**(*title: str | None = None, dtype: dtype | None = None*) → *ImageObj*

Copy object.

**Parameters**

- **title** – title
- **dtype** – data type

**Returns**

Copied object

**set\_data\_type**(*dtype: dtype*) → *None*

Change data type. If data type is integer, clip values to the new data type's range, thus avoiding overflow or underflow.

**Parameters****type** (*Data*) –**update\_plot\_item\_parameters**(*item: MaskedImageItem*) → *None*

Update plot item parameters from object data/metadata

Takes into account a subset of plot item parameters. Those parameters may have been overridden by object metadata entries or other object data. The goal is to update the plot item accordingly.

This is *almost* the inverse operation of *update\_metadata\_from\_plot\_item*.

**Parameters****item** – plot item**update\_metadata\_from\_plot\_item**(*item: MaskedImageItem*) → *None*

Update metadata from plot item.

Takes into account a subset of plot item parameters. Those parameters may have been modified by the user through the plot item GUI. The goal is to update the metadata accordingly.

This is *almost* the inverse operation of *update\_plot\_item\_parameters*.

**Parameters****item** – plot item**make\_item**(*update\_from: MaskedImageItem | None = None*) → *MaskedImageItem*

Make plot item from data.

**Parameters****update\_from** – update from plot item**Returns**

Plot item

**update\_item**(*item: MaskedImageItem, data\_changed: bool = True*) → *None*

Update plot item from data.

**Parameters**

- **item** – plot item
- **data\_changed** – if True, data has changed

**physical\_to\_indices**(*coords*: *list[float]*, *clip*: *bool = False*) → *ndarray*

Convert coordinates from physical (real world) to (array) indices (pixel)

**Parameters**

- **coords** – coordinates
- **clip** – if True, clip values to image boundaries

**Returns**

Indices

**indices\_to\_physical**(*indices*: *list[float | int] | ndarray*) → *ndarray*

Convert coordinates from (array) indices to physical (real world)

**Parameters**

**indices** – indices

**Returns**

Coordinates

**add\_label\_with\_title**(*title*: *str | None = None*) → *None*

Add label with title annotation

**Parameters**

**title** – title (if None, use image title)

**accept**(*vis*: *object*) → *None*

Helper function that passes the visitor to the accept methods of all the items in this dataset

**Parameters**

**vis** (*object*) – visitor object

**add\_annotations\_from\_file**(*filename*: *str*) → *None*

Add object annotations from file (JSON).

**Parameters**

**filename** – filename

**add\_annotations\_from\_items**(*items*: *list*) → *None*

Add object annotations (annotation plot items).

**Parameters**

**items** – annotation plot items

**property annotations:** *str*

Get object annotations (JSON string describing annotation plot items)

**check**() → *list[str]*

Check the dataset item values

**Returns**

list of errors

**Return type**

*list[str]*

**check\_data**()

Check if data is valid, raise an exception if that's not the case

**Raises**

**TypeError** – if data type is not supported

**delete\_results()** → `None`

Delete all object results (shapes and properties)

**deserialize**(*reader: HDF5Reader | JSONReader | INIReader*) → `None`

Deserialize the dataset

**Parameters**

**reader** (*HDF5Reader | JSONReader | INIReader*) – reader object

**edit**(*parent: QWidget | None = None, apply: Callable | None = None, wordwrap: bool = True, size: QSize | tuple[int, int] | None = None*) → `DataSetEditDialog`

Open a dialog box to edit data set

**Parameters**

- **parent** – parent widget (default is `None`, meaning no parent)
- **apply** – apply callback (default is `None`)
- **wordwrap** – if `True`, comment text is wordwrapped
- **size** – dialog size (`QSize` object or integer tuple (width, height))

**get\_comment()** → `str | None`

Return data set comment

**Returns**

comment

**Return type**

`str | None`

**get\_icon()** → `str | None`

Return data set icon

**Returns**

icon

**Return type**

`str | None`

**get\_items**(*copy=False*) → `list[DataItem]`

Returns all the `DataItem` objects from the `DataSet` instance. Ignore private items that have a name starting with an underscore (e.g. `'_private_item = ...'`)

**Parameters**

- **copy** – If `True`, deepcopy the `DataItem` list, else return the original.
- **False.** (*Defaults to*) –

**Returns**

`_description_`

**get\_masked\_view()** → `MaskedArray`

Return masked view for data

**Returns**

Masked view

**get\_metadata\_option**(*name: str*) → *Any*

Return metadata option value

A metadata option is a metadata entry starting with an underscore. It is a way to store application-specific options in object metadata.

**Parameters**

**name** – option name

**Returns**

Option value

**Valid option names:**

‘format’: format string ‘showlabel’: show label

**get\_title**() → *str*

Return data set title

**Returns**

title

**Return type**

*str*

**classmethod get\_valid\_dtypenames**() → *list[str]*

Get valid data type names

**Returns**

Valid data type names supported by this class

**invalidate\_maskdata\_cache**() → *None*

Invalidate mask data cache: force to rebuild it

**iterate\_resultproperties**() → *Iterable[ResultProperties]*

Iterate over object result properties.

**Yields**

Result properties

**iterate\_resultshapes**() → *Iterable[ResultShape]*

Iterate over object result shapes.

**Yields**

Result shape

**iterate\_roi\_indices**() → *Generator[int | None, None, None]*

Iterate over object ROI indices (if there is no ROI, yield None)

**iterate\_shape\_items**(*editable: bool = False*)

Iterate over shape items encoded in metadata (if any).

**Parameters**

**editable** – if True, annotations are editable

**Yields**

Plot item

**property maskdata:** *ndarray*

Return masked data (areas outside defined regions of interest)

**Returns**

Masked data

**property number:** `int`

Return object number (used for short ID)

**read\_config**(*conf*: `UserConfig`, *section*: `str`, *option*: `str`) → `None`

Read configuration from a UserConfig instance

**Parameters**

- **conf** (`UserConfig`) – UserConfig instance
- **section** (`str`) – section name
- **option** (`str`) – option name

**remove\_all\_shapes**() → `None`

Remove metadata shapes and ROIs

**reset\_metadata\_to\_defaults**() → `None`

Reset metadata to default values

**restore\_attr\_from\_metadata**(*attrname*: `str`, *default*: `Any`) → `None`

Restore attribute from metadata

**Parameters**

- **attrname** – attribute name
- **default** – default value

**property roi:** `TypeROI` | `None`

Return object regions of interest object.

**Returns**

Regions of interest object

**roi\_has\_changed**() → `bool`

Return True if ROI has changed since last call to this method.

The first call to this method will return True if ROI has not yet been set, or if ROI has been set and has changed since the last call to this method. The next call to this method will always return False if ROI has not changed in the meantime.

**Returns**

True if ROI has changed

**save\_attr\_to\_metadata**(*attrname*: `str`, *new\_value*: `Any`) → `None`

Save attribute to metadata

**Parameters**

- **attrname** – attribute name
- **new\_value** – new value

**serialize**(*writer*: `HDF5Writer` | `JSONWriter` | `INIWriter`) → `None`

Serialize the dataset

**Parameters****writer** (`HDF5Writer` | `JSONWriter` | `INIWriter`) – writer object

**set\_defaults()** → *None*

Set default values

**classmethod set\_global\_prop**(*realm: str, \*\*kwargs*) → *None*

Set global properties for all data items in the dataset

**Parameters**

- **realm** (*str*) – realm name
- **kwargs** (*dict*) – properties to set

**set\_metadata\_option**(*name: str, value: Any*) → *None*

Set metadata option value

A metadata option is a metadata entry starting with an underscore. It is a way to store application-specific options in object metadata.

**Parameters**

- **name** – option name
- **value** – option value

**Valid option names:**

‘format’: format string ‘showlabel’: show label

**property short\_id:** *str*

Short object ID

**text\_edit()** → *None*

Edit data set with text input only

**to\_string**(*debug: bool | None = False, indent: str | None = None, align: bool | None = False, show\_hidden: bool | None = True*) → *str*

Return readable string representation of the data set If debug is True, add more details on data items

**Parameters**

- **debug** (*bool*) – if True, add more details on data items
- **indent** (*str*) – indentation string (default is None, meaning no indentation)
- **align** (*bool*) – if True, align data items (default is False)
- **show\_hidden** (*bool*) – if True, show hidden data items (default is True)

**Returns**

string representation of the data set

**Return type**

*str*

**transform\_shapes**(*orig, func, param=None*)

Apply transform function to result shape / annotations coordinates.

**Parameters**

- **orig** – original object
- **func** – transform function
- **param** – transform function parameter

**update\_metadata\_from**(*other\_metadata*: *dict[str, Any]*) → *None*

Update metadata from another object's metadata (merge result shapes and annotations, and update the rest of the metadata).

**Parameters**

**other\_metadata** – other object metadata

**update\_metadata\_view\_settings**() → *None*

Update metadata view settings from *Conf.view*

**update\_resultshapes\_from**(*other*: *TypeObj*) → *None*

Update geometric shape from another object (merge metadata).

**Parameters**

**other** – other object, from which to update this object

**view**(*parent*: *QWidget* | *None* = *None*, *wordwrap*: *bool* = *True*, *size*: *QSize* | *tuple[int, int]* | *None* = *None*) → *None*

Open a dialog box to view data set

**Parameters**

- **parent** – parent widget (default is *None*, meaning no parent)
- **wordwrap** – if *True*, comment text is wordwrapped
- **size** – dialog size (*QSize* object or integer tuple (width, height))

**write\_config**(*conf*: *UserConfig*, *section*: *str*, *option*: *str*) → *None*

Write configuration to a *UserConfig* instance

**Parameters**

- **conf** (*UserConfig*) – *UserConfig* instance
- **section** (*str*) – section name
- **option** (*str*) – option name

**cdl.obj.read\_image**(*filename*: *str*) → *ImageObj*

Read an image from a file.

**Parameters**

**filename** – File name.

**Returns**

Image.

**cdl.obj.read\_images**(*filename*: *str*) → *list[ImageObj]*

Read a list of images from a file.

**Parameters**

**filename** – File name.

**Returns**

List of images.

**cdl.obj.create\_image\_roi**(*geometry*: *Literal*['rectangle', 'circle', 'polygon'], *coords*: *ndarray* | *list[float]* | *list[list[float]]*, *indices*: *bool* = *True*, *singleobj*: *bool* | *None* = *None*, *inverse*: *bool* = *False*, *title*: *str* = '') → *ImageROI*

Create Image Regions of Interest (ROI) object. More ROIs can be added to the object after creation, using the *add\_roi* method.

**Parameters**

- **geometry** – ROI type ('rectangle', 'circle', 'polygon')
- **coords** – ROI coords (physical coordinates),  $[x0, y0, dx, dy]$  for a rectangle,  $[xc, yc, r]$  for a circle, or  $[x0, y0, x1, y1, \dots]$  for a polygon (lists or NumPy arrays are accepted). For multiple ROIs, nested lists or NumPy arrays are accepted but with a common geometry type (e.g.  $[[xc1, yc1, r1], [xc2, yc2, r2], \dots]$  for circles).
- **indices** – if True, coordinates are indices, if False, they are physical values (default to True for images)
- **singleobj** – if True, when extracting data defined by ROIs, only one object is created (default to True). If False, one object is created per single ROI. If None, the value is get from the user configuration
- **inverse** – if True, ROI is outside the region
- **title** – title

**Returns**

Regions of Interest (ROI) object

**Raises**

**ValueError** – if ROI type is unknown or if the number of coordinates is invalid

`cdl.obj.create_image(title: str, data: ndarray | None = None, metadata: dict | None = None, units: tuple | None = None, labels: tuple | None = None) → ImageObj`

Create a new Image object

**Parameters**

- **title** – image title
- **data** – image data
- **metadata** – image metadata
- **units** – X, Y, Z units (tuple of strings)
- **labels** – X, Y, Z labels (tuple of strings)

**Returns**

Image object

`cdl.obj.create_image_from_param(newparam: NewImageParam, addparam: gds.DataSet | None = None, edit: bool = False, parent: QW.QWidget | None = None) → ImageObj | None`

Create a new Image object from dialog box.

**Parameters**

- **newparam** – new image parameters
- **addparam** – additional parameters
- **edit** – Open a dialog box to edit parameters (default: False)
- **parent** – parent widget

**Returns**

New image object or None if user cancelled

`cdl.obj.new_image_param(title: str | None = None, itype: ImageTypes | None = None, height: int | None = None, width: int | None = None, dtype: ImageDatatypes | None = None) → NewImageParam`



Create a new Image dataset instance.

#### Parameters

- **title** – dataset title (default: None, uses default title)
- **itype** – image type (default: None, uses default type)
- **height** – image height (default: None, uses default height)
- **width** – image width (default: None, uses default width)
- **dtype** – image data type (default: None, uses default data type)

#### Returns

New image dataset instance

```
class cdl.obj.ImageTypes(value, names=None, *, module=None, qualname=None, type=None, start=1,
                          boundary=None)
```

Image types

**ZEROS** = 'zeros'

Image filled with zeros

**EMPTY** = 'empty'

Empty image (filled with data from memory state)

**GAUSS** = 'gaussian'

2D Gaussian image

**UNIFORMRANDOM** = 'random (uniform law)'

Image filled with random data (uniform law)

**NORMALRANDOM** = 'random (normal law)'

Image filled with random data (normal law)

```
class cdl.obj.NewImageParam
```

New image dataset

**title**

Default: None.

**Type**

`guidata.dataset.dataitems.StringItem`

**height**

Image height: number of rows. Integer higher than 1. Default: None.

**Type**

`guidata.dataset.dataitems.IntItem`

**width**

Image width: number of columns. Integer higher than 1. Default: None.

**Type**

`guidata.dataset.dataitems.IntItem`

**dtype**

Data type. Single choice from: ImageDatatypes.UINT8, ImageDatatypes.UINT16, ImageDatatypes.INT16, ImageDatatypes.FLOAT32, ImageDatatypes.FLOAT64. Default: ImageDatatypes.UINT8.

**Type**`guidata.dataset.dataitems.ChoiceItem`**itype**

Single choice from: `ImageTypes.ZEROS`, `ImageTypes.EMPTY`, `ImageTypes.GAUSS`, `ImageTypes.UNIFORMRANDOM`, `ImageTypes.NORMALRANDOM`. Default: `ImageTypes.ZEROS`.

**Type**`guidata.dataset.dataitems.ChoiceItem`

**classmethod** `create`(*title: str, height: int, width: int, dtype: cdl.core.model.image.ImageDatatypes, itype: cdl.core.model.image.ImageTypes*) → *cdl.core.model.image.NewImageParam*

Returns a new instance of *NewImageParam* with the fields set to the given values.

**Parameters**

- **title** (*str*) – Default: None.
- **height** (*int*) – Image height: number of rows. Integer higher than 1. Default: None.
- **width** (*int*) – Image width: number of columns. Integer higher than 1. Default: None.
- **dtype** (`cdl.core.model.image.ImageDatatypes`) – Data type. Single choice from: `ImageDatatypes.UINT8`, `ImageDatatypes.UINT16`, `ImageDatatypes.INT16`, `ImageDatatypes.FLOAT32`, `ImageDatatypes.FLOAT64`. Default: `ImageDatatypes.UINT8`.
- **itype** (`cdl.core.model.image.ImageTypes`) – Single choice from: `ImageTypes.ZEROS`, `ImageTypes.EMPTY`, `ImageTypes.GAUSS`, `ImageTypes.UNIFORMRANDOM`, `ImageTypes.NORMALRANDOM`. Default: `ImageTypes.ZEROS`.

**Returns**

New instance of *NewImageParam*.

**class** `cdl.obj.Gauss2DParam`

2D Gaussian parameters

**a**

Norm. Default: None.

**Type**`guidata.dataset.dataitems.FloatItem`**xmin**

Default: -10.

**Type**`guidata.dataset.dataitems.FloatItem`**sigma**

. Default: 1.0.

**Type**`guidata.dataset.dataitems.FloatItem`**xmax**

Default: 10.

**Type**`guidata.dataset.dataitems.FloatItem`

**mu**

. Default: 0.0.

**Type**`guidata.dataset.dataitems.FloatItem`**ymin**

Default: -10.

**Type**`guidata.dataset.dataitems.FloatItem`**x0**

Default: 0.

**Type**`guidata.dataset.dataitems.FloatItem`**ymax**

Default: 10.

**Type**`guidata.dataset.dataitems.FloatItem`**y0**

Default: 0.

**Type**`guidata.dataset.dataitems.FloatItem`

**classmethod create**(*a: float, xmin: float, sigma: float, xmax: float, mu: float, ymin: float, x0: float, ymax: float, y0: float*) → `cdl.core.model.image.Gauss2DParam`

Returns a new instance of `Gauss2DParam` with the fields set to the given values.

**Parameters**

- **a** (*float*) – Norm. Default: None.
- **xmin** (*float*) – Default: -10.
- **sigma** (*float*) – . Default: 1.0.
- **xmax** (*float*) – Default: 10.
- **mu** (*float*) – . Default: 0.0.
- **ymin** (*float*) – Default: -10.
- **x0** (*float*) – Default: 0.
- **ymax** (*float*) – Default: 10.
- **y0** (*float*) – Default: 0.

**Returns**

New instance of `Gauss2DParam`.

**class** `cdl.obj.ROI2DParam`

Image ROI parameters

**geometry**

Single choice from: 'rectangle', 'circle', 'polygon'. Default: 'rectangle'.

**Type**`guidata.dataset.dataitems.ChoiceItem`**x0**

$X_0$ . Integer, unit: pixels. Default: None.

**Type**`guidata.dataset.dataitems.IntItem`**y0**

$Y_0$ . Integer, unit: pixels. Default: None.

**Type**`guidata.dataset.dataitems.IntItem`**dx**

$X$ . Integer, unit: pixels. Default: None.

**Type**`guidata.dataset.dataitems.IntItem`**dy**

$Y$ . Integer, unit: pixels. Default: None.

**Type**`guidata.dataset.dataitems.IntItem`**xc**

$X_C$ . Integer, unit: pixels. Default: None.

**Type**`guidata.dataset.dataitems.IntItem`**yc**

$Y_C$ . Integer, unit: pixels. Default: None.

**Type**`guidata.dataset.dataitems.IntItem`**r**

Radius. Integer, unit: pixels. Default: None.

**Type**`guidata.dataset.dataitems.IntItem`**points**

Coordinates (pixels). Default: None.

**Type**`guidata.dataset.dataitems.FloatArrayItem`

**classmethod create**(*geometry: str*, *x0: int*, *y0: int*, *dx: int*, *dy: int*, *xc: int*, *yc: int*, *r: int*, *points: numpy.ndarray*) → `cdl.core.model.image.ROI2DParam`

Returns a new instance of `ROI2DParam` with the fields set to the given values.

**Parameters**

- **geometry** (*str*) – Single choice from: ‘rectangle’, ‘circle’, ‘polygon’. Default: ‘rectangle’.
- **x0** (*int*) –  $X_0$ . Integer, unit: pixels. Default: None.
- **y0** (*int*) –  $Y_0$ . Integer, unit: pixels. Default: None.

- **dx** (*int*) – X. Integer, unit: pixels. Default: None.
- **dy** (*int*) – Y. Integer, unit: pixels. Default: None.
- **xc** (*int*) –  $X_C$ . Integer, unit: pixels. Default: None.
- **yc** (*int*) –  $Y_C$ . Integer, unit: pixels. Default: None.
- **r** (*int*) – Radius. Integer, unit: pixels. Default: None.
- **points** (*numpy.ndarray*) – Coordinates (pixels). Default: None.

**Returns**

New instance of *ROI2DParam*.

**to\_single\_roi** (*obj*: *ImageObj*, *title*: *str* = "") → *PolygonalROI* | *RectangularROI* | *CircularROI*

Convert parameters to single ROI

**Parameters**

- **obj** – image object (used for conversion of pixel to physical coordinates)
- **title** – ROI title

**Returns**

Single ROI

**get\_suffix**() → *str*

Get suffix text representation for ROI extraction

**get\_extracted\_roi** (*obj*: *ImageObj*) → *ImageROI* | *None*

Get extracted ROI, i.e. the remaining ROI after extracting ROI from image.

**Parameters**

**obj** – image object (used for conversion of pixel to physical coordinates)

When extracting ROIs from an image to multiple images (i.e. one image per ROI), this method returns the ROI that has to be kept in the destination image. This is not necessary for a rectangular ROI: the destination image is simply a crop of the source image according to the ROI coordinates. But for a circular ROI or a polygonal ROI, the destination image is a crop of the source image according to the bounding box of the ROI. Thus, to avoid any loss of information, a ROI has to be defined for the destination image: this is the ROI returned by this method. It's simply the same as the source ROI, but with coordinates adjusted to the destination image. One may call this ROI the “extracted ROI”.

**get\_bounding\_box\_indices**() → *tuple*[*int*, *int*, *int*, *int*]

Get bounding box (pixel coordinates)

**get\_data** (*obj*: *ImageObj*) → *ndarray*

Get data in ROI

**Parameters**

**obj** – image object

**Returns**

Data in ROI

**class** *cdl.obj.ImageROI* (*singleobj*: *bool* | *None* = *None*, *inverse*: *bool* = *False*)

Image Regions of Interest

**Parameters**

- **singleobj** – if True, when extracting data defined by ROIs, only one object is created (default to True). If False, one object is created per single ROI. If None, the value is get from the user configuration

- **inverse** – if True, ROI is outside the region

**static** `get_compatible_single_roi_classes()` → `list[Type[BaseSingleImageROI]]`

Return compatible single ROI classes

**to\_mask**(*obj*: `ImageObj`) → `ndarray[bool]`

Create mask from ROI

**Parameters**

**obj** – image object

**Returns**

Mask (boolean array where True values are inside the ROI)

**class** `cdl.obj.ImageDatatypes`(*value*, *names=None*, \*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Image data types

**classmethod** `from_dtype(dtype)`

Return member from NumPy dtype

**classmethod** `check()`

Check if data types are valid

**UINT8** = `'uint8'`

Unsigned integer number stored with 8 bits

**UINT16** = `'uint16'`

Unsigned integer number stored with 16 bits

**INT16** = `'int16'`

Signed integer number stored with 16 bits

**FLOAT32** = `'float32'`

Float number stored with 32 bits

**FLOAT64** = `'float64'`

Float number stored with 64 bits

## 3.4 Computation (`cdl.computation`)

This package contains the computation functions used by the DataLab project. Those functions operate directly on DataLab objects (i.e. `cdl.obj.SignalObj` and `cdl.obj.ImageObj`) and are designed to be used in the DataLab pipeline, but can be used independently as well.

**See also:**

The `cdl.computation` package is the main entry point for the DataLab computation functions when manipulating DataLab objects. See the `cdl.algorithms` package for algorithms that operate directly on NumPy arrays.

Each computation module defines a set of computation objects, that is, functions that implement processing features and classes that implement the corresponding parameters (in the form of `guidata.dataset.datatypes.Dataset` subclasses). The computation functions takes a DataLab object (e.g. `cdl.obj.SignalObj`) and a parameter object (e.g. `cdl.param.MovingAverageParam`) as input and return a DataLab object as output (the result of the computation). The parameter object is used to configure the computation function (e.g. the size of the moving average window).

In DataLab overall architecture, the purpose of this package is to provide the computation functions that are used by the `cdl.core.gui.processor` module, based on the algorithms defined in the `cdl.algorithms` module and on the data model defined in the `cdl.obj` (or `cdl.core.model`) module.

The computation modules are organized in subpackages according to their purpose. The following subpackages are available:

- `cdl.computation.base`: Common processing features
- `cdl.computation.signal`: Signal processing features
- `cdl.computation.image`: Image processing features

### 3.4.1 Common processing features

**class** `cdl.computation.base.ArithmeticParam`

Arithmetic parameters

**operator**

Single choice from: '+', '-', '×', '/'. Default: '+'.  
**Type**  
`guidata.dataset.dataitems.ChoiceItem`

**factor**

Default: 1.0.  
**Type**  
`guidata.dataset.dataitems.FloatItem`

**constant**

Default: 0.0.  
**Type**  
`guidata.dataset.dataitems.FloatItem`

**operation**

Default: '·'.  
**Type**  
`guidata.dataset.dataitems.StringItem`

**restore\_dtype**

Result. Default: True.  
**Type**  
`guidata.dataset.dataitems.BoolItem`

**classmethod** `create(operator: str, factor: float, constant: float, operation: str, restore_dtype: bool) → cdl.computation.base.ArithmeticParam`

Returns a new instance of `ArithmeticParam` with the fields set to the given values.

**Parameters**

- **operator** (*str*) – Single choice from: '+', '-', '×', '/'. Default: '+'.  
• **factor** (*float*) – Default: 1.0.  
• **constant** (*float*) – Default: 0.0.  
• **operation** (*str*) – Default: '·'.

- **restore\_dtype** (*bool*) – Result. Default: True.

**Returns**

New instance of *ArithmeticParam*.

**get\_operation()** → *str*

Return the operation string

**update\_operation**(*\_item*, *\_value*)

Update the operation item

**class** *cdl.computation.base.GaussianParam*

Gaussian filter parameters

**sigma**

. Default: 1.0.

**Type**

*guidata.dataset.dataitems.FloatItem*

**classmethod create**(*sigma: float*) → *cdl.computation.base.GaussianParam*

Returns a new instance of *GaussianParam* with the fields set to the given values.

**Parameters**

**sigma** (*float*) – . Default: 1.0.

**Returns**

New instance of *GaussianParam*.

**class** *cdl.computation.base.MovingAverageParam*

Moving average parameters

**n**

Size of the moving window. Integer higher than 1. Default: 3.

**Type**

*guidata.dataset.dataitems.IntItem*

**mode**

Mode of the filter: - 'reflect': reflect the data at the boundary - 'constant': pad with a constant value - 'nearest': pad with the nearest value - 'mirror': reflect the data at the boundary with the data itself - 'wrap': circular boundary. Single choice from: 'reflect', 'constant', 'nearest', 'mirror', 'wrap'. Default: 'reflect'.

**Type**

*guidata.dataset.dataitems.ChoiceItem*

**classmethod create**(*n: int*, *mode: str*) → *cdl.computation.base.MovingAverageParam*

Returns a new instance of *MovingAverageParam* with the fields set to the given values.

**Parameters**

- **n** (*int*) – Size of the moving window. Integer higher than 1. Default: 3.
- **mode** (*str*) – Mode of the filter: - 'reflect': reflect the data at the boundary - 'constant': pad with a constant value - 'nearest': pad with the nearest value - 'mirror': reflect the data at the boundary with the data itself - 'wrap': circular boundary. Single choice from: 'reflect', 'constant', 'nearest', 'mirror', 'wrap'. Default: 'reflect'.

**Returns**

New instance of *MovingAverageParam*.



**class** `cdl.computation.base.MovingMedianParam`

Moving median parameters

**n**

Size of the moving window. Integer higher than 1, odd. Default: 3.

**Type**`guidata.dataset.dataitems.IntItem`**mode**

Mode of the filter: - 'reflect': reflect the data at the boundary - 'constant': pad with a constant value - 'nearest': pad with the nearest value - 'mirror': reflect the data at the boundary with the data itself - 'wrap': circular boundary. Single choice from: 'reflect', 'constant', 'nearest', 'mirror', 'wrap'. Default: 'nearest'.

**Type**`guidata.dataset.dataitems.ChoiceItem`**classmethod** `create(n: int, mode: str) → cdl.computation.base.MovingMedianParam`Returns a new instance of `MovingMedianParam` with the fields set to the given values.**Parameters**

- **n** (`int`) – Size of the moving window. Integer higher than 1, odd. Default: 3.
- **mode** (`str`) – Mode of the filter: - 'reflect': reflect the data at the boundary - 'constant': pad with a constant value - 'nearest': pad with the nearest value - 'mirror': reflect the data at the boundary with the data itself - 'wrap': circular boundary. Single choice from: 'reflect', 'constant', 'nearest', 'mirror', 'wrap'. Default: 'nearest'.

**Returns**New instance of `MovingMedianParam`.**class** `cdl.computation.base.ClipParam`

Data clipping parameters

**lower**

Lower clipping value. Default: None.

**Type**`guidata.dataset.dataitems.FloatItem`**upper**

Upper clipping value. Default: None.

**Type**`guidata.dataset.dataitems.FloatItem`**classmethod** `create(lower: float, upper: float) → cdl.computation.base.ClipParam`Returns a new instance of `ClipParam` with the fields set to the given values.**Parameters**

- **lower** (`float`) – Lower clipping value. Default: None.
- **upper** (`float`) – Upper clipping value. Default: None.

**Returns**New instance of `ClipParam`.**class** `cdl.computation.base.NormalizeParam`

Normalize parameters

**method**

Normalize with respect to. Single choice from: 'maximum', 'amplitude', 'area', 'energy', 'rms'. Default: 'maximum'.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**classmethod** `create(method: str) → cdl.computation.base.NormalizeParam`

Returns a new instance of `NormalizeParam` with the fields set to the given values.

**Parameters**

**method** (*str*) – Normalize with respect to. Single choice from: 'maximum', 'amplitude', 'area', 'energy', 'rms'. Default: 'maximum'.

**Returns**

New instance of `NormalizeParam`.

**class** `cdl.computation.base.HistogramParam`

Histogram parameters

**bins**

Number of bins. Integer higher than 1. Default: 256.

**Type**

`guidata.dataset.dataitems.IntItem`

**lower**

Lower limit. Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`

**upper**

Upper limit. Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`

**classmethod** `create(bins: int, lower: float, upper: float) → cdl.computation.base.HistogramParam`

Returns a new instance of `HistogramParam` with the fields set to the given values.

**Parameters**

- **bins** (*int*) – Number of bins. Integer higher than 1. Default: 256.
- **lower** (*float*) – Lower limit. Default: None.
- **upper** (*float*) – Upper limit. Default: None.

**Returns**

New instance of `HistogramParam`.

**get\_suffix**(*data: ndarray*) → *str*

Return suffix for the histogram computation

**Parameters**

**data** – data array

**class** `cdl.computation.base.FFTParam`

FFT parameters

**shift**

Shift zero frequency to center. Default: None.

**Type**

`guidata.dataset.dataitems.BoolItem`

**classmethod** `create(shift: bool) → cdl.computation.base.FFTParam`

Returns a new instance of `FFTParam` with the fields set to the given values.

**Parameters**

**shift** (*bool*) – Shift zero frequency to center. Default: None.

**Returns**

New instance of `FFTParam`.

**class** `cdl.computation.base.SpectrumParam`

Spectrum parameters

**log**

Default: False.

**Type**

`guidata.dataset.dataitems.BoolItem`

**classmethod** `create(log: bool) → cdl.computation.base.SpectrumParam`

Returns a new instance of `SpectrumParam` with the fields set to the given values.

**Parameters**

**log** (*bool*) – Default: False.

**Returns**

New instance of `SpectrumParam`.

**class** `cdl.computation.base.ConstantParam`

Parameter used to set a constant value to used in operations

**value**

Constant value. Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`

**classmethod** `create(value: float) → cdl.computation.base.ConstantParam`

Returns a new instance of `ConstantParam` with the fields set to the given values.

**Parameters**

**value** (*float*) – Constant value. Default: None.

**Returns**

New instance of `ConstantParam`.

`cdl.computation.base.dst_11(src: SignalObj | ImageObj, name: str, suffix: str | None = None) → SignalObj | ImageObj`

Create a result object, as returned by the callback function of the `cdl.core.gui.processor.base.BaseProcessor.compute_11()` method

**Parameters**

- **src** – source signal or image object

- **name** – name of the function. If provided, the title of the result object will be *{name}({src.short\_id}){suffix}*, unless the name is a single character, in which case the title will be *{src.short\_id}{name}{suffix}* where *name* is an operator and *suffix* is the other term of the operation.
- **suffix** – suffix to add to the title. Optional.

**Returns**

Result signal or image object

```
cdl.computation.base.dst_nln(src1: SignalObj | ImageObj, src2: SignalObj | ImageObj, name: str, suffix: str | None = None) → SignalObj | ImageObj
```

Create a result object, as returned by the callback function of the `cdl.core.gui.processor.base.BaseProcessor.compute_nln()` method

**Parameters**

- **src1** – input signal or image object
- **src2** – input signal or image object
- **name** – name of the processing function

**Returns**

Output signal or image object

```
cdl.computation.base.new_signal_result(src: SignalObj | ImageObj, name: str, suffix: str | None = None, units: tuple[str, str] | None = None, labels: tuple[str, str] | None = None) → SignalObj
```

Create new signal object as a result of a `compute_11` function

As opposed to the `dst_11` functions, this function creates a new signal object without copying the original object metadata, except for the “source” entry.

**Parameters**

- **src** – input signal or image object
- **name** – name of the processing function
- **suffix** – suffix to add to the title
- **units** – units of the output signal
- **labels** – labels of the output signal

**Returns**

Output signal object

```
cdl.computation.base.calc_resultproperties(title: str, obj: SignalObj | ImageObj, labeledfuncs: dict[str, Callable]) → ResultProperties
```

Calculate result properties by executing a computation function on a signal/image object.

**Parameters**

- **title** – title of the result properties
- **obj** – signal or image object
- **labeledfuncs** – dictionary of labeled computation functions. The keys are the labels of the computation functions and the values are the functions themselves (each function must take a single argument - which is the data of the ROI or the whole signal/image - and return a float)

**Returns**

Result properties object

### 3.4.2 Signal processing features

`cdl.computation.signal.restore_data_outside_roi(dst: SignalObj, src: SignalObj) → None`

Restore data outside the Region Of Interest (ROI) of the input signal after a computation, only if the input signal has a ROI, and if the output signal has the same ROI as the input signal, and if the data types are the same, and if the shapes are the same. Otherwise, do nothing.

**Parameters**

- **dst** – destination signal object
- **src** – source signal object

`class cdl.computation.signal.Wrap11Func(func: Callable, *args: Any, **kwargs: Any)`

Wrap a 1 array → 1 array function (the simple case of  $y1 = f(y0)$ ) to produce a 1 signal → 1 signal function, which can be used inside DataLab's infrastructure to perform computations with `cdl.core.gui.processor.signal.SignalProcessor`.

This wrapping mechanism using a class is necessary for the resulted function to be pickable by the multiprocessing module.

The instance of this wrapper is callable and returns a `cdl.obj.SignalObj` object.

**Example**

```
>>> import numpy as np
>>> from cdl.computation.signal import Wrap11Func
>>> import cdl.obj
>>> def square(y):
...     return y**2
>>> compute_square = Wrap11Func(square)
>>> x = np.linspace(0, 10, 100)
>>> y = np.sin(x)
>>> sig0 = cdl.obj.create_signal("Example", x, y)
>>> sig1 = compute_square(sig0)
```

**Parameters**

- **func** – 1 array → 1 array function
- **\*args** – Additional positional arguments to pass to the function
- **\*\*kwargs** – Additional keyword arguments to pass to the function

`cdl.computation.signal.compute_addition(dst: SignalObj, src: SignalObj) → SignalObj`

Add **dst** and **src** signals and return **dst** signal modified in place

**Parameters**

- **dst** – destination signal
- **src** – source signal

**Returns**

Modified **dst** signal (modified in place)

`cdl.computation.signal.compute_product(dst: SignalObj, src: SignalObj) → SignalObj`

Multiply **dst** and **src** signals and return **dst** signal modified in place

**Parameters**

- **dst** – destination signal
- **src** – source signal

**Returns**

Modified **dst** signal (modified in place)

`cdl.computation.signal.compute_addition_constant(src: SignalObj, p: ConstantParam) → SignalObj`

Add **dst** and a constant value and return a the new result signal object

**Parameters**

- **src** – input signal object
- **p** – constant value

**Returns**

Result signal object **src + p.value** (new object)

`cdl.computation.signal.compute_difference_constant(src: SignalObj, p: ConstantParam) → SignalObj`

Subtract a constant value from a signal

**Parameters**

- **src** – input signal object
- **p** – constant value

**Returns**

Result signal object **src - p.value** (new object)

`cdl.computation.signal.compute_product_constant(src: SignalObj, p: ConstantParam) → SignalObj`

Multiply **dst** by a constant value and return the new result signal object

**Parameters**

- **src** – input signal object
- **p** – constant value

**Returns**

Result signal object **src \* p.value** (new object)

`cdl.computation.signal.compute_division_constant(src: SignalObj, p: ConstantParam) → SignalObj`

Divide a signal by a constant value

**Parameters**

- **src** – input signal object
- **p** – constant value

**Returns**

Result signal object **src / p.value** (new object)

---

`cdl.computation.signal.compute_arithmetic`(*src1*: *SignalObj*, *src2*: *SignalObj*, *p*: *ArithmeticParam*) → *SignalObj*

Perform arithmetic operation on two signals

**Parameters**

- **src1** – source signal 1
- **src2** – source signal 2
- **p** – parameters

**Returns**

Result signal object

`cdl.computation.signal.compute_difference`(*src1*: *SignalObj*, *src2*: *SignalObj*) → *SignalObj*

Compute difference between two signals

---

**Note:** If uncertainty is available, it is propagated.

---

**Parameters**

- **src1** – source signal 1
- **src2** – source signal 2

**Returns**

Result signal object **src1 - src2**

`cdl.computation.signal.compute_quadratic_difference`(*src1*: *SignalObj*, *src2*: *SignalObj*) → *SignalObj*

Compute quadratic difference between two signals

---

**Note:** If uncertainty is available, it is propagated.

---

**Parameters**

- **src1** – source signal 1
- **src2** – source signal 2

**Returns**

Result signal object (**src1 - src2**) / sqrt(2.0)

`cdl.computation.signal.compute_division`(*src1*: *SignalObj*, *src2*: *SignalObj*) → *SignalObj*

Compute division between two signals

**Parameters**

- **src1** – source signal 1
- **src2** – source signal 2

**Returns**

Result signal object **src1 / src2**

`cdl.computation.signal.extract_multiple_roi(src: SignalObj, group: DataSetGroup) → SignalObj`

Extract multiple regions of interest from data

**Parameters**

- **src** – source signal
- **group** – group of parameters

**Returns**

Signal with multiple regions of interest

`cdl.computation.signal.extract_single_roi(src: SignalObj, p: ROI1DParam) → SignalObj`

Extract single region of interest from data

**Parameters**

- **src** – source signal
- **p** – ROI parameters

**Returns**

Signal with single region of interest

`cdl.computation.signal.compute_swap_axes(src: SignalObj) → SignalObj`

Swap axes

**Parameters**

**src** – source signal

**Returns**

Result signal object

`cdl.computation.signal.compute_inverse(src: SignalObj) → SignalObj`

Compute inverse with `numpy.invert`

**Parameters**

**src** – source signal

**Returns**

Result signal object

`cdl.computation.signal.compute_abs(src: SignalObj) → SignalObj`

Compute absolute value with `numpy.absolute`

**Parameters**

**src** – source signal

**Returns**

Result signal object

`cdl.computation.signal.compute_re(src: SignalObj) → SignalObj`

Compute real part with `numpy.real()`

**Parameters**

**src** – source signal

**Returns**

Result signal object

`cdl.computation.signal.compute_im(src: SignalObj) → SignalObj`

Compute imaginary part with `numpy.imag()`



**Parameters****src** – source signal**Returns**

Result signal object

**class** `cdl.computation.signal.DataTypeSParam`

Convert signal data type parameters

**dtype\_str**

Destination data type. Output image data type. Single choice from: 'float32', 'float64', 'complex128'. Default: 'float32'.

**Type**`guidata.dataset.dataitems.ChoiceItem`**classmethod** `create(dtype_str: str) → cdl.computation.signal.DataTypeSParam`Returns a new instance of `DataTypeSParam` with the fields set to the given values.**Parameters****dtype\_str** (*str*) – Destination data type. Output image data type. Single choice from: 'float32', 'float64', 'complex128'. Default: 'float32'.**Returns**New instance of `DataTypeSParam`.`cdl.computation.signal.compute_astype(src: SignalObj, p: DataTypeSParam) → SignalObj`Convert data type with `numpy.astype()`**Parameters**

- **src** – source signal
- **p** – parameters

**Returns**

Result signal object

`cdl.computation.signal.compute_log10(src: SignalObj) → SignalObj`Compute Log10 with `numpy.log10`**Parameters****src** – source signal**Returns**

Result signal object

`cdl.computation.signal.compute_exp(src: SignalObj) → SignalObj`Compute exponential with `numpy.exp`**Parameters****src** – source signal**Returns**

Result signal object

`cdl.computation.signal.compute_sqrt(src: SignalObj) → SignalObj`Compute square root with `numpy.sqrt`**Parameters****src** – source signal

**Returns**

Result signal object

**class** `cdl.computation.signal.PowerParam`

Power parameters

**power**

Default: 2.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**classmethod** `create(power: float) → cdl.computation.signal.PowerParam`

Returns a new instance of `PowerParam` with the fields set to the given values.

**Parameters**

**power** (*float*) – Default: 2.0.

**Returns**

New instance of `PowerParam`.

`cdl.computation.signal.compute_power(src: SignalObj, p: PowerParam) → SignalObj`

Compute power with `numpy.power`

**Parameters**

- **src** – source signal
- **p** – parameters

**Returns**

Result signal object

**class** `cdl.computation.signal.PeakDetectionParam`

Peak detection parameters

**threshold**

Integer between 0 and 100, unit: %. Default: 30.

**Type**

`guidata.dataset.dataitems.IntItem`

**min\_dist**

Minimum distance. Integer higher than 1, unit: points. Default: 1.

**Type**

`guidata.dataset.dataitems.IntItem`

**classmethod** `create(threshold: int, min_dist: int) → cdl.computation.signal.PeakDetectionParam`

Returns a new instance of `PeakDetectionParam` with the fields set to the given values.

**Parameters**

- **threshold** (*int*) – Integer between 0 and 100, unit: %. Default: 30.
- **min\_dist** (*int*) – Minimum distance. Integer higher than 1, unit: points. Default: 1.

**Returns**

New instance of `PeakDetectionParam`.

`cdl.computation.signal.compute_peak_detection(src: SignalObj, p: PeakDetectionParam) → SignalObj`  
 Peak detection with `cdl.algorithms.signal.peak_indices()`

**Parameters**

- **src** – source signal
- **p** – parameters

**Returns**

Result signal object

`cdl.computation.signal.compute_normalize(src: SignalObj, p: NormalizeParam) → SignalObj`  
 Normalize data with `cdl.algorithms.signal.normalize()`

**Parameters**

- **src** – source signal
- **p** – parameters

**Returns**

Result signal object

`cdl.computation.signal.compute_derivative(src: SignalObj) → SignalObj`  
 Compute derivative with `numpy.gradient()`

**Parameters**

**src** – source signal

**Returns**

Result signal object

`cdl.computation.signal.compute_integral(src: SignalObj) → SignalObj`  
 Compute integral with `scipy.integrate.cumulative_trapezoid()`

**Parameters**

**src** – source signal

**Returns**

Result signal object

**class** `cdl.computation.signal.XYCalibrateParam`

Signal calibration parameters

**axis**

Calibrate. Single choice from: 'x', 'y'. Default: 'y'.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**a**

Default: 1.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**b**

Default: 0.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**classmethod** `create(axis: str, a: float, b: float) → cdl.computation.signal.XYCalibrateParam`

Returns a new instance of *XYCalibrateParam* with the fields set to the given values.

**Parameters**

- **axis** (*str*) – Calibrate. Single choice from: ‘x’, ‘y’. Default: ‘y’.
- **a** (*float*) – Default: 1.0.
- **b** (*float*) – Default: 0.0.

**Returns**

New instance of *XYCalibrateParam*.

`cdl.computation.signal.compute_calibration(src: SignalObj, p: XYCalibrateParam) → SignalObj`

Compute linear calibration

**Parameters**

- **src** – source signal
- **p** – parameters

**Returns**

Result signal object

`cdl.computation.signal.compute_clip(src: SignalObj, p: ClipParam) → SignalObj`

Compute maximum data clipping with `numpy.clip()`

**Parameters**

- **src** – source signal
- **p** – parameters

**Returns**

Result signal object

`cdl.computation.signal.compute_offset_correction(src: SignalObj, p: ROI1DParam) → SignalObj`

Correct offset: subtract the mean value of the signal in the specified range (baseline correction)

**Parameters**

- **src** – source signal
- **p** – parameters

**Returns**

Result signal object

`cdl.computation.signal.compute_gaussian_filter(src: SignalObj, p: GaussianParam) → SignalObj`

Compute gaussian filter with `scipy.ndimage.gaussian_filter()`

**Parameters**

- **src** – source signal
- **p** – parameters

**Returns**

Result signal object

`cdl.computation.signal.compute_moving_average(src: SignalObj, p: MovingAverageParam) → SignalObj`

Compute moving average with `scipy.ndimage.uniform_filter()`

**Parameters**

- **src** – source signal
- **p** – parameters

**Returns**

Result signal object

`cdl.computation.signal.compute_moving_median(src: SignalObj, p: MovingMedianParam) → SignalObj`

Compute moving median with `scipy.ndimage.median_filter()`

**Parameters**

- **src** – source signal
- **p** – parameters

**Returns**

Result signal object

`cdl.computation.signal.compute_wiener(src: SignalObj) → SignalObj`

Compute Wiener filter with `scipy.signal.wiener()`

**Parameters**

**src** – source signal

**Returns**

Result signal object

`class cdl.computation.signal.FilterType(value, names=None, *, module=None, qualname=None, type=None, start=1, boundary=None)`

Filter types

`class cdl.computation.signal.BaseHighLowBandParam`

Base class for high-pass, low-pass, band-pass and band-stop filters

**method**

Filter method. Single choice from: 'bessel', 'butter', 'cheby1', 'cheby2', 'ellip'. Default: 'bessel'.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**order**

Filter order. Integer higher than 1. Default: 3.

**Type**

`guidata.dataset.dataitems.IntItem`

**f\_cut0**

Low cutoff frequency. Float higher than 0, non zero, unit: hz. Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`

**f\_cut1**

High cutoff frequency. Float higher than 0, non zero, unit: hz. Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`

**rp**

Passband ripple. Float higher than 0, non zero, unit: db. Default: 1.

**Type**`guidata.dataset.dataitems.FloatItem`**rs**

Stopband attenuation. Float higher than 0, non zero, unit: db. Default: 1.

**Type**`guidata.dataset.dataitems.FloatItem`

**classmethod** **create**(*method: str, order: int, f\_cut0: float, f\_cut1: float, rp: float, rs: float*) → `cdl.computation.signal.BaseHighLowBandParam`

Returns a new instance of `BaseHighLowBandParam` with the fields set to the given values.

**Parameters**

- **method** (*str*) – Filter method. Single choice from: ‘bessel’, ‘butter’, ‘cheby1’, ‘cheby2’, ‘ellip’. Default: ‘bessel’.
- **order** (*int*) – Filter order. Integer higher than 1. Default: 3.
- **f\_cut0** (*float*) – Low cutoff frequency. Float higher than 0, non zero, unit: hz. Default: None.
- **f\_cut1** (*float*) – High cutoff frequency. Float higher than 0, non zero, unit: hz. Default: None.
- **rp** (*float*) – Passband ripple. Float higher than 0, non zero, unit: db. Default: 1.
- **rs** (*float*) – Stopband attenuation. Float higher than 0, non zero, unit: db. Default: 1.

**Returns**

New instance of `BaseHighLowBandParam`.

**static** **get\_nyquist\_frequency**(*obj: SignalObj*) → `float`

Return the Nyquist frequency of a signal object

**Parameters**

**obj** – signal object

**update\_from\_signal**(*obj: SignalObj*) → `None`

Update the filter parameters from a signal object

**Parameters**

**obj** – signal object

**get\_filter\_params**(*obj: SignalObj*) → `tuple[float | str, float | str]`

Return the filter parameters (a and b) as a tuple. These parameters are used in the `scipy.signal` filter functions (eg. `scipy.signal.filtfilt`).

**Parameters**

**obj** – signal object

**Returns**

filter parameters

**Return type**`tuple`

**class** `cdl.computation.signal.LowPassFilterParam`

Low-pass filter parameters

**method**

Filter method. Single choice from: ‘bessel’, ‘butter’, ‘cheby1’, ‘cheby2’, ‘ellip’. Default: ‘bessel’.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**order**

Filter order. Integer higher than 1. Default: 3.

**Type**

`guidata.dataset.dataitems.IntItem`

**f\_cut0**

Low cutoff frequency. Float higher than 0, non zero, unit: hz. Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`

**f\_cut1**

High cutoff frequency. Float higher than 0, non zero, unit: hz. Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`

**rp**

Passband ripple. Float higher than 0, non zero, unit: db. Default: 1.

**Type**

`guidata.dataset.dataitems.FloatItem`

**rs**

Stopband attenuation. Float higher than 0, non zero, unit: db. Default: 1.

**Type**

`guidata.dataset.dataitems.FloatItem`

**classmethod** **create**(*method: str, order: int, f\_cut0: float, f\_cut1: float, rp: float, rs: float*) → *cdl.computation.signal.LowPassFilterParam*

Returns a new instance of *LowPassFilterParam* with the fields set to the given values.

**Parameters**

- **method** (*str*) – Filter method. Single choice from: ‘bessel’, ‘butter’, ‘cheby1’, ‘cheby2’, ‘ellip’. Default: ‘bessel’.
- **order** (*int*) – Filter order. Integer higher than 1. Default: 3.
- **f\_cut0** (*float*) – Low cutoff frequency. Float higher than 0, non zero, unit: hz. Default: None.
- **f\_cut1** (*float*) – High cutoff frequency. Float higher than 0, non zero, unit: hz. Default: None.
- **rp** (*float*) – Passband ripple. Float higher than 0, non zero, unit: db. Default: 1.
- **rs** (*float*) – Stopband attenuation. Float higher than 0, non zero, unit: db. Default: 1.

**Returns**

New instance of *LowPassFilterParam*.

**class** `cdl.computation.signal.HighPassFilterParam`

High-pass filter parameters

**method**

Filter method. Single choice from: 'bessel', 'butter', 'cheby1', 'cheby2', 'ellip'. Default: 'bessel'.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**order**

Filter order. Integer higher than 1. Default: 3.

**Type**

`guidata.dataset.dataitems.IntItem`

**f\_cut0**

Low cutoff frequency. Float higher than 0, non zero, unit: hz. Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`

**f\_cut1**

High cutoff frequency. Float higher than 0, non zero, unit: hz. Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`

**rp**

Passband ripple. Float higher than 0, non zero, unit: db. Default: 1.

**Type**

`guidata.dataset.dataitems.FloatItem`

**rs**

Stopband attenuation. Float higher than 0, non zero, unit: db. Default: 1.

**Type**

`guidata.dataset.dataitems.FloatItem`

**classmethod** **create**(*method: str, order: int, f\_cut0: float, f\_cut1: float, rp: float, rs: float*) → *cdl.computation.signal.HighPassFilterParam*

Returns a new instance of *HighPassFilterParam* with the fields set to the given values.

**Parameters**

- **method** (*str*) – Filter method. Single choice from: 'bessel', 'butter', 'cheby1', 'cheby2', 'ellip'. Default: 'bessel'.
- **order** (*int*) – Filter order. Integer higher than 1. Default: 3.
- **f\_cut0** (*float*) – Low cutoff frequency. Float higher than 0, non zero, unit: hz. Default: None.
- **f\_cut1** (*float*) – High cutoff frequency. Float higher than 0, non zero, unit: hz. Default: None.
- **rp** (*float*) – Passband ripple. Float higher than 0, non zero, unit: db. Default: 1.
- **rs** (*float*) – Stopband attenuation. Float higher than 0, non zero, unit: db. Default: 1.

**Returns**

New instance of *HighPassFilterParam*.

**class** `cdl.computation.signal.BandPassFilterParam`

Band-pass filter parameters



**method**

Filter method. Single choice from: ‘bessel’, ‘butter’, ‘cheby1’, ‘cheby2’, ‘ellip’. Default: ‘bessel’.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**order**

Filter order. Integer higher than 1. Default: 3.

**Type**

`guidata.dataset.dataitems.IntItem`

**f\_cut0**

Low cutoff frequency. Float higher than 0, non zero, unit: hz. Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`

**f\_cut1**

High cutoff frequency. Float higher than 0, non zero, unit: hz. Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`

**rp**

Passband ripple. Float higher than 0, non zero, unit: db. Default: 1.

**Type**

`guidata.dataset.dataitems.FloatItem`

**rs**

Stopband attenuation. Float higher than 0, non zero, unit: db. Default: 1.

**Type**

`guidata.dataset.dataitems.FloatItem`

**classmethod** **create**(*method: str, order: int, f\_cut0: float, f\_cut1: float, rp: float, rs: float*) → *cdl.computation.signal.BandPassFilterParam*

Returns a new instance of *BandPassFilterParam* with the fields set to the given values.

**Parameters**

- **method** (*str*) – Filter method. Single choice from: ‘bessel’, ‘butter’, ‘cheby1’, ‘cheby2’, ‘ellip’. Default: ‘bessel’.
- **order** (*int*) – Filter order. Integer higher than 1. Default: 3.
- **f\_cut0** (*float*) – Low cutoff frequency. Float higher than 0, non zero, unit: hz. Default: None.
- **f\_cut1** (*float*) – High cutoff frequency. Float higher than 0, non zero, unit: hz. Default: None.
- **rp** (*float*) – Passband ripple. Float higher than 0, non zero, unit: db. Default: 1.
- **rs** (*float*) – Stopband attenuation. Float higher than 0, non zero, unit: db. Default: 1.

**Returns**

New instance of *BandPassFilterParam*.

**class** `cdl.computation.signal.BandStopFilterParam`

Band-stop filter parameters

**method**

Filter method. Single choice from: 'bessel', 'butter', 'cheby1', 'cheby2', 'ellip'. Default: 'bessel'.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**order**

Filter order. Integer higher than 1. Default: 3.

**Type**

`guidata.dataset.dataitems.IntItem`

**f\_cut0**

Low cutoff frequency. Float higher than 0, non zero, unit: hz. Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`

**f\_cut1**

High cutoff frequency. Float higher than 0, non zero, unit: hz. Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`

**rp**

Passband ripple. Float higher than 0, non zero, unit: db. Default: 1.

**Type**

`guidata.dataset.dataitems.FloatItem`

**rs**

Stopband attenuation. Float higher than 0, non zero, unit: db. Default: 1.

**Type**

`guidata.dataset.dataitems.FloatItem`

**classmethod** **create**(*method: str, order: int, f\_cut0: float, f\_cut1: float, rp: float, rs: float*) → *cdl.computation.signal.BandStopFilterParam*

Returns a new instance of *BandStopFilterParam* with the fields set to the given values.

**Parameters**

- **method** (*str*) – Filter method. Single choice from: 'bessel', 'butter', 'cheby1', 'cheby2', 'ellip'. Default: 'bessel'.
- **order** (*int*) – Filter order. Integer higher than 1. Default: 3.
- **f\_cut0** (*float*) – Low cutoff frequency. Float higher than 0, non zero, unit: hz. Default: None.
- **f\_cut1** (*float*) – High cutoff frequency. Float higher than 0, non zero, unit: hz. Default: None.
- **rp** (*float*) – Passband ripple. Float higher than 0, non zero, unit: db. Default: 1.
- **rs** (*float*) – Stopband attenuation. Float higher than 0, non zero, unit: db. Default: 1.

**Returns**

New instance of *BandStopFilterParam*.

`cdl.computation.signal.compute_filter(src: SignalObj, p: BaseHighLowBandParam) → SignalObj`  
 Compute frequency filter (low-pass, high-pass, band-pass, band-stop), with `scipy.signal.filtfilt()`

**Parameters**

- **src** – source signal
- **p** – parameters

**Returns**

Result signal object

**class** `cdl.computation.signal.ZeroPadding1DParam`

Zero padding parameters

**strategy**

Single choice from: ‘next\_pow2’, ‘double’, ‘triple’, ‘custom’. Default: ‘custom’.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**n**

Number of points. Number of points to add. Integer higher than 1. Default: None.

**Type**

`guidata.dataset.dataitems.IntItem`

**classmethod** `create(strategy: str, n: int) → cdl.computation.signal.ZeroPadding1DParam`

Returns a new instance of `ZeroPadding1DParam` with the fields set to the given values.

**Parameters**

- **strategy** (*str*) – Single choice from: ‘next\_pow2’, ‘double’, ‘triple’, ‘custom’. Default: ‘custom’.
- **n** (*int*) – Number of points. Number of points to add. Integer higher than 1. Default: None.

**Returns**

New instance of `ZeroPadding1DParam`.

**update\_from\_signal**(*obj: SignalObj*) → None

Update parameters from signal

**choice\_callback**(*item, value*)

Callback for choice item

`cdl.computation.signal.compute_zero_padding(src: SignalObj, p: ZeroPadding1DParam) → SignalObj`

Compute zero padding with `cdl.algorithms.signal.zero_padding()`

**Parameters**

- **src** – source signal
- **p** – parameters

**Returns**

Result signal object

`cdl.computation.signal.compute_fft(src: SignalObj, p: FFTParam | None = None) → SignalObj`

Compute FFT with `cdl.algorithms.signal.fft1d()`

**Parameters**

- **src** – source signal
- **p** – parameters

**Returns**

Result signal object

`cdl.computation.signal.compute_ifft(src: SignalObj, p: FFTParam | None = None) → SignalObj`

Compute iFFT with `cdl.algorithms.signal.ifft1d()`

**Parameters**

- **src** – source signal
- **p** – parameters

**Returns**

Result signal object

`cdl.computation.signal.compute_magnitude_spectrum(src: SignalObj, p: SpectrumParam | None = None) → SignalObj`

Compute magnitude spectrum with `cdl.algorithms.signal.magnitude_spectrum()`

**Parameters**

- **src** – source signal
- **p** – parameters

**Returns**

Result signal object

`cdl.computation.signal.compute_phase_spectrum(src: SignalObj) → SignalObj`

Compute phase spectrum with `cdl.algorithms.signal.phase_spectrum()`

**Parameters**

**src** – source signal

**Returns**

Result signal object

`cdl.computation.signal.compute_psd(src: SignalObj, p: SpectrumParam | None = None) → SignalObj`

Compute power spectral density with `cdl.algorithms.signal.psd()`

**Parameters**

- **src** – source signal
- **p** – parameters

**Returns**

Result signal object

**class** `cdl.computation.signal.PolynomialFitParam`

Polynomial fitting parameters

**degree**

Integer between 1 and 10. Default: 3.

**Type**

`guidata.dataset.dataitems.IntItem`

**classmethod** `create(degree: int) → cdl.computation.signal.PolynomialFitParam`

Returns a new instance of *PolynomialFitParam* with the fields set to the given values.

**Parameters**

**degree** (*int*) – Integer between 1 and 10. Default: 3.

**Returns**

New instance of *PolynomialFitParam*.

`cdl.computation.signal.compute_histogram(src: SignalObj, p: HistogramParam) → SignalObj`

Compute histogram with `numpy.histogram()`

**Parameters**

- **src** – source signal
- **p** – parameters

**Returns**

Result signal object

**class** `cdl.computation.signal.InterpolationParam`

Interpolation parameters

**method**

Interpolation method. Single choice from: 'linear', 'spline', 'quadratic', 'cubic', 'barycentric', 'pchip'. Default: 'linear'.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**fill\_value**

Value to use for points outside the interpolation domain (used only with linear, cubic and pchip methods). Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`

**classmethod** `create(method: str, fill_value: float) → cdl.computation.signal.InterpolationParam`

Returns a new instance of *InterpolationParam* with the fields set to the given values.

**Parameters**

- **method** (*str*) – Interpolation method. Single choice from: 'linear', 'spline', 'quadratic', 'cubic', 'barycentric', 'pchip'. Default: 'linear'.
- **fill\_value** (*float*) – Value to use for points outside the interpolation domain (used only with linear, cubic and pchip methods). Default: None.

**Returns**

New instance of *InterpolationParam*.

`cdl.computation.signal.compute_interpolation(src1: SignalObj, src2: SignalObj, p: InterpolationParam) → SignalObj`

Interpolate data with `cdl.algorithms.signal.interpolate()`

**Parameters**

- **src1** – source signal 1
- **src2** – source signal 2
- **p** – parameters

**Returns**

Result signal object

**class** `cdl.computation.signal.ResamplingParam`

Resample parameters

**method**

Interpolation method. Single choice from: 'linear', 'spline', 'quadratic', 'cubic', 'barycentric', 'pchip'. Default: 'linear'.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**fill\_value**

Value to use for points outside the interpolation domain (used only with linear, cubic and pchip methods). Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`

**xmin**

$X_{\min}$ . Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`

**xmax**

$X_{\max}$ . Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`

**mode**

Single choice from: 'dx', 'nbpts'. Default: 'nbpts'.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**dx**

X. Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`

**nbpts**

Number of points. Default: None.

**Type**

`guidata.dataset.dataitems.IntItem`

**classmethod** `create`(*method*: *str*, *fill\_value*: *float*, *xmin*: *float*, *xmax*: *float*, *mode*: *str*, *dx*: *float*, *nbpts*: *int*)  
→ `cdl.computation.signal.ResamplingParam`

Returns a new instance of `ResamplingParam` with the fields set to the given values.

**Parameters**

- **method** (*str*) – Interpolation method. Single choice from: 'linear', 'spline', 'quadratic', 'cubic', 'barycentric', 'pchip'. Default: 'linear'.
- **fill\_value** (*float*) – Value to use for points outside the interpolation domain (used only with linear, cubic and pchip methods). Default: None.

- **xmin** (*float*) –  $X_{\min}$ . Default: None.
- **xmax** (*float*) –  $X_{\max}$ . Default: None.
- **mode** (*str*) – Single choice from: 'dx', 'nbpts'. Default: 'nbpts'.
- **dx** (*float*) –  $X$ . Default: None.
- **nbpts** (*int*) – Number of points. Default: None.

**Returns**

New instance of *ResamplingParam*.

`cdl.computation.signal.compute_resampling(src: SignalObj, p: ResamplingParam) → SignalObj`

Resample data with *cdl.algorithms.signal.interpolate()*

**Parameters**

- **src** – source signal
- **p** – parameters

**Returns**

Result signal object

**class** `cdl.computation.signal.DetrendingParam`

Detrending parameters

**method**

Detrending method. Single choice from: 'linear', 'constant'. Default: 'linear'.

**Type**

*guidata.dataset.dataitems.ChoiceItem*

**classmethod** `create(method: str) → cdl.computation.signal.DetrendingParam`

Returns a new instance of *DetrendingParam* with the fields set to the given values.

**Parameters**

**method** (*str*) – Detrending method. Single choice from: 'linear', 'constant'. Default: 'linear'.

**Returns**

New instance of *DetrendingParam*.

`cdl.computation.signal.compute_detrending(src: SignalObj, p: DetrendingParam) → SignalObj`

Detrend data with *scipy.signal.detrend()*

**Parameters**

- **src** – source signal
- **p** – parameters

**Returns**

Result signal object

`cdl.computation.signal.compute_XY_mode(src1: SignalObj, src2: SignalObj) → SignalObj`

Simulate the X-Y mode of an oscilloscope.

Use the first signal as the X-axis and the second signal as the Y-axis.

**Parameters**

- **src1** – First input signal (X-axis).
- **src2** – Second input signal (Y-axis).

**Returns**

A signal object representing the X-Y mode.

`cdl.computation.signal.compute_convolution(src1: SignalObj, src2: SignalObj) → SignalObj`

Compute convolution of two signals with `scipy.signal.convolve()`

**Parameters**

- **src1** – source signal 1
- **src2** – source signal 2

**Returns**

Result signal object

**class** `cdl.computation.signal.WindowingParam`

Windowing parameters

**method**

Single choice from: ‘barthann’, ‘bartlett’, ‘blackman’, ‘blackman-harris’, ‘bohman’, ‘boxcar’, ‘cosine’, ‘exponential’, ‘flat-top’, ‘gaussian’, ‘hamming’, ‘hanning’, ‘kaiser’, ‘lanczos’, ‘nuttall’, ‘parzen’, ‘rectangular’, ‘taylor’, ‘tukey’. Default: ‘hamming’.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**alpha**

Shape parameter of the tukey windowing function. Default: 0.5.

**Type**

`guidata.dataset.dataitems.FloatItem`

**beta**

Shape parameter of the kaiser windowing function. Default: 14.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**sigma**

Shape parameter of the gaussian windowing function. Default: 0.5.

**Type**

`guidata.dataset.dataitems.FloatItem`

**classmethod** `create(method: str, alpha: float, beta: float, sigma: float) → cdl.computation.signal.WindowingParam`

Returns a new instance of `WindowingParam` with the fields set to the given values.

**Parameters**

- **method** (*str*) – Single choice from: ‘barthann’, ‘bartlett’, ‘blackman’, ‘blackman-harris’, ‘bohman’, ‘boxcar’, ‘cosine’, ‘exponential’, ‘flat-top’, ‘gaussian’, ‘hamming’, ‘hanning’, ‘kaiser’, ‘lanczos’, ‘nuttall’, ‘parzen’, ‘rectangular’, ‘taylor’, ‘tukey’. Default: ‘hamming’.
- **alpha** (*float*) – Shape parameter of the tukey windowing function. Default: 0.5.
- **beta** (*float*) – Shape parameter of the kaiser windowing function. Default: 14.0.
- **sigma** (*float*) – Shape parameter of the gaussian windowing function. Default: 0.5.

**Returns**

New instance of `WindowingParam`.



`cdl.computation.signal.compute_windowing(src: SignalObj, p: WindowingParam) → SignalObj`

Compute windowing (available methods: hamming, hanning, bartlett, blackman, tukey, rectangular) with `cdl.algorithms.signal.windowing()`

**Parameters**

- **dst** – destination signal
- **src** – source signal

**Returns**

Result signal object

`cdl.computation.signal.compute_reverse_x(src: SignalObj) → SignalObj`

Reverse x-axis

**Parameters**

**src** – source signal

**Returns**

Result signal object

**class** `cdl.computation.signal.AngleUnitParam`

Choice of angle unit.

**unit**

Angle unit. Single choice from: 'rad', 'deg'. Default: 'rad'.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**classmethod** `create(unit: str) → cdl.computation.signal.AngleUnitParam`

Returns a new instance of `AngleUnitParam` with the fields set to the given values.

**Parameters**

**unit** (*str*) – Angle unit. Single choice from: 'rad', 'deg'. Default: 'rad'.

**Returns**

New instance of `AngleUnitParam`.

`cdl.computation.signal.compute_cartesian2polar(src: SignalObj, p: AngleUnitParam) → SignalObj`

Convert cartesian coordinates to polar coordinates with `cdl.algorithms.coordinates.cartesian2polar()`.

**Parameters**

- **src** – Source signal.
- **p** – Parameters.

**Returns**

Result signal object.

`cdl.computation.signal.compute_polar2cartesian(src: SignalObj, p: AngleUnitParam) → SignalObj`

Convert polar coordinates to cartesian coordinates with `cdl.algorithms.coordinates.polar2cartesian()`.

**Parameters**

- **src** – Source signal.
- **p** – Parameters.

**Returns**

Result signal object.

---

**Note:** This function assumes that the x-axis represents the radius and the y-axis represents the angle. Negative values are not allowed for the radius, and will be clipped to 0 (a warning will be raised).

---

**class** `cdl.computation.signal.AllanVarianceParam`

Allan variance parameters

**max\_tau**

Max . Integer higher than 1, unit: pts. Default: 100.

**Type**

`guidata.dataset.dataitems.IntItem`

**classmethod** `create(max_tau: int) → cdl.computation.signal.AllanVarianceParam`

Returns a new instance of `AllanVarianceParam` with the fields set to the given values.

**Parameters**

**max\_tau** (*int*) – Max . Integer higher than 1, unit: pts. Default: 100.

**Returns**

New instance of `AllanVarianceParam`.

`cdl.computation.signal.compute_allan_variance(src: SignalObj, p: AllanVarianceParam) → SignalObj`

Compute Allan variance with `cdl.algorithms.signal.allan_variance()`

**Parameters**

- **src** – source signal
- **p** – parameters

**Returns**

Result signal object

`cdl.computation.signal.compute_allan_deviation(src: SignalObj, p: AllanVarianceParam) → SignalObj`

Compute Allan deviation with `cdl.algorithms.signal.allan_deviation()`

**Parameters**

- **src** – source signal
- **p** – parameters

**Returns**

Result signal object

`cdl.computation.signal.compute_overlapping_allan_variance(src: SignalObj, p: AllanVarianceParam) → SignalObj`

Compute Overlapping Allan variance.

**Parameters**

- **src** – source signal
- **p** – parameters

**Returns**

Result signal object

`cdl.computation.signal.compute_modified_allan_variance(src: SignalObj, p: AllanVarianceParam) → SignalObj`

Compute Modified Allan variance.

**Parameters**

- **src** – source signal
- **p** – parameters

**Returns**

Result signal object

`cdl.computation.signal.compute_hadamard_variance(src: SignalObj, p: AllanVarianceParam) → SignalObj`

Compute Hadamard variance.

**Parameters**

- **src** – source signal
- **p** – parameters

**Returns**

Result signal object

`cdl.computation.signal.compute_total_variance(src: SignalObj, p: AllanVarianceParam) → SignalObj`

Compute Total variance.

**Parameters**

- **src** – source signal
- **p** – parameters

**Returns**

Result signal object

`cdl.computation.signal.compute_time_deviation(src: SignalObj, p: AllanVarianceParam) → SignalObj`

Compute Time Deviation (TDEV).

**Parameters**

- **src** – source signal
- **p** – parameters

**Returns**

Result signal object

`cdl.computation.signal.calc_resultshape(title: str, shape: Literal['rectangle', 'circle', 'ellipse', 'segment', 'marker', 'point', 'polygon'], obj: SignalObj, func: Callable, *args: Any, add_label: bool = False) → ResultShape | None`

Calculate result shape by executing a computation function on a signal object, taking into account the signal ROIs.

**Parameters**

- **title** – result title
- **shape** – result shape kind
- **obj** – input image object
- **func** – computation function

- **\*args** – computation function arguments
- **add\_label** – if True, add a label item (and the geometrical shape) to plot (default to False)

**Returns**

Result shape object or None if no result is found

**Warning:** The computation function must take either a single argument (the data) or multiple arguments (the data followed by the computation parameters).

Moreover, the computation function must return a 1D NumPy array (or a list, or a tuple) containing the result of the computation.

**class** `cdl.computation.signal.FWHMParam`

FWHM parameters

**method**

Single choice from: 'zero-crossing', 'gauss', 'lorentz', 'voigt'. Default: 'zero-crossing'.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**xmin**

$X_{\text{MIN}}$ . Lower x boundary (empty for no limit, i.e. Start of the signal). Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`

**xmax**

$X_{\text{MAX}}$ . Upper x boundary (empty for no limit, i.e. End of the signal). Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`

**classmethod** `create(method: str, xmin: float, xmax: float) → cdl.computation.signal.FWHMParam`

Returns a new instance of `FWHMParam` with the fields set to the given values.

**Parameters**

- **method** (`str`) – Single choice from: 'zero-crossing', 'gauss', 'lorentz', 'voigt'. Default: 'zero-crossing'.
- **xmin** (`float`) –  $X_{\text{MIN}}$ . Lower x boundary (empty for no limit, i.e. Start of the signal). Default: None.
- **xmax** (`float`) –  $X_{\text{MAX}}$ . Upper x boundary (empty for no limit, i.e. End of the signal). Default: None.

**Returns**

New instance of `FWHMParam`.

`cdl.computation.signal.compute_fwhm(obj: SignalObj, param: FWHMParam) → ResultShape | None`

Compute FWHM with `cdl.algorithms.signal.fwhm()`

**Parameters**

- **obj** – source signal
- **param** – parameters

**Returns**

Segment coordinates

`cdl.computation.signal.compute_fw1e2(obj: SignalObj) → ResultShape | None`

Compute FW at  $1/e^2$  with `cdl.algorithms.signal.fw1e2()`

**Parameters**

**obj** – source signal

**Returns**

Segment coordinates

**class** `cdl.computation.signal.OrdinateParam`

Ordinate parameter

**y**

Ordinate. Default: 0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**classmethod** `create(y: float) → cdl.computation.signal.OrdinateParam`

Returns a new instance of `OrdinateParam` with the fields set to the given values.

**Parameters**

**y** (*float*) – Ordinate. Default: 0.

**Returns**

New instance of `OrdinateParam`.

`cdl.computation.signal.compute_full_width_at_y(obj: SignalObj, p: OrdinateParam) → ResultProperties`

Compute full width at a given y value for a signal object.

**Parameters**

- **obj** – The signal object containing x and y data.
- **p** – The ordinate parameter dataset

**Returns**

Segment coordinates

`cdl.computation.signal.compute_x_at_y(obj: SignalObj, p: OrdinateParam) → ResultProperties`

Compute the smallest x-value at a given y-value for a signal object.

**Parameters**

- **obj** – The signal object containing x and y data.
- **p** – The parameter dataset for finding the abscissa.

**Returns**

An object containing the x-value.

**class** `cdl.computation.signal.AbscissaParam`

Abscissa parameter

**x**

Abscissa. Default: 0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**classmethod** `create(x: float) → cdl.computation.signal.AbscissaParam`

Returns a new instance of `AbscissaParam` with the fields set to the given values.

**Parameters**

`x (float)` – Abscissa. Default: 0.

**Returns**

New instance of `AbscissaParam`.

`cdl.computation.signal.compute_y_at_x(obj: SignalObj, p: AbscissaParam) → ResultProperties`

Compute the smallest y-value at a given x-value for a signal object.

**Parameters**

- `obj` – The signal object containing x and y data.
- `p` – The parameter dataset for finding the ordinate.

**Returns**

An object containing the y-value.

`cdl.computation.signal.compute_stats(obj: SignalObj) → ResultProperties`

Compute statistics on a signal

**Parameters**

`obj` – source signal

**Returns**

Result properties object

`cdl.computation.signal.compute_bandwidth_3db(obj: SignalObj) → ResultProperties`

Compute bandwidth at -3 dB with `cdl.algorithms.signal.bandwidth()`

**Parameters**

`obj` – source signal

**Returns**

Result properties with bandwidth

**class** `cdl.computation.signal.DynamicParam`

Parameters for dynamic range computation (ENOB, SNR, SINAD, THD, SFDR)

**full\_scale**

Float higher than 0.0, unit: v. Default: 0.16.

**Type**

`guidata.dataset.dataitems.FloatItem`

**unit**

Unit for sinad. Single choice from: 'dBc', 'dBFS'. Default: 'dBc'.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**nb\_harm**

Number of harmonics. Number of harmonics to consider for thd. Integer higher than 1. Default: 5.

**Type**

`guidata.dataset.dataitems.IntItem`

**classmethod** `create(full_scale: float, unit: str, nb_harm: int) → cdl.computation.signal.DynamicParam`

Returns a new instance of *DynamicParam* with the fields set to the given values.

#### Parameters

- **full\_scale** (*float*) – Float higher than 0.0, unit: v. Default: 0.16.
- **unit** (*str*) – Unit for sinad. Single choice from: ‘dBc’, ‘dBFS’. Default: ‘dBc’.
- **nb\_harm** (*int*) – Number of harmonics. Number of harmonics to consider for thd. Integer higher than 1. Default: 5.

#### Returns

New instance of *DynamicParam*.

`cdl.computation.signal.compute_dynamic_parameters(src: SignalObj, p: DynamicParam) → ResultProperties`

Compute Dynamic parameters using the following functions:

- Freq: `cdl.algorithms.signal.sinus_frequency()`
- ENOB: `cdl.algorithms.signal.enob()`
- SNR: `cdl.algorithms.signal.snr()`
- SINAD: `cdl.algorithms.signal.sinad()`
- THD: `cdl.algorithms.signal.thd()`
- SFDR: `cdl.algorithms.signal.sfdr()`

#### Parameters

- **src** – source signal
- **p** – parameters

#### Returns

Result properties with ENOB, SNR, SINAD, THD, SFDR

`cdl.computation.signal.compute_sampling_rate_period(obj: SignalObj) → ResultProperties`

Compute sampling rate and period using the following functions:

- fs: `cdl.algorithms.signal.sampling_rate()`
- T: `cdl.algorithms.signal.sampling_period()`

#### Parameters

**obj** – source signal

#### Returns

Result properties with sampling rate and period

`cdl.computation.signal.compute_contrast(obj: SignalObj) → ResultProperties`

Compute contrast with `cdl.algorithms.signal.contrast()`

`cdl.computation.signal.compute_x_at_minmax(obj: SignalObj) → ResultProperties`

Compute the smallest argument at the minima and the smallest argument at the maxima.

#### Parameters

**obj** – The signal object.

**Returns**

An object containing the x-values at the minima and the maxima.

### 3.4.3 Image processing features

#### Base image processing features

`cdl.computation.image.restore_data_outside_roi(dst: ImageObj, src: ImageObj) → None`

Restore data outside the Region Of Interest (ROI) of the input image after a computation, only if the input image has a ROI, and if the output image has the same ROI as the input image, and if the data types are compatible, and if the shapes are the same. Otherwise, do nothing.

**Parameters**

- **dst** – output image object
- **src** – input image object

**class** `cdl.computation.image.Wrap11Func(func: Callable, *args: Any, **kwargs: Any)`

Wrap a 1 array → 1 array function to produce a 1 image → 1 image function, which can be used inside DataLab's infrastructure to perform computations with `cdl.core.gui.processor.image.ImageProcessor`.

This wrapping mechanism using a class is necessary for the resulted function to be pickable by the multiprocessing module.

The instance of this wrapper is callable and returns a `cdl.obj.ImageObj` object.

**Example**

```
>>> import numpy as np
>>> from cdl.computation.image import Wrap11Func
>>> import cdl.obj
>>> def add_noise(data):
...     return data + np.random.random(data.shape)
>>> compute_add_noise = Wrap11Func(add_noise)
>>> data= np.ones((100, 100))
>>> ima0 = cdl.obj.create_image("Example", data)
>>> ima1 = compute_add_noise(ima0)
```

**Parameters**

- **func** – 1 array → 1 array function
- **\*args** – Additional positional arguments to pass to the function
- **\*\*kwargs** – Additional keyword arguments to pass to the function

`cdl.computation.image.dst_11_signal(src: ImageObj, name: str, suffix: str | None = None) → SignalObj`

Create a result signal object, as returned by the callback function of the `cdl.core.gui.processor.base.BaseProcessor.compute_11()` method

**Parameters**

- **src** – input image object
- **name** – name of the processing function



**Returns**

Output signal object

`cdl.computation.image.compute_addition(dst: ImageObj, src: ImageObj) → ImageObj`

Add **dst** and **src** images and return **dst** image modified in place

**Parameters**

- **dst** – output image object
- **src** – input image object

**Returns**

Output image object (modified in place)

`cdl.computation.image.compute_product(dst: ImageObj, src: ImageObj) → ImageObj`

Multiply **dst** and **src** images and return **dst** image modified in place

**Parameters**

- **dst** – output image object
- **src** – input image object

**Returns**

Output image object (modified in place)

`cdl.computation.image.compute_addition_constant(src: ImageObj, p: ConstantParam) → ImageObj`

Add **dst** and a constant value and return the new result image object

**Parameters**

- **src** – input image object
- **p** – constant value

**Returns**

Result image object **src + p.value** (new object)

`cdl.computation.image.compute_difference_constant(src: ImageObj, p: ConstantParam) → ImageObj`

Subtract a constant value from an image and return the new result image object

**Parameters**

- **src** – input image object
- **p** – constant value

**Returns**

Result image object **src - p.value** (new object)

`cdl.computation.image.compute_product_constant(src: ImageObj, p: ConstantParam) → ImageObj`

Multiply **dst** by a constant value and return the new result image object

**Parameters**

- **src** – input image object
- **p** – constant value

**Returns**

Result image object **src \* p.value** (new object)

`cdl.computation.image.compute_division_constant(src: ImageObj, p: ConstantParam) → ImageObj`

Divide an image by a constant value and return the new result image object

**Parameters**

- **src** – input image object
- **p** – constant value

**Returns**

Result image object **src** / **p.value** (new object)

`cdl.computation.image.compute_arithmetic(src1: ImageObj, src2: ImageObj, p: ArithmeticParam) → ImageObj`

Compute arithmetic operation on two images

**Parameters**

- **src1** – input image object
- **src2** – input image object
- **p** – arithmetic parameters

**Returns**

Result image object

`cdl.computation.image.compute_difference(src1: ImageObj, src2: ImageObj) → ImageObj`

Compute difference between two images

**Parameters**

- **src1** – input image object
- **src2** – input image object

**Returns**

Result image object **src1** - **src2** (new object)

`cdl.computation.image.compute_quadratic_difference(src1: ImageObj, src2: ImageObj) → ImageObj`

Compute quadratic difference between two images

**Parameters**

- **src1** – input image object
- **src2** – input image object

**Returns**

Result image object (**src1** - **src2**) / sqrt(2.0) (new object)

`cdl.computation.image.compute_division(src1: ImageObj, src2: ImageObj) → ImageObj`

Compute division between two images

**Parameters**

- **src1** – input image object
- **src2** – input image object

**Returns**

Result image object **src1** / **src2** (new object)

`class cdl.computation.image.FlatFieldParam`

Flat-field parameters

**threshold**

Default: 0.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**classmethod create**(*threshold: float*) → *cdl.computation.image.FlatFieldParam*

Returns a new instance of *FlatFieldParam* with the fields set to the given values.

**Parameters**

**threshold** (*float*) – Default: 0.0.

**Returns**

New instance of *FlatFieldParam*.

`cdl.computation.image.compute_flatfield(src1: ImageObj, src2: ImageObj, p: FlatFieldParam) → ImageObj`

Compute flat field correction with *cdl.algorithms.image.flatfield()*

**Parameters**

- **src1** – raw data image object
- **src2** – flat field image object
- **p** – flat field parameters

**Returns**

Output image object

`cdl.computation.image.compute_normalize(src: ImageObj, p: NormalizeParam) → ImageObj`

Normalize image data depending on its maximum, with *cdl.algorithms.image.normalize()*

**Parameters**

**src** – input image object

**Returns**

Output image object

**class** `cdl.computation.image.LogP1Param`

Log10 parameters

**n**

Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`

**classmethod create**(*n: float*) → *cdl.computation.image.LogP1Param*

Returns a new instance of *LogP1Param* with the fields set to the given values.

**Parameters**

**n** (*float*) – Default: None.

**Returns**

New instance of *LogP1Param*.

`cdl.computation.image.compute_logp1(src: ImageObj, p: LogP1Param) → ImageObj`

Compute  $\log_{10}(z+n)$  with `numpy.log10`

**Parameters**

- **src** – input image object

- **p** – parameters

**Returns**

Output image object

**class** `cdl.computation.image.RotateParam`

Rotate parameters

**angle**

Angle (°). Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`

**mode**

Single choice from: 'constant', 'nearest', 'reflect', 'wrap'. Default: 'constant'.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**cval**

Value used for points outside the boundaries of the input if mode is 'constant'. Default: 0.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**reshape**

Reshape the output array so that the input array is contained completely in the output. Default: False.

**Type**

`guidata.dataset.dataitems.BoolItem`

**prefilter**

Default: True.

**Type**

`guidata.dataset.dataitems.BoolItem`

**order**

Spline interpolation order. Integer between 0 and 5. Default: 3.

**Type**

`guidata.dataset.dataitems.IntItem`

**classmethod** `create`(*angle: float, mode: str, cval: float, reshape: bool, prefilter: bool, order: int*) → `cdl.computation.image.RotateParam`

Returns a new instance of `RotateParam` with the fields set to the given values.

**Parameters**

- **angle** (*float*) – Angle (°). Default: None.
- **mode** (*str*) – Single choice from: 'constant', 'nearest', 'reflect', 'wrap'. Default: 'constant'.
- **cval** (*float*) – Value used for points outside the boundaries of the input if mode is 'constant'. Default: 0.0.
- **reshape** (*bool*) – Reshape the output array so that the input array is contained completely in the output. Default: False.
- **prefilter** (*bool*) – Default: True.
- **order** (*int*) – Spline interpolation order. Integer between 0 and 5. Default: 3.

**Returns**

New instance of *RotateParam*.

`cdl.computation.image.rotate_obj_coords(angle: float, obj: ImageObj, orig: ImageObj, coords: ndarray)`  
`→ None`

Apply rotation to coords associated to image obj

**Parameters**

- **angle** – rotation angle (in degrees)
- **obj** – image object
- **orig** – original image object
- **coords** – coordinates to rotate

**Returns**

Output data

`cdl.computation.image.rotate_obj_alpha(obj: ImageObj, orig: ImageObj, coords: ndarray, p: RotateParam)`  
`→ None`

Apply rotation to coords associated to image obj

`cdl.computation.image.compute_rotate(src: ImageObj, p: RotateParam) → ImageObj`

Rotate data with `scipy.ndimage.rotate()`

**Parameters**

- **src** – input image object
- **p** – parameters

**Returns**

Output image object

`cdl.computation.image.rotate_obj_90(dst: ImageObj, src: ImageObj, coords: ndarray) → None`

Apply rotation to coords associated to image obj

`cdl.computation.image.compute_rotate90(src: ImageObj) → ImageObj`

Rotate data 90° with `numpy.rot90()`

**Parameters**

**src** – input image object

**Returns**

Output image object

`cdl.computation.image.rotate_obj_270(dst: ImageObj, src: ImageObj, coords: ndarray) → None`

Apply rotation to coords associated to image obj

`cdl.computation.image.compute_rotate270(src: ImageObj) → ImageObj`

Rotate data 270° with `numpy.rot90()`

**Parameters**

**src** – input image object

**Returns**

Output image object

`cdl.computation.image.hflip_coords(dst: ImageObj, src: ImageObj, coords: ndarray) → None`

Apply HFlip to coords

`cdl.computation.image.compute_fliph(src: ImageObj) → ImageObj`

Flip data horizontally with `numpy.fliplr()`

**Parameters**

**src** – input image object

**Returns**

Output image object

`cdl.computation.image.vflip_coords(dst: ImageObj, src: ImageObj, coords: ndarray) → None`

Apply VFlip to coords

`cdl.computation.image.compute_flipv(src: ImageObj) → ImageObj`

Flip data vertically with `numpy.flipud()`

**Parameters**

**src** – input image object

**Returns**

Output image object

**class** `cdl.computation.image.GridParam`

Grid parameters

**direction**

Distribute over. Single choice from: ‘col’, ‘row’. Default: ‘col’.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**cols**

Columns. Integer, non zero. Default: 1.

**Type**

`guidata.dataset.dataitems.IntItem`

**rows**

Integer, non zero. Default: 1.

**Type**

`guidata.dataset.dataitems.IntItem`

**colspac**

Column spacing. Float higher than 0.0. Default: 0.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**rowspac**

Row spacing. Float higher than 0.0. Default: 0.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**classmethod** `create(direction: str, cols: int, rows: int, colspac: float, rowspac: float) → cdl.computation.image.GridParam`

Returns a new instance of `GridParam` with the fields set to the given values.

**Parameters**

- **direction** (*str*) – Distribute over. Single choice from: ‘col’, ‘row’. Default: ‘col’.

- **cols** (*int*) – Columns. Integer, non zero. Default: 1.
- **rows** (*int*) – Integer, non zero. Default: 1.
- **colspac** (*float*) – Column spacing. Float higher than 0.0. Default: 0.0.
- **rowspac** (*float*) – Row spacing. Float higher than 0.0. Default: 0.0.

**Returns**

New instance of *GridParam*.

**class** `cdl.computation.image.ResizeParam`

Resize parameters

**zoom**

Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`

**mode**

Single choice from: 'constant', 'nearest', 'reflect', 'wrap'. Default: 'constant'.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**cval**

Value used for points outside the boundaries of the input if mode is 'constant'. Default: 0.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**prefilter**

Default: True.

**Type**

`guidata.dataset.dataitems.BoolItem`

**order**

Spline interpolation order. Integer between 0 and 5. Default: 3.

**Type**

`guidata.dataset.dataitems.IntItem`

**classmethod** `create`(*zoom: float, mode: str, cval: float, prefilter: bool, order: int*) → *cdl.computation.image.ResizeParam*

Returns a new instance of *ResizeParam* with the fields set to the given values.

**Parameters**

- **zoom** (*float*) – Default: None.
- **mode** (*str*) – Single choice from: 'constant', 'nearest', 'reflect', 'wrap'. Default: 'constant'.
- **cval** (*float*) – Value used for points outside the boundaries of the input if mode is 'constant'. Default: 0.0.
- **prefilter** (*bool*) – Default: True.
- **order** (*int*) – Spline interpolation order. Integer between 0 and 5. Default: 3.

**Returns**

New instance of *ResizeParam*.

`cdl.computation.image.compute_resize(src: ImageObj, p: ResizeParam) → ImageObj`

Zooming function with `scipy.ndimage.zoom()`

**Parameters**

- **src** – input image object
- **p** – parameters

**Returns**

Output image object

**class** `cdl.computation.image.BinningParam`

Binning parameters

**sx**

Cluster size (X). Number of adjacent pixels to be combined together along x-axis. Integer higher than 2. Default: 2.

**Type**

`guidata.dataset.dataitems.IntItem`

**sy**

Cluster size (Y). Number of adjacent pixels to be combined together along y-axis. Integer higher than 2. Default: 2.

**Type**

`guidata.dataset.dataitems.IntItem`

**operation**

Single choice from: ‘sum’, ‘average’, ‘median’, ‘min’, ‘max’. Default: ‘sum’.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**dtype\_str**

Data type. Output image data type. Single choice from: ‘dtype’, ‘float32’, ‘float64’, ‘complex128’, ‘int32’, ‘int16’, ‘uint16’, ‘uint8’. Default: ‘dtype’.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**change\_pixel\_size**

Change pixel size so that overall image size remains the same. Default: False.

**Type**

`guidata.dataset.dataitems.BoolItem`

**classmethod** `create(sx: int, sy: int, operation: str, dtype_str: str, change_pixel_size: bool) → cdl.computation.image.BinningParam`

Returns a new instance of `BinningParam` with the fields set to the given values.

**Parameters**

- **sx** (*int*) – Cluster size (X). Number of adjacent pixels to be combined together along x-axis. Integer higher than 2. Default: 2.
- **sy** (*int*) – Cluster size (Y). Number of adjacent pixels to be combined together along y-axis. Integer higher than 2. Default: 2.
- **operation** (*str*) – Single choice from: ‘sum’, ‘average’, ‘median’, ‘min’, ‘max’. Default: ‘sum’.



- **dtype\_str** (*str*) – Data type. Output image data type. Single choice from: ‘dtype’, ‘float32’, ‘float64’, ‘complex128’, ‘int32’, ‘int16’, ‘uint16’, ‘uint8’. Default: ‘dtype’.
- **change\_pixel\_size** (*bool*) – Change pixel size so that overall image size remains the same. Default: False.

**Returns**

New instance of *BinningParam*.

`cdl.computation.image.compute_binning(src: ImageObj, param: BinningParam) → ImageObj`

Binning function on data with *cdl.algorithms.image.binning()*

**Parameters**

- **src** – input image object
- **param** – parameters

**Returns**

Output image object

`cdl.computation.image.extract_multiple_roi(src: ImageObj, group: DataSetGroup) → ImageObj`

Extract multiple regions of interest from data

**Parameters**

- **src** – input image object
- **group** – parameters defining the regions of interest

**Returns**

Output image object

`cdl.computation.image.extract_single_roi(src: ImageObj, p: ROI2DParam) → ImageObj`

Extract single ROI

**Parameters**

- **src** – input image object
- **p** – ROI parameters

**Returns**

Output image object

**class** `cdl.computation.image.LineProfileParam`

Horizontal or vertical profile parameters

**direction**

Single choice from: ‘horizontal’, ‘vertical’. Default: ‘horizontal’.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**row**

Integer higher than 0. Default: 0.

**Type**

`guidata.dataset.dataitems.IntItem`

**col**

Column. Integer higher than 0. Default: 0.

**Type**`guidata.dataset.dataitems.IntItem`**classmethod** `create(direction: str, row: int, col: int) → cdl.computation.image.LineProfileParam`Returns a new instance of `LineProfileParam` with the fields set to the given values.**Parameters**

- **direction** (`str`) – Single choice from: ‘horizontal’, ‘vertical’. Default: ‘horizontal’.
- **row** (`int`) – Integer higher than 0. Default: 0.
- **col** (`int`) – Column. Integer higher than 0. Default: 0.

**Returns**New instance of `LineProfileParam`.`cdl.computation.image.compute_line_profile(src: ImageObj, p: LineProfileParam) → SignalObj`

Compute horizontal or vertical profile

**Parameters**

- **src** – input image object
- **p** – parameters

**Returns**

Signal object with the profile

**class** `cdl.computation.image.SegmentProfileParam`

Segment profile parameters

**row1**

Start row. Integer higher than 0. Default: 0.

**Type**`guidata.dataset.dataitems.IntItem`**col1**

Start column. Integer higher than 0. Default: 0.

**Type**`guidata.dataset.dataitems.IntItem`**row2**

End row. Integer higher than 0. Default: 0.

**Type**`guidata.dataset.dataitems.IntItem`**col2**

End column. Integer higher than 0. Default: 0.

**Type**`guidata.dataset.dataitems.IntItem`**classmethod** `create(row1: int, col1: int, row2: int, col2: int) →  
cdl.computation.image.SegmentProfileParam`Returns a new instance of `SegmentProfileParam` with the fields set to the given values.**Parameters**

- **row1** (`int`) – Start row. Integer higher than 0. Default: 0.

- **col1** (*int*) – Start column. Integer higher than 0. Default: 0.
- **row2** (*int*) – End row. Integer higher than 0. Default: 0.
- **col2** (*int*) – End column. Integer higher than 0. Default: 0.

**Returns**

New instance of *SegmentProfileParam*.

`cdl.computation.image.compute_segment_profile(src: ImageObj, p: SegmentProfileParam) → SignalObj`

Compute segment profile

**Parameters**

- **src** – input image object
- **p** – parameters

**Returns**

Signal object with the segment profile

**class** `cdl.computation.image.AverageProfileParam`

Average horizontal or vertical profile parameters

**direction**

Single choice from: 'horizontal', 'vertical'. Default: 'horizontal'.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**row1**

Integer higher than 0. Default: 0.

**Type**

`guidata.dataset.dataitems.IntItem`

**row2**

Integer higher than -1. Default: -1.

**Type**

`guidata.dataset.dataitems.IntItem`

**col1**

Column 1. Integer higher than 0. Default: 0.

**Type**

`guidata.dataset.dataitems.IntItem`

**col2**

Column 2. Integer higher than -1. Default: -1.

**Type**

`guidata.dataset.dataitems.IntItem`

**classmethod** `create(direction: str, row1: int, row2: int, col1: int, col2: int) →`

`cdl.computation.image.AverageProfileParam`

Returns a new instance of *AverageProfileParam* with the fields set to the given values.

**Parameters**

- **direction** (*str*) – Single choice from: 'horizontal', 'vertical'. Default: 'horizontal'.
- **row1** (*int*) – Integer higher than 0. Default: 0.

- **row2** (*int*) – Integer higher than -1. Default: -1.
- **col1** (*int*) – Column 1. Integer higher than 0. Default: 0.
- **col2** (*int*) – Column 2. Integer higher than -1. Default: -1.

**Returns**

New instance of *AverageProfileParam*.

`cdl.computation.image.compute_average_profile(src: ImageObj, p: AverageProfileParam) → SignalObj`

Compute horizontal or vertical average profile

**Parameters**

- **src** – input image object
- **p** – parameters

**Returns**

Signal object with the average profile

**class** `cdl.computation.image.RadialProfileParam`

Radial profile parameters

**center**

Center position. Single choice from: 'centroid', 'center', 'user'. Default: 'centroid'.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**x0**

$X_{\text{Center}}$ . Float, unit: pixel. Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`

**y0**

$X_{\text{Center}}$ . Float, unit: pixel. Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`

**classmethod** `create(center: str, x0: float, y0: float) → cdl.computation.image.RadialProfileParam`

Returns a new instance of *RadialProfileParam* with the fields set to the given values.

**Parameters**

- **center** (*str*) – Center position. Single choice from: 'centroid', 'center', 'user'. Default: 'centroid'.
- **x0** (*float*) –  $X_{\text{Center}}$ . Float, unit: pixel. Default: None.
- **y0** (*float*) –  $X_{\text{Center}}$ . Float, unit: pixel. Default: None.

**Returns**

New instance of *RadialProfileParam*.

**update\_from\_image**(*obj*: ImageObj) → None

Update parameters from image

**choice\_callback**(*item*, *value*)

Callback for choice item

`cdl.computation.image.compute_radial_profile(src: ImageObj, p: RadialProfileParam) → SignalObj`

Compute radial profile around the centroid with `cdl.algorithms.image.get_radial_profile()`

**Parameters**

- **src** – input image object
- **p** – parameters

**Returns**

Signal object with the radial profile

`cdl.computation.image.compute_histogram(src: ImageObj, p: HistogramParam) → SignalObj`

Compute histogram of the image data, with `numpy.histogram()`

**Parameters**

- **src** – input image object
- **p** – parameters

**Returns**

Signal object with the histogram

`cdl.computation.image.compute_swap_axes(src: ImageObj) → ImageObj`

Swap image axes with `numpy.transpose()`

**Parameters**

**src** – input image object

**Returns**

Output image object

`cdl.computation.image.compute_inverse(src: ImageObj) → ImageObj`

Compute the inverse of an image and return the new result image object

**Parameters**

**src** – input image object

**Returns**

Result image object `1 / src` (new object)

`cdl.computation.image.compute_abs(src: ImageObj) → ImageObj`

Compute absolute value with `numpy.absolute`

**Parameters**

**src** – input image object

**Returns**

Output image object

`cdl.computation.image.compute_re(src: ImageObj) → ImageObj`

Compute real part with `numpy.real()`

**Parameters**

**src** – input image object

**Returns**

Output image object

`cdl.computation.image.compute_im(src: ImageObj) → ImageObj`

Compute imaginary part with `numpy.imag()`

**Parameters**

**src** – input image object

**Returns**

Output image object

**class** `cdl.computation.image.DataTypeIParam`

Convert image data type parameters

**dtype\_str**

Destination data type. Output image data type. Single choice from: 'float32', 'float64', 'complex128', 'int32', 'int16', 'uint16', 'uint8'. Default: 'float32'.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**classmethod** `create(dtype_str: str) → cdl.computation.image.DataTypeIParam`

Returns a new instance of `DataTypeIParam` with the fields set to the given values.

**Parameters**

**dtype\_str** (*str*) – Destination data type. Output image data type. Single choice from: 'float32', 'float64', 'complex128', 'int32', 'int16', 'uint16', 'uint8'. Default: 'float32'.

**Returns**

New instance of `DataTypeIParam`.

`cdl.computation.image.compute_astype(src: ImageObj, p: DataTypeIParam) → ImageObj`

Convert image data type with `cdl.algorithms.datatypes.clip_astype()`

**Parameters**

- **src** – input image object
- **p** – parameters

**Returns**

Output image object

`cdl.computation.image.compute_log10(src: ImageObj) → ImageObj`

Compute log10 with `numpy.log10`

**Parameters**

**src** – input image object

**Returns**

Output image object

`cdl.computation.image.compute_exp(src: ImageObj) → ImageObj`

Compute exponential with `numpy.exp`

**Parameters**

**src** – input image object

**Returns**

Output image object

**class** `cdl.computation.image.ZCalibrateParam`

Image linear calibration parameters

**a**

Default: 1.0.

**Type**`guidata.dataset.dataitems.FloatItem`**b**

Default: 0.0.

**Type**`guidata.dataset.dataitems.FloatItem`**classmethod** **create**(*a: float, b: float*) → *cdl.computation.image.ZCalibrateParam*Returns a new instance of *ZCalibrateParam* with the fields set to the given values.**Parameters**

- **a** (*float*) – Default: 1.0.
- **b** (*float*) – Default: 0.0.

**Returns**New instance of *ZCalibrateParam*.`cdl.computation.image.compute_calibration(src: ImageObj, p: ZCalibrateParam) → ImageObj`

Compute linear calibration

**Parameters**

- **src** – input image object
- **p** – calibration parameters

**Returns**

Output image object

`cdl.computation.image.compute_clip(src: ImageObj, p: ClipParam) → ImageObj`Apply clipping with `numpy.clip()`**Parameters**

- **src** – input image object
- **p** – parameters

**Returns**

Output image object

`cdl.computation.image.compute_offset_correction(src: ImageObj, p: ROI2DParam) → ImageObj`

Apply offset correction

**Parameters**

- **src** – input image object
- **p** – parameters

**Returns**

Output image object

`cdl.computation.image.compute_gaussian_filter(src: ImageObj, p: GaussianParam) → ImageObj`Compute gaussian filter with `scipy.ndimage.gaussian_filter()`**Parameters**

- **src** – input image object
- **p** – parameters

**Returns**

Output image object

`cdl.computation.image.compute_moving_average(src: ImageObj, p: MovingAverageParam) → ImageObj`

Compute moving average with `scipy.ndimage.uniform_filter()`

**Parameters**

- **src** – input image object
- **p** – parameters

**Returns**

Output image object

`cdl.computation.image.compute_moving_median(src: ImageObj, p: MovingMedianParam) → ImageObj`

Compute moving median with `scipy.ndimage.median_filter()`

**Parameters**

- **src** – input image object
- **p** – parameters

**Returns**

Output image object

`cdl.computation.image.compute_wiener(src: ImageObj) → ImageObj`

Compute Wiener filter with `scipy.signal.wiener()`

**Parameters**

**src** – input image object

**Returns**

Output image object

**class** `cdl.computation.image.ZeroPadding2DParam`

Zero padding parameters for 2D images

**strategy**

Padding strategy. Single choice from: 'next\_pow2', 'multiple\_of\_64', 'custom'. Default: 'custom'.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**rows**

Rows to add. Integer higher than 0. Default: None.

**Type**

`guidata.dataset.dataitems.IntItem`

**cols**

Columns to add. Integer higher than 0. Default: None.

**Type**

`guidata.dataset.dataitems.IntItem`

**position**

Padding position. Single choice from: 'bottom-right', 'center'. Default: 'bottom-right'.

**Type**

`guidata.dataset.dataitems.ChoiceItem`



**classmethod** `create(strategy: str, rows: int, cols: int, position: str) →`  
`cdl.computation.image.ZeroPadding2DParam`

Returns a new instance of `ZeroPadding2DParam` with the fields set to the given values.

#### Parameters

- **strategy** (*str*) – Padding strategy. Single choice from: ‘next\_pow2’, ‘multiple\_of\_64’, ‘custom’. Default: ‘custom’.
- **rows** (*int*) – Rows to add. Integer higher than 0. Default: None.
- **cols** (*int*) – Columns to add. Integer higher than 0. Default: None.
- **position** (*str*) – Padding position. Single choice from: ‘bottom-right’, ‘center’. Default: ‘bottom-right’.

#### Returns

New instance of `ZeroPadding2DParam`.

**update\_from\_image**(*obj: ImageObj*) → None

Update parameters from image

**choice\_callback**(*item, value*)

Callback to update padding values

`cdl.computation.image.compute_zero_padding(src: ImageObj, p: ZeroPadding2DParam) → ImageObj`

Compute zero padding for an image using `cdl.algorithms.image.zero_padding`.

#### Parameters

- **src** – source image object
- **p** – parameters

#### Returns

New padded image object

`cdl.computation.image.compute_fft(src: ImageObj, p: FFTParam | None = None) → ImageObj`

Compute FFT with `cdl.algorithms.image.fft2d()`

#### Parameters

- **src** – input image object
- **p** – parameters

#### Returns

Output image object

`cdl.computation.image.compute_ift(src: ImageObj, p: FFTParam | None = None) → ImageObj`

Compute inverse FFT with `cdl.algorithms.image.ift2d()`

#### Parameters

- **src** – input image object
- **p** – parameters

#### Returns

Output image object

`cdl.computation.image.compute_magnitude_spectrum(src: ImageObj, p: SpectrumParam | None = None)`  
`→ ImageObj`

Compute magnitude spectrum with `cdl.algorithms.image.magnitude_spectrum()`

**Parameters**

- **src** – input image object
- **p** – parameters

**Returns**

Output image object

`cdl.computation.image.compute_phase_spectrum(src: ImageObj) → ImageObj`

Compute phase spectrum with `cdl.algorithms.image.phase_spectrum()`

**Parameters**

**src** – input image object

**Returns**

Output image object

`cdl.computation.image.compute_psd(src: ImageObj, p: SpectrumParam | None = None) → ImageObj`

Compute power spectral density with `cdl.algorithms.image.psd()`

**Parameters**

- **src** – input image object
- **p** – parameters

**Returns**

Output image object

**class** `cdl.computation.image.ButterworthParam`

Butterworth filter parameters

**cut\_off**

Cut-off frequency ratio. Cut-off frequency ratio. Float between 0.0 and 0.5. Default: 0.005.

**Type**

`guidata.dataset.dataitems.FloatItem`

**high\_pass**

If true, apply high-pass filter instead of low-pass. Default: False.

**Type**

`guidata.dataset.dataitems.BoolItem`

**order**

Order of the butterworth filter. Integer higher than 1. Default: 2.

**Type**

`guidata.dataset.dataitems.IntItem`

**classmethod** `create(cut_off: float, high_pass: bool, order: int) →`

`cdl.computation.image.ButterworthParam`

Returns a new instance of `ButterworthParam` with the fields set to the given values.

**Parameters**

- **cut\_off** (*float*) – Cut-off frequency ratio. Cut-off frequency ratio. Float between 0.0 and 0.5. Default: 0.005.
- **high\_pass** (*bool*) – If true, apply high-pass filter instead of low-pass. Default: False.
- **order** (*int*) – Order of the butterworth filter. Integer higher than 1. Default: 2.

**Returns**

New instance of *ButterworthParam*.

`cdl.computation.image.compute_butterworth(src: ImageObj, p: ButterworthParam) → ImageObj`

Compute Butterworth filter with `skimage.filters.butterworth()`

**Parameters**

- **src** – input image object
- **p** – parameters

**Returns**

Output image object

`cdl.computation.image.calc_resultshape(title: str, shape: Literal['rectangle', 'circle', 'ellipse', 'segment', 'marker', 'point', 'polygon'], obj: ImageObj, func: Callable, *args: Any, add_label: bool = False) → ResultShape | None`

Calculate result shape by executing a computation function on an image object, taking into account the image origin (x0, y0), scale (dx, dy) and ROIs.

**Parameters**

- **title** – result title
- **shape** – result shape kind
- **obj** – input image object
- **func** – computation function
- **\*args** – computation function arguments
- **add\_label** – if True, add a label item (and the geometrical shape) to plot (default to False)

**Returns**

Result shape object or None if no result is found

**Warning:** The computation function must take either a single argument (the data) or multiple arguments (the data followed by the computation parameters).

Moreover, the computation function must return a single value or a NumPy array containing the result of the computation. This array contains the coordinates of points, polygons, circles or ellipses in the form `[[x, y], ...]`, or `[[x0, y0, x1, y1, ...], ...]`, or `[[x0, y0, r], ...]`, or `[[x0, y0, a, b, theta], ...]`.

`cdl.computation.image.get_centroid_coords(data: ndarray) → ndarray`

Return centroid coordinates with `cdl.algorithms.image.get_centroid_fourier()`

**Parameters**

**data** – input data

**Returns**

Centroid coordinates

`cdl.computation.image.compute_centroid(image: ImageObj) → ResultShape | None`

Compute centroid with `cdl.algorithms.image.get_centroid_fourier()`

**Parameters**

**image** – input image

**Returns**

Centroid coordinates

`cdl.computation.image.get_enclosing_circle_coords(data: ndarray) → ndarray`

Return diameter coords for the circle contour enclosing image values above threshold (FWHM)

**Parameters**

**data** – input data

**Returns**

Diameter coords

`cdl.computation.image.compute_enclosing_circle(image: ImageObj) → ResultShape | None`

Compute minimum enclosing circle with `cdl.algorithms.image.get_enclosing_circle()`

**Parameters**

**image** – input image

**Returns**

Diameter coords

**class** `cdl.computation.image.HoughCircleParam`

Circle Hough transform parameters

**min\_radius**

Radius<sub>min</sub>. Integer higher than 0, non zero, unit: pixels. Default: None.

**Type**

`guidata.dataset.dataitems.IntItem`

**max\_radius**

Radius<sub>max</sub>. Integer higher than 0, non zero, unit: pixels. Default: None.

**Type**

`guidata.dataset.dataitems.IntItem`

**min\_distance**

Minimal distance. Integer higher than 0. Default: None.

**Type**

`guidata.dataset.dataitems.IntItem`

**classmethod** `create(min_radius: int, max_radius: int, min_distance: int) → cdl.computation.image.HoughCircleParam`

Returns a new instance of `HoughCircleParam` with the fields set to the given values.

**Parameters**

- **min\_radius** (*int*) – Radius<sub>min</sub>. Integer higher than 0, non zero, unit: pixels. Default: None.
- **max\_radius** (*int*) – Radius<sub>max</sub>. Integer higher than 0, non zero, unit: pixels. Default: None.
- **min\_distance** (*int*) – Minimal distance. Integer higher than 0. Default: None.

**Returns**

New instance of `HoughCircleParam`.

`cdl.computation.image.compute_hough_circle_peaks(image: ImageObj, p: HoughCircleParam) → ResultShape | None`

Compute Hough circles with `cdl.algorithms.image.get_hough_circle_peaks()`

**Parameters**

- **image** – input image
- **p** – parameters

**Returns**

Circle coordinates

`cdl.computation.image.compute_stats(obj: ImageObj) → ResultProperties`

Compute statistics on an image

**Parameters**

**obj** – input image object

**Returns**

Result properties

**Threshold features****Threshold computation module**

**class** `cdl.computation.image.threshold.ThresholdParam`

Histogram threshold parameters

**method**

Threshold method. Single choice from: 'manual', 'isodata', 'li', 'mean', 'minimum', 'otsu', 'triangle', 'yen'. Default: 'manual'.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**bins**

Number of bins. Integer higher than 1. Default: 256.

**Type**

`guidata.dataset.dataitems.IntItem`

**value**

Threshold value. Default: 0.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**operation**

Single choice from: '>', '<'. Default: '>'.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**classmethod** `create(method: str, bins: int, value: float, operation: str) → cdl.computation.image.threshold.ThresholdParam`

Returns a new instance of *ThresholdParam* with the fields set to the given values.

**Parameters**

- **method** (*str*) – Threshold method. Single choice from: 'manual', 'isodata', 'li', 'mean', 'minimum', 'otsu', 'triangle', 'yen'. Default: 'manual'.
- **bins** (*int*) – Number of bins. Integer higher than 1. Default: 256.
- **value** (*float*) – Threshold value. Default: 0.0.

- **operation** (*str*) – Single choice from: '>', '<'. Default: '>'.

**Returns**

New instance of *ThresholdParam*.

`cdl.computation.image.threshold.compute_threshold(src: ImageObj, p: ThresholdParam) → ImageObj`

Compute the threshold, using one of the available algorithms:

- Manual: a fixed threshold value
- ISODATA: `skimage.filters.threshold_isodata()`
- Li: `skimage.filters.threshold_li()`
- Mean: `skimage.filters.threshold_mean()`
- Minimum: `skimage.filters.threshold_minimum()`
- Otsu: `skimage.filters.threshold_otsu()`
- Triangle: `skimage.filters.threshold_triangle()`
- Yen: `skimage.filters.threshold_yen()`

**Parameters**

- **src** – input image object
- **p** – parameters

**Returns**

Output image object

`cdl.computation.image.threshold.compute_threshold_isodata(src: ImageObj) → ImageObj`

Compute the threshold using the Isodata algorithm with default parameters, see `skimage.filters.threshold_isodata()`

**Parameters**

**src** – input image object

**Returns**

Output image object

`cdl.computation.image.threshold.compute_threshold_li(src: ImageObj) → ImageObj`

Compute the threshold using the Li algorithm with default parameters, see `skimage.filters.threshold_li()`

**Parameters**

**src** – input image object

**Returns**

Output image object

`cdl.computation.image.threshold.compute_threshold_mean(src: ImageObj) → ImageObj`

Compute the threshold using the Mean algorithm, see `skimage.filters.threshold_mean()`

**Parameters**

**src** – input image object

**Returns**

Output image object

`cdl.computation.image.threshold.compute_threshold_minimum(src: ImageObj) → ImageObj`  
 Compute the threshold using the Minimum algorithm with default parameters, see `skimage.filters.threshold_minimum()`

**Parameters**

**src** – input image object

**Returns**

Output image object

`cdl.computation.image.threshold.compute_threshold_otsu(src: ImageObj) → ImageObj`  
 Compute the threshold using the Otsu algorithm with default parameters, see `skimage.filters.threshold_otsu()`

**Parameters**

**src** – input image object

**Returns**

Output image object

`cdl.computation.image.threshold.compute_threshold_triangle(src: ImageObj) → ImageObj`  
 Compute the threshold using the Triangle algorithm with default parameters, see `skimage.filters.threshold_triangle()`

**Parameters**

**src** – input image object

**Returns**

Output image object

`cdl.computation.image.threshold.compute_threshold_yen(src: ImageObj) → ImageObj`  
 Compute the threshold using the Yen algorithm with default parameters, see `skimage.filters.threshold_yen()`

**Parameters**

**src** – input image object

**Returns**

Output image object

## Exposure correction features

### Exposure computation module

**class** `cdl.computation.image.exposure.AdjustGammaParam`

Gamma adjustment parameters

**gamma**

Gamma correction factor (higher values give more contrast). Float higher than 0.0. Default: 1.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**gain**

Gain factor (higher values give more contrast). Float higher than 0.0. Default: 1.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**classmethod** `create(gamma: float, gain: float) → cdl.computation.image.exposure.AdjustGammaParam`

Returns a new instance of `AdjustGammaParam` with the fields set to the given values.

**Parameters**

- **gamma** (*float*) – Gamma correction factor (higher values give more contrast). Float higher than 0.0. Default: 1.0.
- **gain** (*float*) – Gain factor (higher values give more contrast). Float higher than 0.0. Default: 1.0.

**Returns**

New instance of `AdjustGammaParam`.

`cdl.computation.image.exposure.compute_adjust_gamma(src: ImageObj, p: AdjustGammaParam) → ImageObj`

Gamma correction with `skimage.exposure.adjust_gamma()`

**Parameters**

- **src** – input image object
- **p** – parameters

**Returns**

Output image object

**class** `cdl.computation.image.exposure.AdjustLogParam`

Logarithmic adjustment parameters

**gain**

Gain factor (higher values give more contrast). Float higher than 0.0. Default: 1.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**inv**

If true, apply inverse logarithmic transformation. Default: False.

**Type**

`guidata.dataset.dataitems.BoolItem`

**classmethod** `create(gain: float, inv: bool) → cdl.computation.image.exposure.AdjustLogParam`

Returns a new instance of `AdjustLogParam` with the fields set to the given values.

**Parameters**

- **gain** (*float*) – Gain factor (higher values give more contrast). Float higher than 0.0. Default: 1.0.
- **inv** (*bool*) – If true, apply inverse logarithmic transformation. Default: False.

**Returns**

New instance of `AdjustLogParam`.

`cdl.computation.image.exposure.compute_adjust_log(src: ImageObj, p: AdjustLogParam) → ImageObj`

Compute log correction with `skimage.exposure.adjust_log()`

**Parameters**

- **src** – input image object
- **p** – parameters



**Returns**

Output image object

**class** `cdl.computation.image.exposure.AdjustSigmoidParam`

Sigmoid adjustment parameters

**cutoff**

Cutoff value (higher values give more contrast). Float between 0.0 and 1.0. Default: 0.5.

**Type**

`guidata.dataset.dataitems.FloatItem`

**gain**

Gain factor (higher values give more contrast). Float higher than 0.0. Default: 10.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**inv**

If true, apply inverse sigmoid transformation. Default: False.

**Type**

`guidata.dataset.dataitems.BoolItem`

**classmethod** `create(cutoff: float, gain: float, inv: bool) →`

`cdl.computation.image.exposure.AdjustSigmoidParam`

Returns a new instance of `AdjustSigmoidParam` with the fields set to the given values.

**Parameters**

- **cutoff** (*float*) – Cutoff value (higher values give more contrast). Float between 0.0 and 1.0. Default: 0.5.
- **gain** (*float*) – Gain factor (higher values give more contrast). Float higher than 0.0. Default: 10.0.
- **inv** (*bool*) – If true, apply inverse sigmoid transformation. Default: False.

**Returns**

New instance of `AdjustSigmoidParam`.

`cdl.computation.image.exposure.compute_adjust_sigmoid(src: ImageObj, p: AdjustSigmoidParam) →`  
`ImageObj`

Compute sigmoid correction with `skimage.exposure.adjust_sigmoid()`

**Parameters**

- **src** – input image object
- **p** – parameters

**Returns**

Output image object

**class** `cdl.computation.image.exposure.RescaleIntensityParam`

Intensity rescaling parameters

**in\_range**

Input range. Min and max intensity values of input image ('image' refers to input image min/max levels, 'dtype' refers to input image data type range). Single choice from: 'image', 'dtype', 'float32', 'float64', 'complex128', 'int32', 'int16', 'uint16', 'uint8'. Default: 'image'.

**Type**`guidata.dataset.dataitems.ChoiceItem`**out\_range**

Output range. Min and max intensity values of output image ('image' refers to input image min/max levels, 'dtype' refers to input image data type range).. Single choice from: 'image', 'dtype', 'float32', 'float64', 'complex128', 'int32', 'int16', 'uint16', 'uint8'. Default: 'dtype'.

**Type**`guidata.dataset.dataitems.ChoiceItem`

**classmethod** **create**(*in\_range: str, out\_range: str*) →

`cdl.computation.image.exposure.RescaleIntensityParam`

Returns a new instance of `RescaleIntensityParam` with the fields set to the given values.

**Parameters**

- **in\_range** (*str*) – Input range. Min and max intensity values of input image ('image' refers to input image min/max levels, 'dtype' refers to input image data type range). Single choice from: 'image', 'dtype', 'float32', 'float64', 'complex128', 'int32', 'int16', 'uint16', 'uint8'. Default: 'image'.
- **out\_range** (*str*) – Output range. Min and max intensity values of output image ('image' refers to input image min/max levels, 'dtype' refers to input image data type range).. Single choice from: 'image', 'dtype', 'float32', 'float64', 'complex128', 'int32', 'int16', 'uint16', 'uint8'. Default: 'dtype'.

**Returns**

New instance of `RescaleIntensityParam`.

`cdl.computation.image.exposure.compute_rescale_intensity`(*src: ImageObj, p: RescaleIntensityParam*) → *ImageObj*

Rescale image intensity levels with `skimage.exposure.rescale_intensity()`

**Parameters**

- **src** – input image object
- **p** – parameters

**Returns**

Output image object

**class** `cdl.computation.image.exposure.EqualizeHistParam`

Histogram equalization parameters

**nbins**

Number of bins. Number of bins for image histogram. Integer higher than 1. Default: 256.

**Type**`guidata.dataset.dataitems.IntItem`

**classmethod** **create**(*nbins: int*) → `cdl.computation.image.exposure.EqualizeHistParam`

Returns a new instance of `EqualizeHistParam` with the fields set to the given values.

**Parameters**

**nbins** (*int*) – Number of bins. Number of bins for image histogram. Integer higher than 1. Default: 256.

**Returns**

New instance of `EqualizeHistParam`.

`cdl.computation.image.exposure.compute_equalize_hist(src: ImageObj, p: EqualizeHistParam) → ImageObj`

Histogram equalization with `skimage.exposure.equalize_hist()`

#### Parameters

- **src** – input image object
- **p** – parameters

#### Returns

Output image object

**class** `cdl.computation.image.exposure.EqualizeAdaptHistParam`

Adaptive histogram equalization parameters

#### **nbins**

Number of bins. Number of bins for image histogram. Integer higher than 1. Default: 256.

#### Type

`guidata.dataset.dataitems.IntItem`

#### **clip\_limit**

Clipping limit. Clipping limit (higher values give more contrast). Float between 0.0 and 1.0. Default: 0.01.

#### Type

`guidata.dataset.dataitems.FloatItem`

**classmethod** `create(nbins: int, clip_limit: float) →`

`cdl.computation.image.exposure.EqualizeAdaptHistParam`

Returns a new instance of `EqualizeAdaptHistParam` with the fields set to the given values.

#### Parameters

- **nbins** (*int*) – Number of bins. Number of bins for image histogram. Integer higher than 1. Default: 256.
- **clip\_limit** (*float*) – Clipping limit. Clipping limit (higher values give more contrast). Float between 0.0 and 1.0. Default: 0.01.

#### Returns

New instance of `EqualizeAdaptHistParam`.

`cdl.computation.image.exposure.compute_equalize_adapthist(src: ImageObj, p: EqualizeAdaptHistParam) → ImageObj`

Adaptive histogram equalization with `skimage.exposure.equalize_adapthist()`

#### Parameters

- **src** – input image object
- **p** – parameters

#### Returns

Output image object

## Restoration features

### Restoration computation module

**class** `cdl.computation.image.restoration.DenoiseTVParam`

Total Variation denoising parameters

**weight**

Denoising weight. The greater weight, the more denoising (at the expense of fidelity to input). Float higher than 0, non zero. Default: 0.1.

**Type**

`guidata.dataset.dataitems.FloatItem`

**eps**

Epsilon. Relative difference of the value of the cost function that determines the stop criterion. The algorithm stops when:  $(e_{(n-1)} - e_n) < \text{eps} * e_0$ . Float higher than 0, non zero. Default: 0.0002.

**Type**

`guidata.dataset.dataitems.FloatItem`

**max\_num\_iter**

Max. iterations. Maximal number of iterations used for the optimization. Integer higher than 0, non zero. Default: 200.

**Type**

`guidata.dataset.dataitems.IntItem`

**classmethod** `create(weight: float, eps: float, max_num_iter: int) →`  
`cdl.computation.image.restoration.DenoiseTVParam`

Returns a new instance of `DenoiseTVParam` with the fields set to the given values.

**Parameters**

- **weight** (*float*) – Denoising weight. The greater weight, the more denoising (at the expense of fidelity to input). Float higher than 0, non zero. Default: 0.1.
- **eps** (*float*) – Epsilon. Relative difference of the value of the cost function that determines the stop criterion. The algorithm stops when:  $(e_{(n-1)} - e_n) < \text{eps} * e_0$ . Float higher than 0, non zero. Default: 0.0002.
- **max\_num\_iter** (*int*) – Max. iterations. Maximal number of iterations used for the optimization. Integer higher than 0, non zero. Default: 200.

**Returns**

New instance of `DenoiseTVParam`.

`cdl.computation.image.restoration.compute_denoise_tv(src: ImageObj, p: DenoiseTVParam) →`  
`ImageObj`

Compute Total Variation denoising with `skimage.restoration.denoise_tv_chambolle()`

**Parameters**

- **src** – input image object
- **p** – parameters

**Returns**

Output image object

**class** `cdl.computation.image.restoration.DenoiseBilateralParam`

Bilateral filter denoising parameters

**sigma\_spatial**

spatial. Standard deviation for range distance. A larger value results in averaging of pixels with larger spatial differences. Float higher than 0, non zero, unit: pixels. Default: 1.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**mode**

Single choice from: 'constant', 'edge', 'symmetric', 'reflect', 'wrap'. Default: 'constant'.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**cval**

Used in conjunction with mode 'constant', the value outside the image boundaries. Default: 0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**classmethod** `create(sigma_spatial: float, mode: str, cval: float) →`

`cdl.computation.image.restoration.DenoiseBilateralParam`

Returns a new instance of `DenoiseBilateralParam` with the fields set to the given values.

**Parameters**

- **sigma\_spatial** (*float*) – spatial. Standard deviation for range distance. A larger value results in averaging of pixels with larger spatial differences. Float higher than 0, non zero, unit: pixels. Default: 1.0.
- **mode** (*str*) – Single choice from: 'constant', 'edge', 'symmetric', 'reflect', 'wrap'. Default: 'constant'.
- **cval** (*float*) – Used in conjunction with mode 'constant', the value outside the image boundaries. Default: 0.

**Returns**

New instance of `DenoiseBilateralParam`.

`cdl.computation.image.restoration.compute_denoise_bilateral(src: ImageObj, p:`

`DenoiseBilateralParam) → ImageObj`

Compute bilateral filter denoising with `skimage.restoration.denoise_bilateral()`

**Parameters**

- **src** – input image object
- **p** – parameters

**Returns**

Output image object

**class** `cdl.computation.image.restoration.DenoiseWaveletParam`

Wavelet denoising parameters

**wavelet**

Single choice from: 'bior1.1', 'bior1.3', 'bior1.5', 'bior2.2', 'bior2.4', 'bior2.6', 'bior2.8', 'bior3.1', 'bior3.3', 'bior3.5', 'bior3.7', 'bior3.9', 'bior4.4', 'bior5.5', 'bior6.8', 'cgau1', 'cgau2', 'cgau3', 'cgau4', 'cgau5', 'cgau6', 'cgau7', 'cgau8', 'cmor', 'coif1', 'coif2', 'coif3', 'coif4', 'coif5', 'coif6', 'coif7', 'coif8',

'coif9', 'coif10', 'coif11', 'coif12', 'coif13', 'coif14', 'coif15', 'coif16', 'coif17', 'db1', 'db2', 'db3', 'db4', 'db5', 'db6', 'db7', 'db8', 'db9', 'db10', 'db11', 'db12', 'db13', 'db14', 'db15', 'db16', 'db17', 'db18', 'db19', 'db20', 'db21', 'db22', 'db23', 'db24', 'db25', 'db26', 'db27', 'db28', 'db29', 'db30', 'db31', 'db32', 'db33', 'db34', 'db35', 'db36', 'db37', 'db38', 'dmey', 'fbasp', 'gaus1', 'gaus2', 'gaus3', 'gaus4', 'gaus5', 'gaus6', 'gaus7', 'gaus8', 'haar', 'mexh', 'morl', 'rbio1.1', 'rbio1.3', 'rbio1.5', 'rbio2.2', 'rbio2.4', 'rbio2.6', 'rbio2.8', 'rbio3.1', 'rbio3.3', 'rbio3.5', 'rbio3.7', 'rbio3.9', 'rbio4.4', 'rbio5.5', 'rbio6.8', 'shan', 'sym2', 'sym3', 'sym4', 'sym5', 'sym6', 'sym7', 'sym8', 'sym9', 'sym10', 'sym11', 'sym12', 'sym13', 'sym14', 'sym15', 'sym16', 'sym17', 'sym18', 'sym19', 'sym20'. Default: 'sym9'.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**mode**

Single choice from: 'soft', 'hard'. Default: 'soft'.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**method**

Single choice from: 'BayesShrink', 'VisuShrink'. Default: 'VisuShrink'.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**classmethod** `create(wavelet: str, mode: str, method: str) →`

`cdl.computation.image.restoration.DenoiseWaveletParam`

Returns a new instance of `DenoiseWaveletParam` with the fields set to the given values.

**Parameters**

- **wavelet** (*str*) – Single choice from: 'bior1.1', 'bior1.3', 'bior1.5', 'bior2.2', 'bior2.4', 'bior2.6', 'bior2.8', 'bior3.1', 'bior3.3', 'bior3.5', 'bior3.7', 'bior3.9', 'bior4.4', 'bior5.5', 'bior6.8', 'cgau1', 'cgau2', 'cgau3', 'cgau4', 'cgau5', 'cgau6', 'cgau7', 'cgau8', 'cmor', 'coif1', 'coif2', 'coif3', 'coif4', 'coif5', 'coif6', 'coif7', 'coif8', 'coif9', 'coif10', 'coif11', 'coif12', 'coif13', 'coif14', 'coif15', 'coif16', 'coif17', 'db1', 'db2', 'db3', 'db4', 'db5', 'db6', 'db7', 'db8', 'db9', 'db10', 'db11', 'db12', 'db13', 'db14', 'db15', 'db16', 'db17', 'db18', 'db19', 'db20', 'db21', 'db22', 'db23', 'db24', 'db25', 'db26', 'db27', 'db28', 'db29', 'db30', 'db31', 'db32', 'db33', 'db34', 'db35', 'db36', 'db37', 'db38', 'dmey', 'fbasp', 'gaus1', 'gaus2', 'gaus3', 'gaus4', 'gaus5', 'gaus6', 'gaus7', 'gaus8', 'haar', 'mexh', 'morl', 'rbio1.1', 'rbio1.3', 'rbio1.5', 'rbio2.2', 'rbio2.4', 'rbio2.6', 'rbio2.8', 'rbio3.1', 'rbio3.3', 'rbio3.5', 'rbio3.7', 'rbio3.9', 'rbio4.4', 'rbio5.5', 'rbio6.8', 'shan', 'sym2', 'sym3', 'sym4', 'sym5', 'sym6', 'sym7', 'sym8', 'sym9', 'sym10', 'sym11', 'sym12', 'sym13', 'sym14', 'sym15', 'sym16', 'sym17', 'sym18', 'sym19', 'sym20'. Default: 'sym9'.
- **mode** (*str*) – Single choice from: 'soft', 'hard'. Default: 'soft'.
- **method** (*str*) – Single choice from: 'BayesShrink', 'VisuShrink'. Default: 'VisuShrink'.

**Returns**

New instance of `DenoiseWaveletParam`.

`cdl.computation.image.restoration.compute_denoise_wavelet(src: ImageObj, p: DenoiseWaveletParam) → ImageObj`

Compute Wavelet denoising with `skimage.restoration.denoise_wavelet()`

**Parameters**

- **src** – input image object

- **p** – parameters

**Returns**

Output image object

`cdl.computation.image.restoration.compute_denoise_tophat(src: ImageObj, p: MorphologyParam) → ImageObj`

Denoise using White Top-Hat with `skimage.morphology.white_tophat()`

**Parameters**

- **src** – input image object
- **p** – parameters

**Returns**

Output image object

**Morphological features****Morphology computation module**

`class cdl.computation.image.morphology.MorphologyParam`

White Top-Hat parameters

**radius**

Footprint (disk) radius. Integer higher than 1. Default: 1.

**Type**`guidata.dataset.dataitems.IntItem`

`classmethod create(radius: int) → cdl.computation.image.morphology.MorphologyParam`

Returns a new instance of `MorphologyParam` with the fields set to the given values.**Parameters**

**radius** (`int`) – Footprint (disk) radius. Integer higher than 1. Default: 1.

**Returns**New instance of `MorphologyParam`.

`cdl.computation.image.morphology.compute_white_tophat(src: ImageObj, p: MorphologyParam) → ImageObj`

Compute White Top-Hat with `skimage.morphology.white_tophat()`

**Parameters**

- **src** – input image object
- **p** – parameters

**Returns**

Output image object

`cdl.computation.image.morphology.compute_black_tophat(src: ImageObj, p: MorphologyParam) → ImageObj`

Compute Black Top-Hat with `skimage.morphology.black_tophat()`

**Parameters**

- **src** – input image object

- **p** – parameters

**Returns**

Output image object

`cdl.computation.image.morphology.compute_erosion(src: ImageObj, p: MorphologyParam) → ImageObj`  
Compute Erosion with `skimage.morphology.erosion()`

**Parameters**

- **src** – input image object
- **p** – parameters

**Returns**

Output image object

`cdl.computation.image.morphology.compute_dilation(src: ImageObj, p: MorphologyParam) → ImageObj`

Compute Dilation with `skimage.morphology.dilation()`

**Parameters**

- **src** – input image object
- **p** – parameters

**Returns**

Output image object

`cdl.computation.image.morphology.compute_opening(src: ImageObj, p: MorphologyParam) → ImageObj`  
Compute morphological opening with `skimage.morphology.opening()`

**Parameters**

- **src** – input image object
- **p** – parameters

**Returns**

Output image object

`cdl.computation.image.morphology.compute_closing(src: ImageObj, p: MorphologyParam) → ImageObj`  
Compute morphological closing with `skimage.morphology.closing()`

**Parameters**

- **src** – input image object
- **p** – parameters

**Returns**

Output image object



## Edge detection features

### Edges computation module

**class** `cdl.computation.image.edges.CannyParam`

Canny filter parameters

**sigma**

Standard deviation of the gaussian filter. Float higher than 0, non zero, unit: pixels. Default: 1.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**low\_threshold**

Lower bound for hysteresis thresholding (linking edges). Float higher than 0. Default: 0.1.

**Type**

`guidata.dataset.dataitems.FloatItem`

**high\_threshold**

Upper bound for hysteresis thresholding (linking edges). Float higher than 0. Default: 0.9.

**Type**

`guidata.dataset.dataitems.FloatItem`

**use\_quantiles**

If true then treat `low_threshold` and `high_threshold` as quantiles of the edge magnitude image, rather than absolute edge magnitude values. If true then the thresholds must be in the range [0, 1]. Default: True.

**Type**

`guidata.dataset.dataitems.BoolItem`

**mode**

Single choice from: 'reflect', 'constant', 'nearest', 'mirror', 'wrap'. Default: 'constant'.

**Type**

`guidata.dataset.dataitems.ChoiceItem`

**cval**

Value to fill past edges of input if mode is constant. Default: 0.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**classmethod** `create(sigma: float, low_threshold: float, high_threshold: float, use_quantiles: bool, mode: str, cval: float) → cdl.computation.image.edges.CannyParam`

Returns a new instance of `CannyParam` with the fields set to the given values.

**Parameters**

- **sigma** (*float*) – Standard deviation of the gaussian filter. Float higher than 0, non zero, unit: pixels. Default: 1.0.
- **low\_threshold** (*float*) – Lower bound for hysteresis thresholding (linking edges). Float higher than 0. Default: 0.1.
- **high\_threshold** (*float*) – Upper bound for hysteresis thresholding (linking edges). Float higher than 0. Default: 0.9.

- **use\_quantiles** (*bool*) – If true then treat `low_threshold` and `high_threshold` as quantiles of the edge magnitude image, rather than absolute edge magnitude values. If true then the thresholds must be in the range [0, 1]. Default: True.
- **mode** (*str*) – Single choice from: 'reflect', 'constant', 'nearest', 'mirror', 'wrap'. Default: 'constant'.
- **cval** (*float*) – Value to fill past edges of input if mode is constant. Default: 0.0.

**Returns**

New instance of *CannyParam*.

`cdl.computation.image.edges.compute_canny(src: ImageObj, p: CannyParam) → ImageObj`

Compute Canny filter with `skimage.feature.canny()`

**Parameters**

- **src** – input image object
- **p** – parameters

**Returns**

Output image object

`cdl.computation.image.edges.compute_roberts(src: ImageObj) → ImageObj`

Compute Roberts filter with `skimage.filters.roberts()`

**Parameters**

**src** – input image object

**Returns**

Output image object

`cdl.computation.image.edges.compute_prewitt(src: ImageObj) → ImageObj`

Compute Prewitt filter with `skimage.filters.prewitt()`

**Parameters**

**src** – input image object

**Returns**

Output image object

`cdl.computation.image.edges.compute_prewitt_h(src: ImageObj) → ImageObj`

Compute horizontal Prewitt filter with `skimage.filters.prewitt_h()`

**Parameters**

**src** – input image object

**Returns**

Output image object

`cdl.computation.image.edges.compute_prewitt_v(src: ImageObj) → ImageObj`

Compute vertical Prewitt filter with `skimage.filters.prewitt_v()`

**Parameters**

**src** – input image object

**Returns**

Output image object

`cdl.computation.image.edges.compute_sobel(src: ImageObj) → ImageObj`

Compute Sobel filter with `skimage.filters.sobel()`

**Parameters****src** – input image object**Returns**

Output image object

`cdl.computation.image.edges.compute_sobel_h(src: ImageObj) → ImageObj`Compute horizontal Sobel filter with `skimage.filters.sobel_h()`**Parameters****src** – input image object**Returns**

Output image object

`cdl.computation.image.edges.compute_sobel_v(src: ImageObj) → ImageObj`Compute vertical Sobel filter with `skimage.filters.sobel_v()`**Parameters****src** – input image object**Returns**

Output image object

`cdl.computation.image.edges.compute_scharr(src: ImageObj) → ImageObj`Compute Scharr filter with `skimage.filters.scharr()`**Parameters****src** – input image object**Returns**

Output image object

`cdl.computation.image.edges.compute_scharr_h(src: ImageObj) → ImageObj`Compute horizontal Scharr filter with `skimage.filters.scharr_h()`**Parameters****src** – input image object**Returns**

Output image object

`cdl.computation.image.edges.compute_scharr_v(src: ImageObj) → ImageObj`Compute vertical Scharr filter with `skimage.filters.scharr_v()`**Parameters****src** – input image object**Returns**

Output image object

`cdl.computation.image.edges.compute_farid(src: ImageObj) → ImageObj`Compute Farid filter with `skimage.filters.farid()`**Parameters****src** – input image object**Returns**

Output image object

`cdl.computation.image.edges.compute_farid_h(src: ImageObj) → ImageObj`

Compute horizontal Farid filter with `skimage.filters.farid_h()`

**Parameters**

**src** – input image object

**Returns**

Output image object

`cdl.computation.image.edges.compute_farid_v(src: ImageObj) → ImageObj`

Compute vertical Farid filter with `skimage.filters.farid_v()`

**Parameters**

**src** – input image object

**Returns**

Output image object

`cdl.computation.image.edges.compute_laplace(src: ImageObj) → ImageObj`

Compute Laplace filter with `skimage.filters.laplace()`

**Parameters**

**src** – input image object

**Returns**

Output image object

## Detection features

### Blob detection computation module

**class** `cdl.computation.image.detection.GenericDetectionParam`

Generic detection parameters

**threshold**

Relative threshold. Detection threshold, relative to difference between data maximum and minimum. Float between 0.1 and 0.9. Default: 0.5.

**Type**

`guidata.dataset.dataitems.FloatItem`

**classmethod** `create(threshold: float) → cdl.computation.image.detection.GenericDetectionParam`

Returns a new instance of `GenericDetectionParam` with the fields set to the given values.

**Parameters**

**threshold** (*float*) – Relative threshold. Detection threshold, relative to difference between data maximum and minimum. Float between 0.1 and 0.9. Default: 0.5.

**Returns**

New instance of `GenericDetectionParam`.

**class** `cdl.computation.image.detection.Peak2DDetectionParam`

Peak detection parameters

**threshold**

Relative threshold. Detection threshold, relative to difference between data maximum and minimum. Float between 0.1 and 0.9. Default: 0.5.

**Type**`guidata.dataset.dataitems.FloatItem`**size**

Neighborhoods size. Size of the sliding window used in maximum/minimum filtering algorithm. Integer higher than 1, unit: pixels. Default: 10.

**Type**`guidata.dataset.dataitems.IntItem`**create\_rois**

Default: True.

**Type**`guidata.dataset.dataitems.BoolItem`

**classmethod** **create**(*threshold: float, size: int, create\_rois: bool*) →  
`cdl.computation.image.detection.Peak2DDetectionParam`

Returns a new instance of `Peak2DDetectionParam` with the fields set to the given values.

**Parameters**

- **threshold** (*float*) – Relative threshold. Detection threshold, relative to difference between data maximum and minimum. Float between 0.1 and 0.9. Default: 0.5.
- **size** (*int*) – Neighborhoods size. Size of the sliding window used in maximum/minimum filtering algorithm. Integer higher than 1, unit: pixels. Default: 10.
- **create\_rois** (*bool*) – Default: True.

**Returns**

New instance of `Peak2DDetectionParam`.

`cdl.computation.image.detection.compute_peak_detection`(*image: ImageObj, p: Peak2DDetectionParam*) → *ResultShape* | None

Compute 2D peak detection with `cdl.algorithms.image.get_2d_peaks_coords()`

**Parameters**

- **imageOutput** – input image
- **p** – parameters

**Returns**

Peak coordinates

**class** `cdl.computation.image.detection.ContourShapeParam`

Contour shape parameters

**threshold**

Relative threshold. Detection threshold, relative to difference between data maximum and minimum. Float between 0.1 and 0.9. Default: 0.5.

**Type**`guidata.dataset.dataitems.FloatItem`**shape**

Single choice from: 'ellipse', 'circle', 'polygon'. Default: 'ellipse'.

**Type**`guidata.dataset.dataitems.ChoiceItem`

**classmethod** `create(threshold: float, shape: str) → cdl.computation.image.detection.ContourShapeParam`

Returns a new instance of `ContourShapeParam` with the fields set to the given values.

#### Parameters

- **threshold** (*float*) – Relative threshold. Detection threshold, relative to difference between data maximum and minimum. Float between 0.1 and 0.9. Default: 0.5.
- **shape** (*str*) – Single choice from: ‘ellipse’, ‘circle’, ‘polygon’. Default: ‘ellipse’.

#### Returns

New instance of `ContourShapeParam`.

`cdl.computation.image.detection.compute_contour_shape(image: ImageObj, p: ContourShapeParam) → ResultShape | None`

Compute contour shape fit with `cdl.algorithms.image.get_contour_shapes()`

**class** `cdl.computation.image.detection.BaseBlobParam`

Base class for blob detection parameters

#### **min\_sigma**

min. The minimum standard deviation for gaussian kernel. Keep this low to detect smaller blobs. Float higher than 0, non zero, unit: pixels. Default: 1.0.

#### Type

`guidata.dataset.dataitems.FloatItem`

#### **max\_sigma**

max. The maximum standard deviation for gaussian kernel. Keep this high to detect larger blobs. Float higher than 0, non zero, unit: pixels. Default: 30.0.

#### Type

`guidata.dataset.dataitems.FloatItem`

#### **threshold\_rel**

Relative threshold. Minimum intensity of blobs. Float between 0.0 and 1.0. Default: 0.2.

#### Type

`guidata.dataset.dataitems.FloatItem`

#### **overlap**

If two blobs overlap by a fraction greater than this value, the smaller blob is eliminated. Float between 0.0 and 1.0. Default: 0.5.

#### Type

`guidata.dataset.dataitems.FloatItem`

**classmethod** `create(min_sigma: float, max_sigma: float, threshold_rel: float, overlap: float) → cdl.computation.image.detection.BaseBlobParam`

Returns a new instance of `BaseBlobParam` with the fields set to the given values.

#### Parameters

- **min\_sigma** (*float*) – min. The minimum standard deviation for gaussian kernel. Keep this low to detect smaller blobs. Float higher than 0, non zero, unit: pixels. Default: 1.0.
- **max\_sigma** (*float*) – max. The maximum standard deviation for gaussian kernel. Keep this high to detect larger blobs. Float higher than 0, non zero, unit: pixels. Default: 30.0.
- **threshold\_rel** (*float*) – Relative threshold. Minimum intensity of blobs. Float between 0.0 and 1.0. Default: 0.2.

- **overlap** (*float*) – If two blobs overlap by a fraction greater than this value, the smaller blob is eliminated. Float between 0.0 and 1.0. Default: 0.5.

**Returns**

New instance of *BaseBlobParam*.

**class** `cdl.computation.image.detection.BlobDOGParam`

Blob detection using Difference of Gaussian method

**min\_sigma**

min. The minimum standard deviation for gaussian kernel. Keep this low to detect smaller blobs. Float higher than 0, non zero, unit: pixels. Default: 1.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**max\_sigma**

max. The maximum standard deviation for gaussian kernel. Keep this high to detect larger blobs. Float higher than 0, non zero, unit: pixels. Default: 30.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**threshold\_rel**

Relative threshold. Minimum intensity of blobs. Float between 0.0 and 1.0. Default: 0.2.

**Type**

`guidata.dataset.dataitems.FloatItem`

**overlap**

If two blobs overlap by a fraction greater than this value, the smaller blob is eliminated. Float between 0.0 and 1.0. Default: 0.5.

**Type**

`guidata.dataset.dataitems.FloatItem`

**exclude\_border**

If true, exclude blobs from the border of the image. Default: True.

**Type**

`guidata.dataset.dataitems.BoolItem`

**classmethod** `create(min_sigma: float, max_sigma: float, threshold_rel: float, overlap: float, exclude_border: bool) → cdl.computation.image.detection.BlobDOGParam`

Returns a new instance of *BlobDOGParam* with the fields set to the given values.

**Parameters**

- **min\_sigma** (*float*) – min. The minimum standard deviation for gaussian kernel. Keep this low to detect smaller blobs. Float higher than 0, non zero, unit: pixels. Default: 1.0.
- **max\_sigma** (*float*) – max. The maximum standard deviation for gaussian kernel. Keep this high to detect larger blobs. Float higher than 0, non zero, unit: pixels. Default: 30.0.
- **threshold\_rel** (*float*) – Relative threshold. Minimum intensity of blobs. Float between 0.0 and 1.0. Default: 0.2.
- **overlap** (*float*) – If two blobs overlap by a fraction greater than this value, the smaller blob is eliminated. Float between 0.0 and 1.0. Default: 0.5.
- **exclude\_border** (*bool*) – If true, exclude blobs from the border of the image. Default: True.

**Returns**

New instance of *BlobDOGParam*.

`cdl.computation.image.detection.compute_blob_dog(image: ImageObj, p: BlobDOGParam) → ResultShape | None`

Compute blobs using Difference of Gaussian method with *cdl.algorithms.image.find\_blobs\_dog()*

**Parameters**

- **imageOutput** – input image
- **p** – parameters

**Returns**

Blobs coordinates

**class** `cdl.computation.image.detection.BlobDOHParam`

Blob detection using Determinant of Hessian method

**min\_sigma**

min. The minimum standard deviation for gaussian kernel. Keep this low to detect smaller blobs. Float higher than 0, non zero, unit: pixels. Default: 1.0.

**Type**

*guidata.dataset.dataitems.FloatItem*

**max\_sigma**

max. The maximum standard deviation for gaussian kernel. Keep this high to detect larger blobs. Float higher than 0, non zero, unit: pixels. Default: 30.0.

**Type**

*guidata.dataset.dataitems.FloatItem*

**threshold\_rel**

Relative threshold. Minimum intensity of blobs. Float between 0.0 and 1.0. Default: 0.2.

**Type**

*guidata.dataset.dataitems.FloatItem*

**overlap**

If two blobs overlap by a fraction greater than this value, the smaller blob is eliminated. Float between 0.0 and 1.0. Default: 0.5.

**Type**

*guidata.dataset.dataitems.FloatItem*

**log\_scale**

If set intermediate values of standard deviations are interpolated using a logarithmic scale to the base 10. If not, linear interpolation is used. Default: False.

**Type**

*guidata.dataset.dataitems.BoolItem*

**classmethod** `create(min_sigma: float, max_sigma: float, threshold_rel: float, overlap: float, log_scale: bool) → cdl.computation.image.detection.BlobDOHParam`

Returns a new instance of *BlobDOHParam* with the fields set to the given values.

**Parameters**

- **min\_sigma** (*float*) – min. The minimum standard deviation for gaussian kernel. Keep this low to detect smaller blobs. Float higher than 0, non zero, unit: pixels. Default: 1.0.



- **max\_sigma** (*float*) – <sub>max</sub>. The maximum standard deviation for gaussian kernel. Keep this high to detect larger blobs. Float higher than 0, non zero, unit: pixels. Default: 30.0.
- **threshold\_rel** (*float*) – Relative threshold. Minimum intensity of blobs. Float between 0.0 and 1.0. Default: 0.2.
- **overlap** (*float*) – If two blobs overlap by a fraction greater than this value, the smaller blob is eliminated. Float between 0.0 and 1.0. Default: 0.5.
- **log\_scale** (*bool*) – If set intermediate values of standard deviations are interpolated using a logarithmic scale to the base 10. If not, linear interpolation is used. Default: False.

**Returns**

New instance of *BlobDOHParam*.

`cdl.computation.image.detection.compute_blob_doh(image: ImageObj, p: BlobDOHParam) → ResultShape | None`

Compute blobs using Determinant of Hessian method with `cdl.algorithms.image.find_blobs_doh()`

**Parameters**

- **imageOutput** – input image
- **p** – parameters

**Returns**

Blobs coordinates

**class** `cdl.computation.image.detection.BlobLOGParam`

Blob detection using Laplacian of Gaussian method

**min\_sigma**

<sub>min</sub>. The minimum standard deviation for gaussian kernel. Keep this low to detect smaller blobs. Float higher than 0, non zero, unit: pixels. Default: 1.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**max\_sigma**

<sub>max</sub>. The maximum standard deviation for gaussian kernel. Keep this high to detect larger blobs. Float higher than 0, non zero, unit: pixels. Default: 30.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**threshold\_rel**

Relative threshold. Minimum intensity of blobs. Float between 0.0 and 1.0. Default: 0.2.

**Type**

`guidata.dataset.dataitems.FloatItem`

**overlap**

If two blobs overlap by a fraction greater than this value, the smaller blob is eliminated. Float between 0.0 and 1.0. Default: 0.5.

**Type**

`guidata.dataset.dataitems.FloatItem`

**log\_scale**

If set intermediate values of standard deviations are interpolated using a logarithmic scale to the base 10. If not, linear interpolation is used. Default: False.

**Type**`guidata.dataset.dataitems.BoolItem`**exclude\_border**

If true, exclude blobs from the border of the image. Default: True.

**Type**`guidata.dataset.dataitems.BoolItem`

**classmethod** **create**(*min\_sigma: float, max\_sigma: float, threshold\_rel: float, overlap: float, log\_scale: bool, exclude\_border: bool*) → `cdl.computation.image.detection.BlobLOGParam`

Returns a new instance of `BlobLOGParam` with the fields set to the given values.

**Parameters**

- **min\_sigma** (*float*) – min. The minimum standard deviation for gaussian kernel. Keep this low to detect smaller blobs. Float higher than 0, non zero, unit: pixels. Default: 1.0.
- **max\_sigma** (*float*) – max. The maximum standard deviation for gaussian kernel. Keep this high to detect larger blobs. Float higher than 0, non zero, unit: pixels. Default: 30.0.
- **threshold\_rel** (*float*) – Relative threshold. Minimum intensity of blobs. Float between 0.0 and 1.0. Default: 0.2.
- **overlap** (*float*) – If two blobs overlap by a fraction greater than this value, the smaller blob is eliminated. Float between 0.0 and 1.0. Default: 0.5.
- **log\_scale** (*bool*) – If set intermediate values of standard deviations are interpolated using a logarithmic scale to the base 10. If not, linear interpolation is used. Default: False.
- **exclude\_border** (*bool*) – If true, exclude blobs from the border of the image. Default: True.

**Returns**

New instance of `BlobLOGParam`.

`cdl.computation.image.detection.compute_blob_log`(*image: ImageObj, p: BlobLOGParam*) → `ResultShape` | None

Compute blobs using Laplacian of Gaussian method with `cdl.algorithms.image.find_blobs_log()`

**Parameters**

- **imageOutput** – input image
- **p** – parameters

**Returns**

Blobs coordinates

**class** `cdl.computation.image.detection.BlobOpenCVParam`

Blob detection using OpenCV

**min\_threshold**

Min. threshold. The minimum threshold between local maxima and minima. This parameter does not affect the quality of the blobs, only the quantity. Lower thresholds result in larger numbers of blobs. Float higher than 0.0. Default: 10.0.

**Type**`guidata.dataset.dataitems.FloatItem`

**max\_threshold**

Max. threshold. The maximum threshold between local maxima and minima. This parameter does not affect the quality of the blobs, only the quantity. Lower thresholds result in larger numbers of blobs. Float higher than 0.0. Default: 200.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**min\_repeatability**

Min. repeatability. The minimum number of times a blob needs to be detected in a sequence of images to be considered valid. Integer higher than 1. Default: 2.

**Type**

`guidata.dataset.dataitems.IntItem`

**min\_dist\_between\_blobs**

Min. distance between blobs. The minimum distance between two blobs. If blobs are found closer together than this distance, the smaller blob is removed. Float higher than 0.0, non zero. Default: 10.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**filter\_by\_color**

If true, the image is filtered by color instead of intensity. Default: True.

**Type**

`guidata.dataset.dataitems.BoolItem`

**blob\_color**

The color of the blobs to detect (0 for dark blobs, 255 for light blobs). Default: 0.

**Type**

`guidata.dataset.dataitems.IntItem`

**filter\_by\_area**

If true, the image is filtered by blob area. Default: True.

**Type**

`guidata.dataset.dataitems.BoolItem`

**min\_area**

Min. area. The minimum blob area. Float higher than 0.0. Default: 25.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**max\_area**

Max. area. The maximum blob area. Float higher than 0.0. Default: 500.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**filter\_by\_circularity**

If true, the image is filtered by blob circularity. Default: False.

**Type**

`guidata.dataset.dataitems.BoolItem`

**min\_circularity**

Min. circularity. The minimum circularity of the blobs. Float between 0.0 and 1.0. Default: 0.8.

**Type**

`guidata.dataset.dataitems.FloatItem`

**max\_circularity**

Max. circularity. The maximum circularity of the blobs. Float between 0.0 and 1.0. Default: 1.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**filter\_by\_inertia**

If true, the image is filtered by blob inertia. Default: False.

**Type**

`guidata.dataset.dataitems.BoolItem`

**min\_inertia\_ratio**

Min. inertia ratio. The minimum inertia ratio of the blobs. Float between 0.0 and 1.0. Default: 0.6.

**Type**

`guidata.dataset.dataitems.FloatItem`

**max\_inertia\_ratio**

Max. inertia ratio. The maximum inertia ratio of the blobs. Float between 0.0 and 1.0. Default: 1.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**filter\_by\_convexity**

If true, the image is filtered by blob convexity. Default: False.

**Type**

`guidata.dataset.dataitems.BoolItem`

**min\_convexity**

Min. convexity. The minimum convexity of the blobs. Float between 0.0 and 1.0. Default: 0.8.

**Type**

`guidata.dataset.dataitems.FloatItem`

**max\_convexity**

Max. convexity. The maximum convexity of the blobs. Float between 0.0 and 1.0. Default: 1.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**classmethod create**(*min\_threshold: float, max\_threshold: float, min\_repeatability: int, min\_dist\_between\_blobs: float, filter\_by\_color: bool, blob\_color: int, filter\_by\_area: bool, min\_area: float, max\_area: float, filter\_by\_circularity: bool, min\_circularity: float, max\_circularity: float, filter\_by\_inertia: bool, min\_inertia\_ratio: float, max\_inertia\_ratio: float, filter\_by\_convexity: bool, min\_convexity: float, max\_convexity: float*) → `cdl.computation.image.detection.BlobOpenCVParam`

Returns a new instance of `BlobOpenCVParam` with the fields set to the given values.

**Parameters**

- **min\_threshold** (*float*) – Min. threshold. The minimum threshold between local maxima and minima. This parameter does not affect the quality of the blobs, only the quantity. Lower thresholds result in larger numbers of blobs. Float higher than 0.0. Default: 10.0.

- **max\_threshold** (*float*) – Max. threshold. The maximum threshold between local maxima and minima. This parameter does not affect the quality of the blobs, only the quantity. Lower thresholds result in larger numbers of blobs. Float higher than 0.0. Default: 200.0.
- **min\_repeatability** (*int*) – Min. repeatability. The minimum number of times a blob needs to be detected in a sequence of images to be considered valid. Integer higher than 1. Default: 2.
- **min\_dist\_between\_blobs** (*float*) – Min. distance between blobs. The minimum distance between two blobs. If blobs are found closer together than this distance, the smaller blob is removed. Float higher than 0.0, non zero. Default: 10.0.
- **filter\_by\_color** (*bool*) – If true, the image is filtered by color instead of intensity. Default: True.
- **blob\_color** (*int*) – The color of the blobs to detect (0 for dark blobs, 255 for light blobs). Default: 0.
- **filter\_by\_area** (*bool*) – If true, the image is filtered by blob area. Default: True.
- **min\_area** (*float*) – Min. area. The minimum blob area. Float higher than 0.0. Default: 25.0.
- **max\_area** (*float*) – Max. area. The maximum blob area. Float higher than 0.0. Default: 500.0.
- **filter\_by\_circularity** (*bool*) – If true, the image is filtered by blob circularity. Default: False.
- **min\_circularity** (*float*) – Min. circularity. The minimum circularity of the blobs. Float between 0.0 and 1.0. Default: 0.8.
- **max\_circularity** (*float*) – Max. circularity. The maximum circularity of the blobs. Float between 0.0 and 1.0. Default: 1.0.
- **filter\_by\_inertia** (*bool*) – If true, the image is filtered by blob inertia. Default: False.
- **min\_inertia\_ratio** (*float*) – Min. inertia ratio. The minimum inertia ratio of the blobs. Float between 0.0 and 1.0. Default: 0.6.
- **max\_inertia\_ratio** (*float*) – Max. inertia ratio. The maximum inertia ratio of the blobs. Float between 0.0 and 1.0. Default: 1.0.
- **filter\_by\_convexity** (*bool*) – If true, the image is filtered by blob convexity. Default: False.
- **min\_convexity** (*float*) – Min. convexity. The minimum convexity of the blobs. Float between 0.0 and 1.0. Default: 0.8.
- **max\_convexity** (*float*) – Max. convexity. The maximum convexity of the blobs. Float between 0.0 and 1.0. Default: 1.0.

#### Returns

New instance of *BlobOpenCVParam*.

`cdl.computation.image.detection.compute_blob_opencv(image: ImageObj, p: BlobOpenCVParam) → ResultShape | None`

Compute blobs using OpenCV with `cdl.algorithms.image.find_blobs_opencv()`

#### Parameters

- **imageOutput** – input image
- **p** – parameters

**Returns**

Blobs coordinates

## 3.5 Proxy objects (`cdl.proxy`)

The `cdl.proxy` module provides a way to access DataLab features from a proxy class.

The list of compute methods accessible from the proxy objects is available in the *Calling processor methods using proxy objects* section.

### 3.5.1 Remote proxy

The remote proxy is used when DataLab is started from a different process than the proxy. In this case, the proxy connects to DataLab XML-RPC server.

**class** `cdl.proxy.RemoteProxy`(*autoconnect*: *bool* = *True*)

DataLab remote proxy class.

This class provides access to DataLab features from a proxy class. This is the remote version of proxy, which is used when DataLab is started from a different process than the proxy.

**Parameters**

**autoconnect** – Automatically connect to DataLab XML-RPC server.

**Raises**

- **ConnectionRefusedError** – Unable to connect to DataLab
- **ValueError** – Invalid timeout (must be  $\geq 0.0$ )
- **ValueError** – Invalid number of retries (must be  $\geq 1$ )

---

**Note:** The proxy object also allows to access DataLab computing methods exposed by the processor classes (see *Calling processor methods using proxy objects*).

---

### Examples

Here is a simple example of how to use RemoteProxy in a Python script or in a Jupyter notebook:

```
>>> from cdl.proxy import RemoteProxy
>>> proxy = RemoteProxy()
Connecting to DataLab XML-RPC server...OK (port: 28867)
>>> proxy.get_version()
'1.0.0'
>>> proxy.add_signal("toto", np.array([1., 2., 3.]), np.array([4., 5., -1.]))
True
>>> proxy.get_object_titles()
['toto']
>>> proxy["toto"] # from title
<cdl.core.model.signal.SignalObj at 0x7f7f1c0b4a90>
>>> proxy[1] # from number
<cdl.core.model.signal.SignalObj at 0x7f7f1c0b4a90>
```

(continues on next page)

(continued from previous page)

```
>>> proxy[1].data
array([1., 2., 3.])
>>> proxy.set_current_panel("image")
```

**add\_annotations\_from\_items**(items: *list*, refresh\_plot: *bool* = *True*, panel: *str* | *None* = *None*) → *None*

Add object annotations (annotation plot items).

#### Parameters

- **items** – annotation plot items
- **refresh\_plot** – refresh plot. Defaults to *True*.
- **panel** – panel name (valid values: “signal”, “image”). If *None*, current panel is used.

**add\_group**(title: *str*, panel: *str* | *None* = *None*, select: *bool* = *False*) → *None*

Add group to DataLab.

#### Parameters

- **title** – Group title
- **panel** – Panel name (valid values: “signal”, “image”). Defaults to *None*.
- **select** – Select the group after creation. Defaults to *False*.

**add\_image**(title: *str*, data: *ndarray*, xunit: *str* | *None* = *None*, yunit: *str* | *None* = *None*, zunit: *str* | *None* = *None*, xlabel: *str* | *None* = *None*, ylabel: *str* | *None* = *None*, zlabel: *str* | *None* = *None*, group\_id: *str* = “”, set\_current: *bool* = *True*) → *bool*

Add image data to DataLab.

#### Parameters

- **title** – Image title
- **data** – Image data
- **xunit** – X unit. Defaults to *None*
- **yunit** – Y unit. Defaults to *None*
- **zunit** – Z unit. Defaults to *None*
- **xlabel** – X label. Defaults to *None*
- **ylabel** – Y label. Defaults to *None*
- **zlabel** – Z label. Defaults to *None*
- **group\_id** – group id in which to add the image. Defaults to “”
- **set\_current** – if *True*, set the added image as current

#### Returns

*True* if image was added successfully, *False* otherwise

#### Raises

**ValueError** – Invalid data dtype

**add\_label\_with\_title**(title: *str* | *None* = *None*, panel: *str* | *None* = *None*) → *None*

Add a label with object title on the associated plot

#### Parameters

- **title** – Label title. Defaults to *None*. If *None*, the title is the object title.

- **panel** – panel name (valid values: “signal”, “image”). If None, current panel is used.

**add\_object**(*obj*: *SignalObj* | *ImageObj*, *group\_id*: *str* = "", *set\_current*: *bool* = *True*) → *None*

Add object to DataLab.

#### Parameters

- **obj** – Signal or image object
- **group\_id** – group id in which to add the object. Defaults to “”
- **set\_current** – if True, set the added object as current

**add\_signal**(*title*: *str*, *xdata*: *ndarray*, *ydata*: *ndarray*, *xunit*: *str* | *None* = *None*, *yunit*: *str* | *None* = *None*, *xlabel*: *str* | *None* = *None*, *ylabel*: *str* | *None* = *None*, *group\_id*: *str* = "", *set\_current*: *bool* = *True*) → *bool*

Add signal data to DataLab.

#### Parameters

- **title** – Signal title
- **xdata** – X data
- **ydata** – Y data
- **xunit** – X unit. Defaults to None
- **yunit** – Y unit. Defaults to None
- **xlabel** – X label. Defaults to None
- **ylabel** – Y label. Defaults to None
- **group\_id** – group id in which to add the signal. Defaults to “”
- **set\_current** – if True, set the added signal as current

#### Returns

True if signal was added successfully, False otherwise

#### Raises

- **ValueError** – Invalid xdata dtype
- **ValueError** – Invalid ydata dtype

**calc**(*name*: *str*, *param*: *DataSet* | *None* = *None*) → *None*

Call compute function name in current panel’s processor.

#### Parameters

- **name** – Compute function name
- **param** – Compute function parameter. Defaults to None.

#### Raises

**ValueError** – unknown function

**close\_application**() → *None*

Close DataLab application

**connect**(*port*: *str* | *None* = *None*, *timeout*: *float* | *None* = *None*, *retries*: *int* | *None* = *None*) → *None*

Try to connect to DataLab XML-RPC server.

#### Parameters



- **port** – XML-RPC port to connect to. If not specified, the port is automatically retrieved from DataLab configuration.
- **timeout** – Timeout in seconds. Defaults to 5.0.
- **retries** – Number of retries. Defaults to 10.

**Raises**

- **ConnectionRefusedError** – Unable to connect to DataLab
- **ValueError** – Invalid timeout (must be  $\geq 0.0$ )
- **ValueError** – Invalid number of retries (must be  $\geq 1$ )

**context\_no\_refresh()** → `Generator[None, None, None]`

Return a context manager to temporarily disable auto refresh.

**Returns**

Context manager

**Example**

```
>>> with proxy.context_no_refresh():
...     proxy.add_image("image1", data1)
...     proxy.compute_fft()
...     proxy.compute_wiener()
...     proxy.compute_ifft()
...     # Auto refresh is disabled during the above operations
```

**delete\_metadata(refresh\_plot: bool = True, keep\_roi: bool = False)** → `None`

Delete metadata of selected objects

**Parameters**

- **refresh\_plot** – Refresh plot. Defaults to True.
- **keep\_roi** – Keep ROI. Defaults to False.

**disconnect()** → `None`

Disconnect from DataLab XML-RPC server.

**get\_current\_panel()** → `str`

Return current panel name.

**Returns**

“signal”, “image”, “macro”))

**Return type**

Panel name (valid values

**get\_group\_titles\_with\_object\_infos()** → `tuple[list[str], list[list[str]], list[list[str]]]`

Return groups titles and lists of inner objects uuids and titles.

**Returns**

groups titles, lists of inner objects uuids and titles

**Return type**

Tuple

**get\_method\_list()** → list[str]

Return list of available methods.

**get\_object**(nb\_id\_title: int | str | None = None, panel: str | None = None) → SignalObj | ImageObj

Get object (signal/image) from index.

**Parameters**

- **nb\_id\_title** – Object number, or object id, or object title. Defaults to None (current object).
- **panel** – Panel name. Defaults to None (current panel).

**Returns**

Object

**Raises**

**KeyError** – if object not found

**get\_object\_shapes**(nb\_id\_title: int | str | None = None, panel: str | None = None) → list

Get plot item shapes associated to object (signal/image).

**Parameters**

- **nb\_id\_title** – Object number, or object id, or object title. Defaults to None (current object).
- **panel** – Panel name. Defaults to None (current panel).

**Returns**

List of plot item shapes

**get\_object\_titles**(panel: str | None = None) → list[str]

Get object (signal/image) list for current panel. Objects are sorted by group number and object index in group.

**Parameters**

**panel** – panel name (valid values: “signal”, “image”, “macro”). If None, current data panel is used (i.e. signal or image panel).

**Returns**

List of object titles

**Raises**

**ValueError** – if panel not found

**get\_object\_uuids**(panel: str | None = None, group: int | str | None = None) → list[str]

Get object (signal/image) uuid list for current panel. Objects are sorted by group number and object index in group.

**Parameters**

- **panel** – panel name (valid values: “signal”, “image”). If None, current panel is used.
- **group** – Group number, or group id, or group title. Defaults to None (all groups).

**Returns**

List of object uuids

**Raises**

**ValueError** – if panel not found

**classmethod** `get_public_methods()` → `list[str]`

Return all public methods of the class, except itself.

**Returns**

List of public methods

**get\_sel\_object\_uuids**(*include\_groups: bool = False*) → `list[str]`

Return selected objects uuids.

**Parameters**

**include\_groups** – If True, also return objects from selected groups.

**Returns**

List of selected objects uuids.

**get\_version()** → `str`

Return DataLab public version.

**Returns**

DataLab version

**import\_h5\_file**(*filename: str, reset\_all: bool | None = None*) → `None`

Open DataLab HDF5 browser to Import HDF5 file.

**Parameters**

- **filename** – HDF5 file name
- **reset\_all** – Reset all application data. Defaults to None.

**import\_macro\_from\_file**(*filename: str*) → `None`

Import macro from file

**Parameters**

**filename** – Filename.

**is\_connected()** → `bool`

Return True if connected to DataLab XML-RPC server.

**load\_from\_directory**(*path: str*) → `None`

Open objects from directory in current panel (signals/images).

**Parameters**

**path** – directory path

**load\_from\_files**(*filenames: list[str]*) → `None`

Open objects from files in current panel (signals/images).

**Parameters**

**filenames** – list of file names

**open\_h5\_files**(*h5files: list[str] | None = None, import\_all: bool | None = None, reset\_all: bool | None = None*) → `None`

Open a DataLab HDF5 file or import from any other HDF5 file.

**Parameters**

- **h5files** – List of HDF5 files to open. Defaults to None.
- **import\_all** – Import all objects from HDF5 files. Defaults to None.
- **reset\_all** – Reset all application data. Defaults to None.

**raise\_window()** → *None*

Raise DataLab window

**reset\_all()** → *None*

Reset all application data

**run\_macro**(*number\_or\_title: int | str | None = None*) → *None*

Run macro.

**Parameters**

**number\_or\_title** – Macro number, or macro title. Defaults to *None* (current macro).

**Raises**

**ValueError** – if macro not found

**save\_to\_h5\_file**(*filename: str*) → *None*

Save to a DataLab HDF5 file.

**Parameters**

**filename** – HDF5 file name

**select\_groups**(*selection: list[int | str] | None = None, panel: str | None = None*) → *None*

Select groups in current panel.

**Parameters**

- **selection** – List of group numbers (1 to N), or list of group uuids, or *None* to select all groups. Defaults to *None*.
- **panel** – panel name (valid values: “signal”, “image”). If *None*, current panel is used. Defaults to *None*.

**select\_objects**(*selection: list[int | str], panel: str | None = None*) → *None*

Select objects in current panel.

**Parameters**

- **selection** – List of object numbers (1 to N) or uuids to select
- **panel** – panel name (valid values: “signal”, “image”). If *None*, current panel is used. Defaults to *None*.

**set\_current\_panel**(*panel: str*) → *None*

Switch to panel.

**Parameters**

**panel** – Panel name (valid values: “signal”, “image”, “macro”)

**stop\_macro**(*number\_or\_title: int | str | None = None*) → *None*

Stop macro.

**Parameters**

**number\_or\_title** – Macro number, or macro title. Defaults to *None* (current macro).

**Raises**

**ValueError** – if macro not found

**toggle\_auto\_refresh**(*state: bool*) → *None*

Toggle auto refresh state.

**Parameters**

**state** – Auto refresh state

**toggle\_show\_titles**(state: *bool*) → *None*

Toggle show titles state.

**Parameters**

**state** – Show titles state

### 3.5.2 Local proxy

The local proxy is used when DataLab is started from the same process as the proxy. In this case, the proxy is directly connected to DataLab main window instance. The typical use case is high-level scripting.

**class** `cdl.proxy.LocalProxy`(cdl: *CDLMainWindow* | *ServerProxy* | *None* = *None*)

DataLab local proxy class.

This class provides access to DataLab features from a proxy class. This is the local version of proxy, which is used when DataLab is started from the same process as the proxy.

**Parameters**

**cdl** (*CDLMainWindow*) – *CDLMainWindow* instance.

---

**Note:** The proxy object also allows to access DataLab computing methods exposed by the processor classes (see *Calling processor methods using proxy objects*).

---

**add\_signal**(title: *str*, xdata: *ndarray*, ydata: *ndarray*, xunit: *str* | *None* = *None*, yunit: *str* | *None* = *None*, xlabel: *str* | *None* = *None*, ylabel: *str* | *None* = *None*, group\_id: *str* = "", set\_current: *bool* = *True*) → *bool*

Add signal data to DataLab.

**Parameters**

- **title** – Signal title
- **xdata** – X data
- **ydata** – Y data
- **xunit** – X unit. Defaults to *None*
- **yunit** – Y unit. Defaults to *None*
- **xlabel** – X label. Defaults to *None*
- **ylabel** – Y label. Defaults to *None*
- **group\_id** – group id in which to add the signal. Defaults to ""
- **set\_current** – if *True*, set the added signal as current

**Returns**

*True* if signal was added successfully, *False* otherwise

**Raises**

- **ValueError** – Invalid xdata dtype
- **ValueError** – Invalid ydata dtype

**add\_image**(title: *str*, data: *ndarray*, xunit: *str* | *None* = *None*, yunit: *str* | *None* = *None*, zunit: *str* | *None* = *None*, xlabel: *str* | *None* = *None*, ylabel: *str* | *None* = *None*, zlabel: *str* | *None* = *None*, group\_id: *str* = "", set\_current: *bool* = *True*) → *bool*

Add image data to DataLab.

**Parameters**

- **title** – Image title
- **data** – Image data
- **xunit** – X unit. Defaults to None
- **yunit** – Y unit. Defaults to None
- **zunit** – Z unit. Defaults to None
- **xlabel** – X label. Defaults to None
- **ylabel** – Y label. Defaults to None
- **zlabel** – Z label. Defaults to None
- **group\_id** – group id in which to add the image. Defaults to “”
- **set\_current** – if True, set the added image as current

**Returns**

True if image was added successfully, False otherwise

**Raises**

**ValueError** – Invalid data dtype

**add\_object**(*obj*: [SignalObj](#) | [ImageObj](#), *group\_id*: *str* = "", *set\_current*: *bool* = True) → None

Add object to DataLab.

**Parameters**

- **obj** – Signal or image object
- **group\_id** – group id in which to add the object. Defaults to “”
- **set\_current** – if True, set the added object as current

**calc**(*name*: *str*, *param*: [DataSet](#) | None = None) → None

Call compute function name in current panel’s processor.

**Parameters**

- **name** – Compute function name
- **param** – Compute function parameter. Defaults to None

**Raises**

**ValueError** – unknown function

**get\_object**(*nb\_id\_title*: *int* | *str* | None = None, *panel*: *str* | None = None) → [SignalObj](#) | [ImageObj](#)

Get object (signal/image) from index.

**Parameters**

- **nb\_id\_title** – Object number, or object id, or object title. Defaults to None (current object)
- **panel** – Panel name. Defaults to None (current panel)

**Returns**

Object

**Raises**

**KeyError** – if object not found

**get\_object\_shapes**(*nb\_id\_title*: *int* | *str* | *None* = *None*, *panel*: *str* | *None* = *None*) → *list*

Get plot item shapes associated to object (signal/image).

**Parameters**

- **nb\_id\_title** – Object number, or object id, or object title. Defaults to *None* (current object)
- **panel** – Panel name. Defaults to *None* (current panel)

**Returns**

List of plot item shapes

**add\_annotations\_from\_items**(*items*: *list*, *refresh\_plot*: *bool* = *True*, *panel*: *str* | *None* = *None*) → *None*

Add object annotations (annotation plot items).

**Parameters**

- **itemsTrue** – annotation plot items
- **refresh\_plotTrue** – refresh plot. Defaults to *True*
- **panel** – panel name (valid values: “signal”, “image”). If *None*, current panel is used

**add\_group**(*title*: *str*, *panel*: *str* | *None* = *None*, *select*: *bool* = *False*) → *None*

Add group to DataLab.

**Parameters**

- **title** – Group title
- **panel** – Panel name (valid values: “signal”, “image”). Defaults to *None*.
- **select** – Select the group after creation. Defaults to *False*.

**add\_label\_with\_title**(*title*: *str* | *None* = *None*, *panel*: *str* | *None* = *None*) → *None*

Add a label with object title on the associated plot

**Parameters**

- **title** – Label title. Defaults to *None*. If *None*, the title is the object title.
- **panel** – panel name (valid values: “signal”, “image”). If *None*, current panel is used.

**close\_application**() → *None*

Close DataLab application

**context\_no\_refresh**() → *Generator*[*None*, *None*, *None*]

Return a context manager to temporarily disable auto refresh.

**Returns**

Context manager

### Example

```
>>> with proxy.context_no_refresh():
...     proxy.add_image("image1", data1)
...     proxy.compute_fft()
...     proxy.compute_wiener()
...     proxy.compute_ifft()
...     # Auto refresh is disabled during the above operations
```

**delete\_metadata**(*refresh\_plot*: *bool* = *True*, *keep\_roi*: *bool* = *False*) → *None*

Delete metadata of selected objects

#### Parameters

- **refresh\_plot** – Refresh plot. Defaults to *True*.
- **keep\_roi** – Keep ROI. Defaults to *False*.

**get\_current\_panel**() → *str*

Return current panel name.

#### Returns

“signal”, “image”, “macro”))

#### Return type

Panel name (valid values

**get\_group\_titles\_with\_object\_infos**() → *tuple*[*list*[*str*], *list*[*list*[*str*]], *list*[*list*[*str*]]]

Return groups titles and lists of inner objects uuids and titles.

#### Returns

groups titles, lists of inner objects uuids and titles

#### Return type

Tuple

**get\_object\_titles**(*panel*: *str* | *None* = *None*) → *list*[*str*]

Get object (signal/image) list for current panel. Objects are sorted by group number and object index in group.

#### Parameters

**panel** – panel name (valid values: “signal”, “image”, “macro”). If *None*, current data panel is used (i.e. signal or image panel).

#### Returns

List of object titles

#### Raises

**ValueError** – if panel not found

**get\_object\_uuids**(*panel*: *str* | *None* = *None*, *group*: *int* | *str* | *None* = *None*) → *list*[*str*]

Get object (signal/image) uuid list for current panel. Objects are sorted by group number and object index in group.

#### Parameters

- **panel** – panel name (valid values: “signal”, “image”). If *None*, current panel is used.
- **group** – Group number, or group id, or group title. Defaults to *None* (all groups).

#### Returns

List of object uuids



**Raises****ValueError** – if panel not found**classmethod** **get\_public\_methods**() → list[str]

Return all public methods of the class, except itself.

**Returns**

List of public methods

**get\_sel\_object\_uuids**(include\_groups: bool = False) → list[str]

Return selected objects uuids.

**Parameters****include\_groups** – If True, also return objects from selected groups.**Returns**

List of selected objects uuids.

**get\_version**() → str

Return DataLab public version.

**Returns**

DataLab version

**import\_h5\_file**(filename: str, reset\_all: bool | None = None) → None

Open DataLab HDF5 browser to Import HDF5 file.

**Parameters**

- **filename** – HDF5 file name
- **reset\_all** – Reset all application data. Defaults to None.

**import\_macro\_from\_file**(filename: str) → None

Import macro from file

**Parameters****filename** – Filename.**load\_from\_directory**(path: str) → None

Open objects from directory in current panel (signals/images).

**Parameters****path** – directory path**load\_from\_files**(filenames: list[str]) → None

Open objects from files in current panel (signals/images).

**Parameters****filenames** – list of file names**open\_h5\_files**(h5files: list[str] | None = None, import\_all: bool | None = None, reset\_all: bool | None = None) → None

Open a DataLab HDF5 file or import from any other HDF5 file.

**Parameters**

- **h5files** – List of HDF5 files to open. Defaults to None.
- **import\_all** – Import all objects from HDF5 files. Defaults to None.
- **reset\_all** – Reset all application data. Defaults to None.

**raise\_window()** → *None*

Raise DataLab window

**reset\_all()** → *None*

Reset all application data

**run\_macro**(*number\_or\_title: int | str | None = None*) → *None*

Run macro.

**Parameters**

**number\_or\_title** – Macro number, or macro title. Defaults to *None* (current macro).

**Raises**

**ValueError** – if macro not found

**save\_to\_h5\_file**(*filename: str*) → *None*

Save to a DataLab HDF5 file.

**Parameters**

**filename** – HDF5 file name

**select\_groups**(*selection: list[int | str] | None = None, panel: str | None = None*) → *None*

Select groups in current panel.

**Parameters**

- **selection** – List of group numbers (1 to N), or list of group uuids, or *None* to select all groups. Defaults to *None*.
- **panel** – panel name (valid values: “signal”, “image”). If *None*, current panel is used. Defaults to *None*.

**select\_objects**(*selection: list[int | str], panel: str | None = None*) → *None*

Select objects in current panel.

**Parameters**

- **selection** – List of object numbers (1 to N) or uuids to select
- **panel** – panel name (valid values: “signal”, “image”). If *None*, current panel is used. Defaults to *None*.

**set\_current\_panel**(*panel: str*) → *None*

Switch to panel.

**Parameters**

**panel** – Panel name (valid values: “signal”, “image”, “macro”)

**stop\_macro**(*number\_or\_title: int | str | None = None*) → *None*

Stop macro.

**Parameters**

**number\_or\_title** – Macro number, or macro title. Defaults to *None* (current macro).

**Raises**

**ValueError** – if macro not found

**toggle\_auto\_refresh**(*state: bool*) → *None*

Toggle auto refresh state.

**Parameters**

**state** – Auto refresh state

`toggle_show_titles(state: bool) → None`

Toggle show titles state.

**Parameters**

**state** – Show titles state

### 3.5.3 Proxy context manager

The proxy context manager is a convenient way to handle proxy creation and destruction. It is used as follows:

```
with proxy_context("local") as proxy:
    proxy.add_signal(...)
```

The proxy type can be “local” or “remote”. For remote proxy, the port can be specified as “remote:port”.

**Note:** The proxy context manager allows to use the proxy in various contexts (Python script, Jupyter notebook, etc.). It also allows to switch seamlessly between local and remote proxy, keeping the same code inside the context.

`cdl.proxy.proxy_context(what: str) → Generator[LocalProxy | RemoteProxy, None, None]`

Context manager handling CDL proxy creation and destruction.

**Parameters**

**what** – proxy type (“local” or “remote”) For remote proxy, the port can be specified as “remote:port”

**Yields**

**proxy**

LocalProxy if what == “local” RemoteProxy if what == “remote” or “remote:port”

#### Example

```
with proxy_context("local") as proxy:
    proxy.add_signal(...)
```

### 3.5.4 Calling processor methods using proxy objects

All the proxy objects provide access to the DataLab computing methods exposed by the processor classes:

- `cdl.core.gui.processor.signal.SignalProcessor`
- `cdl.core.gui.processor.image.ImageProcessor`

See also:

The list of processor methods is available in tables below.

There are two ways to call a processor method:

1. Using the `calc()` method of the proxy object:

```
# Call a method without parameter
proxy.calc("compute_average")

# This is equivalent to:
```

(continues on next page)

(continued from previous page)

```

proxy.calc("average")

# Call a method with parameters
p = cdl.param.MovingAverageParam.create(n=30)
proxy.calc("compute_moving_average", p)

```

2. Directly calling the processor method from the proxy object:

```

# Call a method without parameter
proxy.compute_average()

# Call a method with parameters
p = cdl.param.MovingAverageParam.create(n=30)
proxy.compute_moving_average(p)

```

**Warning:** The `compute_{name}` methods are not statically defined in the proxy classes (and not even dynamically). They are nevertheless available through the proxy objects thanks to the magic method `__getattr__()` which forwards the call to the `calc()` method. However, this means that the methods are not listed in the proxy classes documentation, and they are not available in the auto-completion feature of your IDE.

## Number of compute methods

Table 1: Number of compute methods

Signal	Image	Total
75	102	177

## Signal processing

The following table lists the signal processor methods - it is automatically generated from the source code:

Table 2: Signal processor methods

Compute method	Description
<code>compute_XY_mode()</code>	Compute XY mode with <code>cdl.computation.signal.compute_XY_mode()</code>
<code>compute_abs()</code>	Compute absolute value with <code>cdl.computation.signal.compute_abs()</code>
<code>compute_addition_constant()</code>	Compute sum with a constant
<code>compute_all_stability()</code>	Compute all stability analysis features
<code>compute_allan_deviation()</code>	Compute Allan deviation
<code>compute_allan_variance()</code>	Compute Allan variance
<code>compute_arithmetic()</code>	Compute arithmetic operation between two signals
<code>compute_astype()</code>	Convert data type with <code>cdl.computation.signal.compute_astype()</code>
<code>compute_average()</code>	Compute average with <code>cdl.computation.signal.compute_addition()</code>

continues on next page

Table 2 – continued from previous page

Compute method	Description
<code>compute_bandpass()</code>	Compute band-pass filter
<code>compute_bandstop()</code>	Compute band-stop filter
<code>compute_bandwidth_3db()</code>	Compute bandwidth at -3dB
<code>compute_calibration()</code>	Compute data linear calibration
<code>compute_cartesian2polar()</code>	Convert cartesian to polar coordinates
<code>compute_clip()</code>	Compute maximum data clipping
<code>compute_contrast()</code>	Compute contrast with <code>cdl.computation.signal.compute_contrast()</code>
<code>compute_convolution()</code>	Compute convolution
<code>compute_derivative()</code>	Compute derivative
<code>compute_detrending()</code>	Compute detrending
<code>compute_difference()</code>	Compute difference between two signals
<code>compute_difference_constant()</code>	Compute difference with a constant
<code>compute_division()</code>	Compute division between two signals
<code>compute_division_constant()</code>	Compute division by a constant
<code>compute_dynamic_parameters()</code>	Compute Dynamic Parameters (ENOB, SINAD, THD, SFDR, SNR)
<code>compute_exp()</code>	Compute Log10 with <code>cdl.computation.signal.compute_exp()</code>
<code>compute_fft()</code>	Compute FFT with <code>cdl.computation.signal.compute_fft()</code>
<code>compute_fit()</code>	Compute fitting curve using an interactive dialog
<code>compute_full_width_at_y()</code>	Compute full width at a given y
<code>compute_fw1e2()</code>	Compute FW at $1/e^2$ with <code>cdl.computation.signal.compute_fw1e2()</code>
<code>compute_fwhm()</code>	Compute FWHM with <code>cdl.computation.signal.compute_fwhm()</code>
<code>compute_gaussian_filter()</code>	Compute gaussian filter
<code>compute_hadamard_variance()</code>	Compute Hadamard variance
<code>compute_highpass()</code>	Compute high-pass filter
<code>compute_histogram()</code>	Compute histogram
<code>compute_iff()</code>	Compute iFFT with <code>cdl.computation.signal.compute_iff()</code>
<code>compute_im()</code>	Compute imaginary part with <code>cdl.computation.signal.compute_im()</code>
<code>compute_integral()</code>	Compute integral with <code>cdl.computation.signal.compute_integral()</code>
<code>compute_interpolation()</code>	Compute interpolation
<code>compute_inverse()</code>	Compute inverse
<code>compute_log10()</code>	Compute Log10 with <code>cdl.computation.signal.compute_log10()</code>
<code>compute_lowpass()</code>	Compute high-pass filter
<code>compute_magnitude_spectrum()</code>	Compute magnitude spectrum
<code>compute_modified_allan_variance()</code>	Compute modified Allan variance
<code>compute_moving_average()</code>	Compute moving average
<code>compute_moving_median()</code>	Compute moving median
<code>compute_multigaussianfit()</code>	Compute multi-Gaussian fitting curve using an interactive dialog
<code>compute_normalize()</code>	Normalize data with <code>cdl.computation.signal.compute_normalize()</code>
<code>compute_offset_correction()</code>	Compute offset correction
<code>compute_overlapping_allan_variance()</code>	Compute overlapping Allan variance
<code>compute_peak_detection()</code>	Detect peaks from data

continues on next page

Table 2 – continued from previous page

Compute method	Description
<code>compute_phase_spectrum()</code>	Compute phase spectrum
<code>compute_polar2cartesian()</code>	Convert polar to cartesian coordinates
<code>compute_polyfit()</code>	Compute polynomial fitting curve
<code>compute_power()</code>	Compute power with <code>cdl.computation.signal.compute_power()</code>
<code>compute_product()</code>	Compute product with <code>cdl.computation.signal.compute_product()</code>
<code>compute_product_constant()</code>	Compute product with a constant
<code>compute_psd()</code>	Compute power spectral density
<code>compute_quadratic_difference()</code>	Compute quadratic difference between two signals
<code>compute_re()</code>	Compute real part with <code>cdl.computation.signal.compute_re()</code>
<code>compute_resampling()</code>	Compute resampling
<code>compute_reverse_x()</code>	Reverse X axis with <code>cdl.computation.signal.compute_reverse_x()</code>
<code>compute_roi_extraction()</code>	Extract Region Of Interest (ROI) from data
<code>compute_sampling_rate_period()</code>	Compute sampling rate and period (mean and std)
<code>compute_sqrt()</code>	Compute square root with <code>cdl.computation.signal.compute_sqrt()</code>
<code>compute_stats()</code>	Compute data statistics
<code>compute_sum()</code>	Compute sum with <code>cdl.computation.signal.compute_addition()</code>
<code>compute_swap_axes()</code>	Swap data axes with <code>cdl.computation.signal.compute_swap_axes()</code>
<code>compute_time_deviation()</code>	Compute time deviation
<code>compute_total_variance()</code>	Compute total variance
<code>compute_wiener()</code>	Compute Wiener filter
<code>compute_windowing()</code>	Compute windowing
<code>compute_x_at_minmax()</code>	Compute x at min/max
<code>compute_x_at_y()</code>	Compute x at y with <code>cdl.computation.signal.compute_x_at_y()</code>
<code>compute_y_at_x()</code>	Compute y at x with <code>cdl.computation.signal.compute_y_at_x()</code>
<code>compute_zero_padding()</code>	Compute zero padding

## Image processing

The following table lists the image processor methods - it is automatically generated from the source code:

Table 3: Image processor methods

Compute method	Description
<code>compute_abs()</code>	Compute absolute value with <code>cdl.computation.image.compute_abs()</code>
<code>compute_addition_constant()</code>	Compute sum with a constant
<code>compute_adjust_gamma()</code>	Compute gamma correction
<code>compute_adjust_log()</code>	Compute log correction
<code>compute_adjust_sigmoid()</code>	Compute sigmoid correction
<code>compute_all_denoise()</code>	Compute all denoising filters

continues on next page

Table 3 – continued from previous page

Compute method	Description
<code>compute_all_edges()</code>	Compute all edges filters
<code>compute_all_morphology()</code>	Compute all morphology filters
<code>compute_all_threshold()</code>	Compute all threshold algorithms
<code>compute_arithmetic()</code>	Compute arithmetic operation between two images
<code>compute_astype()</code>	Convert data type with <code>cdl.computation.image.compute_astype()</code>
<code>compute_average()</code>	Compute average with <code>cdl.computation.image.compute_addition()</code>
<code>compute_average_profile()</code>	Compute average profile
<code>compute_binning()</code>	Binning image with <code>cdl.computation.image.compute_binning()</code>
<code>compute_black_tophat()</code>	Compute Black Top-Hat
<code>compute_blob_dog()</code>	Compute blob detection using Difference of Gaussian method
<code>compute_blob_doh()</code>	Compute blob detection using Determinant of Hessian method
<code>compute_blob_log()</code>	Compute blob detection using Laplacian of Gaussian method
<code>compute_blob_opencv()</code>	Compute blob detection using OpenCV
<code>compute_butterworth()</code>	Compute Butterworth filter
<code>compute_calibration()</code>	Compute data linear calibration
<code>compute_canny()</code>	Compute Canny filter
<code>compute_centroid()</code>	Compute image centroid
<code>compute_clip()</code>	Compute maximum data clipping
<code>compute_closing()</code>	Compute morphological closing
<code>compute_contour_shape()</code>	Compute contour shape fit
<code>compute_denoise_bilateral()</code>	Compute bilateral filter denoising
<code>compute_denoise_tophat()</code>	Denoise using White Top-Hat
<code>compute_denoise_tv()</code>	Compute Total Variation denoising
<code>compute_denoise_wavelet()</code>	Compute Wavelet denoising
<code>compute_difference()</code>	Compute difference between two images
<code>compute_difference_constant()</code>	Compute difference with a constant
<code>compute_dilation()</code>	Compute Dilation
<code>compute_division()</code>	Compute division between two images
<code>compute_division_constant()</code>	Compute division by a constant
<code>compute_enclosing_circle()</code>	Compute minimum enclosing circle
<code>compute_equalize_adapthist()</code>	Adaptive histogram equalization
<code>compute_equalize_hist()</code>	Histogram equalization
<code>compute_erosion()</code>	Compute Erosion
<code>compute_exp()</code>	Compute Log10 with <code>cdl.computation.image.compute_exp()</code>
<code>compute_farid()</code>	Compute Farid filter
<code>compute_farid_h()</code>	Compute Farid filter (horizontal)
<code>compute_farid_v()</code>	Compute Farid filter (vertical)
<code>compute_fft()</code>	Compute FFT with <code>cdl.computation.image.compute_fft()</code>
<code>compute_flatfield()</code>	Compute flat field correction
<code>compute_flip()</code>	Flip data horizontally
<code>compute_flipv()</code>	Flip data vertically with <code>cdl.computation.image.compute_flipv()</code>
<code>compute_gaussian_filter()</code>	Compute gaussian filter
<code>compute_histogram()</code>	Compute histogram with <code>cdl.computation.image.compute_histogram()</code>
<code>compute_hough_circle_peaks()</code>	Compute peak detection based on a circle Hough transform
<code>compute_ifft()</code>	Compute iFFT with <code>cdl.computation.image.compute_ifft()</code>

continues on next page



Table 3 – continued from previous page

Compute method	Description
<code>compute_im()</code>	Compute imaginary part with <code>cdl.computation.image.compute_im()</code>
<code>compute_inverse()</code>	Compute inverse
<code>compute_laplace()</code>	Compute Laplace filter
<code>compute_line_profile()</code>	Compute profile along a vertical or horizontal line
<code>compute_log10()</code>	Compute Log10 with <code>cdl.computation.image.compute_log10()</code>
<code>compute_logp1()</code>	Compute base 10 logarithm
<code>compute_magnitude_spectrum()</code>	Compute magnitude spectrum
<code>compute_moving_average()</code>	Compute moving average
<code>compute_moving_median()</code>	Compute moving median
<code>compute_normalize()</code>	Normalize data with <code>cdl.computation.image.compute_normalize()</code>
<code>compute_offset_correction()</code>	Compute offset correction
<code>compute_opening()</code>	Compute morphological opening
<code>compute_peak_detection()</code>	Compute 2D peak detection
<code>compute_phase_spectrum()</code>	Compute phase spectrum
<code>compute_prewitt()</code>	Compute Prewitt filter
<code>compute_prewitt_h()</code>	Compute Prewitt filter (horizontal)
<code>compute_prewitt_v()</code>	Compute Prewitt filter (vertical)
<code>compute_product()</code>	Compute product with <code>cdl.computation.image.compute_product()</code>
<code>compute_product_constant()</code>	Compute product with a constant
<code>compute_psd()</code>	Compute Power Spectral Density (PSD)
<code>compute_quadratic_difference()</code>	Compute quadratic difference between two images
<code>compute_radial_profile()</code>	Compute radial profile
<code>compute_re()</code>	Compute real part with <code>cdl.computation.image.compute_re()</code>
<code>compute_rescale_intensity()</code>	Rescale image intensity levels
<code>compute_resize()</code>	Resize image with <code>cdl.computation.image.compute_resize()</code>
<code>compute_roberts()</code>	Compute Roberts filter
<code>compute_roi_extraction()</code>	Extract Region Of Interest (ROI) from data
<code>compute_rotate()</code>	Rotate data arbitrarily
<code>compute_rotate270()</code>	Rotate data 270° with <code>cdl.computation.image.compute_rotate270()</code>
<code>compute_rotate90()</code>	Rotate data 90° with <code>cdl.computation.image.compute_rotate90()</code>
<code>compute_scharr()</code>	Compute Scharr filter
<code>compute_scharr_h()</code>	Compute Scharr filter (horizontal)
<code>compute_scharr_v()</code>	Compute Scharr filter (vertical)
<code>compute_segment_profile()</code>	Compute profile along a segment
<code>compute_sobel()</code>	Compute Sobel filter
<code>compute_sobel_h()</code>	Compute Sobel filter (horizontal)
<code>compute_sobel_v()</code>	Compute Sobel filter (vertical)
<code>compute_stats()</code>	Compute data statistics
<code>compute_sum()</code>	Compute sum with <code>cdl.computation.image.compute_addition()</code>
<code>compute_swap_axes()</code>	Swap data axes with <code>cdl.computation.image.compute_swap_axes()</code>

continues on next page



Table 3 – continued from previous page

Compute method	Description
<code>compute_threshold()</code>	Compute parametric threshold
<code>compute_threshold_isodata()</code>	Compute threshold using Isodata algorithm
<code>compute_threshold_li()</code>	Compute threshold using Li algorithm
<code>compute_threshold_mean()</code>	Compute threshold using Mean algorithm
<code>compute_threshold_minimum()</code>	Compute threshold using Minimum algorithm
<code>compute_threshold_otsu()</code>	Compute threshold using Otsu algorithm
<code>compute_threshold_triangle()</code>	Compute threshold using Triangle algorithm
<code>compute_threshold_yen()</code>	Compute threshold using Yen algorithm
<code>compute_white_tophat()</code>	Compute White Top-Hat
<code>compute_wiener()</code>	Compute Wiener filter
<code>compute_zero_padding()</code>	Compute zero padding

## 3.6 GUI

The `cdl.core.gui` package contains functionalities related to the graphical user interface (GUI) of the DataLab (CDL) project. Those features are mostly specific to DataLab and are not intended to be used independently.

The purpose of this section of the documentation is to provide an overview of the DataLab GUI architecture and to describe the main features of the modules contained in this package. It is not intended to provide a detailed description of the GUI features, but rather to provide a starting point for the reader who wants to understand the DataLab internal architecture.

DataLab's main window is composed of several parts, each of them being handled by a specific module of this package:

- **Signal and image panels:** those panels are used to display signals and images and to provide a set of tools to manipulate them. Each data panel relies on a set of modules to handle the GUI features (`cdl.core.gui.actionhandler` and `cdl.core.gui.objectview`), the data model (`cdl.core.gui.objectmodel`), the data visualization (`cdl.core.gui.plothandler`), and the data processing (`cdl.core.gui.processor`).
- **Macro panel:** this panel is used to display and run macros. It relies on the `cdl.core.gui.macroeditor` module to handle the macro edition and execution.
- **Specialized widgets:** those widgets are used to handle specific features such as ROI edition (`cdl.core.gui.roieditor`), Intensity profile edition (`cdl.core.gui.profiledialog`), etc.

Submodule	Purpose
<code>cdl.core.gui.main</code>	DataLab main window and application
<code>cdl.core.gui.panel</code>	Signal, image and macro panels
<code>cdl.core.gui.actionhandler</code>	Application actions (menus, toolbars, context menu)
<code>cdl.core.gui.objectview</code>	Widgets to display object (signal/image) trees
<code>cdl.core.gui.plothandler</code>	<i>PlotPy</i> plot items for representing signals and images
<code>cdl.core.gui.roieditor</code>	ROI editor
<code>cdl.core.gui.processor</code>	Processor
<code>cdl.core.gui.docks</code>	Dock widgets
<code>cdl.core.gui.h5io</code>	HDF5 input/output

## 3.7 Main window

The `cdl.core.gui.main` module provides the main window of the DataLab (CDL) project.

**class** `cdl.core.gui.main.CDLMainWindow`(*console=None, hide\_on\_close=False*)

DataLab main window

### Parameters

- **console** – enable internal console
- **hide\_on\_close** – True to hide window on close

**static** `get_instance`(*console=None, hide\_on\_close=False*)

Return singleton instance

**static** `xmlrpc_server_started`(*port*)

XML-RPC server has started, writing comm port in configuration file

`get_group_titles_with_object_infos`() → `tuple[list[str], list[list[str]], list[list[str]]]`

Return groups titles and lists of inner objects uuids and titles.

### Returns

Groups titles, lists of inner objects uuids and titles

`get_object_titles`(*panel: str | None = None*) → `list[str]`

Get object (signal/image) list for current panel. Objects are sorted by group number and object index in group.

### Parameters

**panel** – panel name (valid values: “signal”, “image”, “macro”). If None, current data panel is used (i.e. signal or image panel).

### Returns

List of object titles

### Raises

**ValueError** – if panel is unknown

`get_object`(*nb\_id\_title: int | str | None = None, panel: str | None = None*) → `SignalObj | ImageObj`

Get object (signal/image) from index.

### Parameters

- **nb\_id\_title** – Object number, or object id, or object title. Defaults to None (current object).
- **panel** – Panel name. Defaults to None (current panel).

### Returns

Object

### Raises

- **KeyError** – if object not found
- **TypeError** – if index\_id\_title type is invalid

`get_object_uuids`(*panel: str | None = None, group: int | str | None = None*) → `list[str]`

Get object (signal/image) uuid list for current panel. Objects are sorted by group number and object index in group.

### Parameters

- **panel** – panel name (valid values: “signal”, “image”). If None, current panel is used.
- **group** – Group number, or group id, or group title. Defaults to None (all groups).

**Returns**

List of object uuids

**Raises**

**ValueError** – if panel is unknown

**get\_sel\_object\_uuids**(*include\_groups: bool = False*) → list[str]

Return selected objects uuids.

**Parameters**

**include\_groups** – If True, also return objects from selected groups.

**Returns**

List of selected objects uuids.

**add\_group**(*title: str, panel: str | None = None, select: bool = False*) → None

Add group to DataLab.

**Parameters**

- **title** – Group title
- **panel** – Panel name (valid values: “signal”, “image”). Defaults to None.
- **select** – Select the group after creation. Defaults to False.

**select\_objects**(*selection: list[int | str], panel: str | None = None*) → None

Select objects in current panel.

**Parameters**

- **selection** – List of object numbers (1 to N) or uuids to select
- **panel** – panel name (valid values: “signal”, “image”). If None, current panel is used. Defaults to None.

**select\_groups**(*selection: list[int | str] | None = None, panel: str | None = None*) → None

Select groups in current panel.

**Parameters**

- **selection** – List of group numbers (1 to N), or list of group uuids, or None to select all groups. Defaults to None.
- **panel** – panel name (valid values: “signal”, “image”). If None, current panel is used. Defaults to None.

**delete\_metadata**(*refresh\_plot: bool = True, keep\_roi: bool = False*) → None

Delete metadata of selected objects

**Parameters**

- **refresh\_plot** – Refresh plot. Defaults to True.
- **keep\_roi** – Keep ROI. Defaults to False.

**get\_object\_shapes**(*nb\_id\_title: int | str | None = None, panel: str | None = None*) → list

Get plot item shapes associated to object (signal/image).

**Parameters**

- **nb\_id\_title** – Object number, or object id, or object title. Defaults to None (current object).
- **panel** – Panel name. Defaults to None (current panel).

**Returns**

List of plot item shapes

**add\_annotations\_from\_items**(items: *list*, refresh\_plot: *bool* = *True*, panel: *str* | *None* = *None*) → *None*

Add object annotations (annotation plot items).

**Parameters**

- **items** – annotation plot items
- **refresh\_plot** – refresh plot. Defaults to True.
- **panel** – panel name (valid values: “signal”, “image”). If None, current panel is used.

**add\_label\_with\_title**(title: *str* | *None* = *None*, panel: *str* | *None* = *None*) → *None*

Add a label with object title on the associated plot

**Parameters**

- **title** – Label title. Defaults to None. If None, the title is the object title.
- **panel** – panel name (valid values: “signal”, “image”). If None, current panel is used.

**run\_macro**(number\_or\_title: *int* | *str* | *None* = *None*) → *None*

Run macro.

**Parameters**

**number** – Number of the macro (starting at 1). Defaults to None (run current macro, or does nothing if there is no macro).

**stop\_macro**(number\_or\_title: *int* | *str* | *None* = *None*) → *None*

Stop macro.

**Parameters**

**number** – Number of the macro (starting at 1). Defaults to None (stop current macro, or does nothing if there is no macro).

**import\_macro\_from\_file**(filename: *str*) → *None*

Import macro from file

**Parameters**

**filename** – Filename.

**property panels:** *tuple*[*AbstractPanel*, ...]

Return the tuple of implemented panels (signal, image)

**Returns**

Tuple of panels

**confirm\_memory\_state**() → *bool*

Check memory warning state and eventually show a warning dialog

**Returns**

True if memory state is ok

**check\_stable\_release**() → *None*

Check if this is a stable release

**check\_for\_previous\_crash()** → *None*

Check for previous crash

**execute\_post\_show\_actions()** → *None*

Execute post-show actions

**take\_screenshot(name: str)** → *None*

Take main window screenshot

**take\_menu\_screenshots()** → *None*

Take menu screenshots

**setup(console: bool = False)** → *None*

Setup main window

**Parameters**

**console** – True to setup console

**set\_process\_isolation\_enabled(state: bool)** → *None*

Enable/disable process isolation

**Parameters**

**state** – True to enable process isolation

**get\_current\_panel()** → *str*

Return current panel name

**Returns**

“signal”, “image”, “macro”

**Return type**

Panel name (valid values

**set\_current\_panel(panel: str)** → *None*

Switch to panel.

**Parameters**

**panel** – panel name (valid values: “signal”, “image”, “macro”)

**Raises**

**ValueError** – unknown panel

**calc(name: str, param: DataSet | None = None)** → *None*

Call compute function name in current panel’s processor.

**Parameters**

- **name** – Compute function name
- **param** – Compute function parameter. Defaults to None.

**Raises**

**ValueError** – unknown function

**has\_objects()** → *bool*

Return True if sig/ima panels have any object

**set\_modified(state: bool = True)** → *None*

Set mainwindow modified state

**repopulate\_panel\_trees()** → *None*

Repopulate all panel trees

**toggle\_show\_titles**(*state: bool*) → *None*

Toggle show annotations option

**Parameters**

**state** – state

**toggle\_auto\_refresh**(*state: bool*) → *None*

Toggle auto refresh option

**Parameters**

**state** – state

**toggle\_show\_first\_only**(*state: bool*) → *None*

Toggle show first only option

**Parameters**

**state** – state

**reset\_all()** → *None*

Reset all application data

**save\_to\_h5\_file**(*filename=None*) → *None*

Save to a DataLab HDF5 file

**Parameters**

**filename** – HDF5 filename. If *None*, a file dialog is opened.

**Raises**

**IOError** – if filename is invalid or file cannot be saved.

**open\_h5\_files**(*h5files: list[str] | None = None, import\_all: bool | None = None, reset\_all: bool | None = None*) → *None*

Open a DataLab HDF5 file or import from any other HDF5 file.

**Parameters**

- **h5files** – HDF5 filenames (optionally with dataset name, separated by “:”)
- **import\_all** – Import all datasets from HDF5 files
- **reset\_all** – Reset all application data before importing

**browse\_h5\_files**(*filenames: list[str], reset\_all: bool*) → *None*

Browse HDF5 files

**Parameters**

- **filenames** – HDF5 filenames
- **reset\_all** – Reset all application data before importing

**import\_h5\_file**(*filename: str, reset\_all: bool | None = None*) → *None*

Import HDF5 file into DataLab

**Parameters**

- **filename** – HDF5 filename (optionally with dataset name,
- " (separated by) – “)
- **reset\_all** – Delete all DataLab signals/images before importing data

**add\_object**(*obj*: *SignalObj* | *ImageObj*, *group\_id*: *str* = "", *set\_current*=*True*) → *None*

Add object - signal or image

**Parameters**

- **obj** – object to add (signal or image)
- **group\_id** – group ID (optional)
- **set\_current** – True to set the object as current object

**load\_from\_files**(*filenames*: *list[str]*) → *None*

Open objects from files in current panel (signals/images)

**Parameters**

**filenames** – list of filenames

**load\_from\_directory**(*path*: *str*) → *None*

Open objects from directory in current panel (signals/images).

**Parameters**

**path** – directory path

**get\_version**() → *str*

Return DataLab public version.

**Returns**

DataLab version

**close\_application**() → *None*

Close DataLab application

**raise\_window**() → *None*

Raise DataLab window

**add\_signal**(*title*: *str*, *xdata*: *ndarray*, *ydata*: *ndarray*, *xunit*: *str* | *None* = *None*, *yunit*: *str* | *None* = *None*, *xlabel*: *str* | *None* = *None*, *ylabel*: *str* | *None* = *None*, *group\_id*: *str* = "", *set\_current*: *bool* = *True*) → *bool*

Add signal data to DataLab.

**Parameters**

- **title** – Signal title
- **xdata** – X data
- **ydata** – Y data
- **xunit** – X unit. Defaults to None
- **yunit** – Y unit. Defaults to None
- **xlabel** – X label. Defaults to None
- **ylabel** – Y label. Defaults to None
- **group\_id** – group id in which to add the signal. Defaults to ""
- **set\_current** – if True, set the added signal as current

**Returns**

True if signal was added successfully, False otherwise

**Raises**

- **ValueError** – Invalid xdata dtype
- **ValueError** – Invalid ydata dtype

**add\_image**(title: *str*, data: *ndarray*, xunit: *str* | *None* = *None*, yunit: *str* | *None* = *None*, zunit: *str* | *None* = *None*, xlabel: *str* | *None* = *None*, ylabel: *str* | *None* = *None*, zlabel: *str* | *None* = *None*, group\_id: *str* = "", set\_current: *bool* = *True*) → *bool*

Add image data to DataLab.

#### Parameters

- **title** – Image title
- **data** – Image data
- **xunit** – X unit. Defaults to *None*
- **yunit** – Y unit. Defaults to *None*
- **zunit** – Z unit. Defaults to *None*
- **xlabel** – X label. Defaults to *None*
- **ylabel** – Y label. Defaults to *None*
- **zlabel** – Z label. Defaults to *None*
- **group\_id** – group id in which to add the image. Defaults to ""
- **set\_current** – if *True*, set the added image as current

#### Returns

*True* if image was added successfully, *False* otherwise

#### Raises

**ValueError** – Invalid data dtype

**play\_demo**() → *None*

Play demo

**show\_tour**() → *None*

Show tour

**static test\_segfault\_error**() → *None*

Generate errors (both fault and traceback)

**show**() → *None*

Reimplement QMainWindow method

**close\_properly**() → *bool*

Close properly

#### Returns

*True* if closed properly, *False* otherwise

**closeEvent**(event: *QCloseEvent*) → *None*

Reimplement QMainWindow method



## 3.8 Panel

The `cdl.core.gui.panel` package provides the **panel objects** for signals and images.

Three types of panels are available:

- `cdl.core.gui.panel.signal.SignalPanel`: Signal panel
- `cdl.core.gui.panel.image.ImagePanel`: Image panel
- `cdl.core.gui.panel.macro.MacroPanel`: Macro panel

Signal and Image Panels are called **Data Panels** and are used to display and handle signals and images in the main window of DataLab.

Data Panels rely on the `cdl.core.gui.panel.base.ObjectProp` class (managing the object properties) and a set of modules to handle the GUI features:

- `cdl.core.gui.actionhandler`: Application actions (menus, toolbars, context menu)
- `cdl.core.gui.objectview`: Widgets to display object (signal/image) trees
- `cdl.core.gui.plothandler`: *PlotPy* items for representing signals and images
- `cdl.core.gui.processor`: Processor (computation)
- `cdl.core.gui.panel.roieditor`: ROI editor

The Macro Panel is used to display and run macros. It relies on the `cdl.core.gui.macroeditor` module to handle the macro edition and execution.

### 3.8.1 Base features

`cdl.core.gui.panel.base.is_plot_item_serializable(item: ShapeTypes) → bool`

Return True if plot item is serializable

**class** `cdl.core.gui.panel.base.ObjectProp(panel: BaseDataPanel, paramclass: SignalObj | ImageObj)`

Object handling panel properties

**add\_button**(button)

Add additional button on bottom of properties panel

**set\_param\_label**(param: SignalObj | ImageObj)

Set computing parameters label

**update\_properties\_from**(param: SignalObj | ImageObj | None = None)

Update properties from signal/image dataset

**class** `cdl.core.gui.panel.base.AbstractPanelMeta(name, bases, namespace, /, **kwargs)`

Mixed metaclass to avoid conflicts

**class** `cdl.core.gui.panel.base.AbstractPanel(parent)`

Object defining DataLab panel interface, based on a vertical QSplitter widget

A panel handle an object list (objects are signals, images, macros...). Each object must implement `cdl.core.gui.ObjItf` interface

**get\_serializable\_name**(obj: ObjItf) → str

Return serializable name of object

**serialize\_object\_to\_hdf5**(*obj*: *ObjIf*, *writer*: *NativeH5Writer*) → *None*

Serialize object to HDF5 file

**deserialize\_object\_from\_hdf5**(*reader*: *NativeH5Reader*, *name*: *str*) → *ObjIf*

Deserialize object from a HDF5 file

**abstract serialize\_to\_hdf5**(*writer*: *NativeH5Writer*) → *None*

Serialize whole panel to a HDF5 file

**abstract deserialize\_from\_hdf5**(*reader*: *NativeH5Reader*) → *None*

Deserialize whole panel from a HDF5 file

**abstract create\_object**() → *ObjIf*

Create and return object

**abstract add\_object**(*obj*: *ObjIf*) → *None*

Add object to panel

**abstract remove\_all\_objects**()

Remove all objects

**class** `cdl.core.gui.panel.base.ResultData`(*results*: *list*[*ResultShape* | *ResultProperties*] = *None*, *xlabels*:  
*list*[*str*] = *None*, *ylabels*: *list*[*str*] = *None*)

Result data associated to a shapetype

`cdl.core.gui.panel.base.create_resultdata_dict`(*objs*: *list*[*SignalObj* | *ImageObj*]) → *dict*[*str*,  
*ResultData*]

Return result data dictionary

**Parameters**

**objs** – List of objects

**Returns**

keys are result categories, values are *ResultData*

**Return type**

Result data dictionary

**class** `cdl.core.gui.panel.base.PasteMetadataParam`

Paste metadata parameters

**keep\_roi**

Default: True.

**Type**

`guidata.dataset.dataitems.BoolItem`

**keep\_resultshapes**

Default: False.

**Type**

`guidata.dataset.dataitems.BoolItem`

**keep\_annotations**

Default: True.

**Type**

`guidata.dataset.dataitems.BoolItem`

**keep\_resultproperties**

Default: False.

**Type**`guidata.dataset.dataitems.BoolItem`**keep\_other**

Default: True.

**Type**`guidata.dataset.dataitems.BoolItem`

**classmethod create**(*keep\_roi: bool, keep\_resultshapes: bool, keep\_annotations: bool, keep\_resultproperties: bool, keep\_other: bool*) → `cdl.core.gui.panel.base.PasteMetadataParam`

Returns a new instance of `PasteMetadataParam` with the fields set to the given values.**Parameters**

- **keep\_roi** (*bool*) – Default: True.
- **keep\_resultshapes** (*bool*) – Default: False.
- **keep\_annotations** (*bool*) – Default: True.
- **keep\_resultproperties** (*bool*) – Default: False.
- **keep\_other** (*bool*) – Default: True.

**Returns**New instance of `PasteMetadataParam`.

**class** `cdl.core.gui.panel.base.BaseDataPanel`(*parent: QWidget*)

Object handling the item list, the selected item properties and plot

**abstract static get\_roieditor\_class**() → `Type[TypeROIEditor]`

Return ROI editor class

**closeEvent**(*event*)

Reimplement QMainWindow method

**plot\_item\_parameters\_changed**(*item: CurveItem | MaskedImageItem | LabelItem*) → `None`

Plot items changed: update metadata of all objects from plot items

**plot\_item\_moved**(*item: LabelItem, x0: float, y0: float, x1: float, y1: float*) → `None`

Plot item moved: update metadata of all objects from plot items

**Parameters**

- **item** – Plot item
- **x0** – new x0 coordinate
- **y0** – new y0 coordinate
- **x1** – new x1 coordinate
- **y1** – new y1 coordinate

**serialize\_object\_to\_hdf5**(*obj: TypeObj, writer: NativeH5Writer*) → `None`

Serialize object to HDF5 file

**serialize\_to\_hdf5**(*writer: NativeH5Writer*) → *None*

Serialize whole panel to a HDF5 file

**deserialize\_from\_hdf5**(*reader: NativeH5Reader*) → *None*

Deserialize whole panel from a HDF5 file

**create\_object**() → *TypeObj*

Create object (signal or image)

**Returns**

*SignalObj* or *ImageObj* object

**add\_object**(*obj: TypeObj, group\_id: str | None = None, set\_current: bool = True*) → *None*

Add object

**Parameters**

- **obj** – *SignalObj* or *ImageObj* object
- **group\_id** – group id to which the object belongs. If *None* or empty string, the object is added to the current group.
- **set\_current** – if *True*, set the added object as current

**remove\_all\_objects**() → *None*

Remove all objects

**setup\_panel**() → *None*

Setup panel

**refresh\_plot**(*what: str, update\_items: bool = True, force: bool = False, only\_visible: bool = True, only\_existing: bool = False*) → *None*

Refresh plot.

**Parameters**

- **what** – string describing the objects to refresh. Valid values are “selected” (refresh the selected objects), “all” (refresh all objects), “existing” (refresh existing plot items), or an object uuid.
- **update\_items** – if *True*, update the items. If *False*, only show the items (do not update them, except if the option “Use reference item LUT range” is enabled and more than one item is selected). Defaults to *True*.
- **force** – if *True*, force refresh even if auto refresh is disabled. Defaults to *False*.
- **only\_visible** – if *True*, only refresh visible items. Defaults to *True*. Visible items are the ones that are not hidden by other items or the items except the first one if the option “Show first only” is enabled. This is useful for images, where the last image is the one that is shown. If *False*, all items are refreshed.
- **only\_existing** – if *True*, only refresh existing items. Defaults to *False*. Existing items are the ones that have already been created and are associated to the object uuid. If *False*, create new items for the objects that do not have an item yet.

**Raises**

**ValueError** – if *what* is not a valid value

**manual\_refresh**() → *None*

Manual refresh

**get\_category\_actions**(category: [ActionCategory](#)) → list[QAction]

Return actions for category

**get\_context\_menu**() → QMenu

Update and return context menu

**add\_group**(title: *str*, select: *bool* = *False*) → ObjectGroup

Add group

#### Parameters

- **title** – group title
- **select** – if True, select the group in the tree view. Defaults to False.

#### Returns

Created group object

**duplicate\_object**() → [None](#)

Duplication signal/image object

**copy\_metadata**() → [None](#)

Copy object metadata

**paste\_metadata**(param: [PasteMetadataParam](#) | *None* = *None*) → [None](#)

Paste metadata to selected object(s)

**remove\_object**(force: *bool* = *False*) → [None](#)

Remove signal/image object

#### Parameters

**force** – if True, remove object without confirmation. Defaults to False.

**delete\_all\_objects**() → [None](#)

Confirm before removing all objects

**delete\_metadata**(refresh\_plot: *bool* = *True*, keep\_roi: *bool* | *None* = *None*) → [None](#)

Delete metadata of selected objects

#### Parameters

- **refresh\_plot** – Refresh plot. Defaults to True.
- **keep\_roi** – Keep regions of interest, if any. Defaults to None (ask user).

**add\_annotations\_from\_items**(items: *list*, refresh\_plot: *bool* = *True*) → [None](#)

Add object annotations (annotation plot items).

#### Parameters

- **items** – annotation plot items
- **refresh\_plot** – refresh plot. Defaults to True.

**update\_metadata\_view\_settings**() → [None](#)

Update metadata view settings

**copy\_titles\_to\_clipboard**() → [None](#)

Copy object titles to clipboard (for reproducibility)

**new\_group**() → [None](#)

Create a new group

**rename\_selected\_object\_or\_group**(*new\_name*: *str* | *None* = *None*) → *None*

Rename selected object or group

**Parameters**

**new\_name** – new name (default: *None*, i.e. ask user)

**abstract get\_newparam\_from\_current**(*newparam*: *NewSignalParam* | *NewImageParam* | *None* = *None*)  
→ *NewSignalParam* | *NewImageParam* | *None*

Get new object parameters from the current object.

**Parameters**

**newparam** – new object parameters. If *None*, create a new one.

**Returns**

New object parameters

**abstract new\_object**(*newparam*: *NewSignalParam* | *NewImageParam* | *None* = *None*, *addparam*:  
*gds.DataSet* | *None* = *None*, *edit*: *bool* = *True*, *add\_to\_panel*: *bool* = *True*) →  
*TypeObj* | *None*

Create a new object (signal/image).

**Parameters**

- **newparam** – new object parameters
- **addparam** – additional parameters
- **edit** – Open a dialog box to edit parameters (default: *True*)
- **add\_to\_panel** – Add object to panel (default: *True*)

**Returns**

New object

**set\_current\_object\_title**(*title*: *str*) → *None*

Set current object title

**load\_from\_directory**(*directory*: *str* | *None* = *None*) → *list*[*TypeObj*]

Open objects from directory (signals or images, depending on the panel), add them to DataLab and return them. If the directory is not specified, ask the user to select a directory.

**Parameters**

**directory** – directory name

**Returns**

list of new objects

**load\_from\_files**(*filenames*: *list*[*str*] | *None* = *None*, *create\_group*: *bool* = *False*, *add\_objects*: *bool* = *True*,  
*ignore\_errors*: *bool* = *False*) → *list*[*TypeObj*]

Open objects from file (signals/images), add them to DataLab and return them.

**Parameters**

- **filenames** – File names
- **create\_group** – if *True*, create a new group if more than one object is loaded for a single file. Defaults to *False*: all objects are added to the current group.
- **add\_objects** – if *True*, add objects to the panel. Defaults to *True*.
- **ignore\_errors** – if *True*, ignore errors when loading files. Defaults to *False*.

**Returns**

list of new objects

**save\_to\_files**(*filenames: list[str] | str | None = None*) → *None*

Save selected objects to files (signal/image).

**Parameters****filenames** – File names**handle\_dropped\_files**(*filenames: list[str] | None = None*) → *None*

Handle dropped files

**Parameters****filenames** – File names**Returns***None***exec\_import\_wizard**() → *None*

Execute import wizard

**import\_metadata\_from\_file**(*filename: str | None = None*) → *None*

Import metadata from file (JSON).

**Parameters****filename** – File name**export\_metadata\_from\_file**(*filename: str | None = None*) → *None*

Export metadata to file (JSON).

**Parameters****filename** – File name**selection\_changed**(*update\_items: bool = False*) → *None*

Object selection changed: update object properties, refresh plot and update object view.

**Parameters****update\_items** – Update plot items (default: False)**properties\_changed**() → *None*

The properties ‘Apply’ button was clicked: update object properties, refresh plot and update object view.

**add\_plot\_items\_to\_dialog**(*dlg: PlotDialog, oids: list[str]*) → *None*

Add plot items to dialog

**Parameters**

- **dlg** – Dialog
- **oids** – Object IDs

**open\_separate\_view**(*oids: list[str] | None = None, edit\_annotations: bool = False*) → *PlotDialog | None*

Open separate view for visualizing selected objects

**Parameters**

- **oids** – Object IDs (default: None)
- **edit\_annotations** – Edit annotations (default: False)

**Returns**

Instance of PlotDialog

**create\_new\_dialog**(*edit*: *bool* = *False*, *toolbar*: *bool* = *True*, *title*: *str* | *None* = *None*, *name*: *str* | *None* = *None*, *options*: *dict* | *None* = *None*) → *PlotDialog* | *None*

Create new pop-up signal/image plot dialog.

**Parameters**

- **edit** – Edit mode
- **toolbar** – Show toolbar
- **title** – Dialog title
- **name** – Dialog object name
- **options** – Plot options

**Returns**

Plot dialog instance

**get\_roi\_editor\_output**(*extract*: *bool*) → *tuple*[*TypeROI*, *bool*] | *None*

Get ROI data (array) from specific dialog box.

**Parameters**

**extract** – Extract ROI from data

**Returns**

A tuple containing the ROI object and a boolean indicating whether the dialog was accepted or not.

**get\_objects\_with\_dialog**(*title*: *str*, *comment*: *str* = "", *nb\_objects*: *int* = 1, *parent*: *QWidget* | *None* = *None*) → *TypeObj* | *None*

Get object with dialog box.

**Parameters**

- **title** – Dialog title
- **comment** – Optional dialog comment
- **nb\_objects** – Number of objects to select
- **parent** – Parent widget

**Returns**

Object(s) (signal(s) or image(s), or None if dialog was canceled)

**add\_objprop\_buttons**() → *None*

Insert additional buttons in object properties panel

**show\_results**() → *None*

Show results

**plot\_results**() → *None*

Plot results

**delete\_results**() → *None*

Delete results

**add\_label\_with\_title**(*title*: *str* | *None* = *None*) → *None*

Add a label with object title on the associated plot

**Parameters**

**title** – Label title. Defaults to None. If None, the title is the object title.



### 3.8.2 Signal panel

**class** `cdl.core.gui.panel.signal.SignalPanel`(parent: *QW.QWidget*, dockableplotwidget: *DockablePlotWidget*, panel\_toolbar: *QW.QToolBar*)

Object handling the item list, the selected item properties and plot, specialized for Signal objects

#### PARAMCLASS

Signal object

#### **uuid**

Default: None.

#### **Type**

`guidata.dataset.dataitems.StringItem`

#### **xydata**

Default: None.

#### **Type**

`guidata.dataset.dataitems.FloatArrayItem`

#### **metadata**

Default: {}.

#### **Type**

`guidata.dataset.dataitems.DictItem`

#### **title**

Signal title. Default: 'Untitled'.

#### **Type**

`guidata.dataset.dataitems.StringItem`

#### **xlabel**

Title. Default: ''.

#### **Type**

`guidata.dataset.dataitems.StringItem`

#### **xunit**

Default: ''.

#### **Type**

`guidata.dataset.dataitems.StringItem`

#### **ylabel**

Title. Default: ''.

#### **Type**

`guidata.dataset.dataitems.StringItem`

#### **yunit**

Default: ''.

#### **Type**

`guidata.dataset.dataitems.StringItem`

#### **autoscale**

Default: True.

#### **Type**

`guidata.dataset.dataitems.BoolItem`

#### **xscalelog**

Default: False.

**Type**`guidata.dataset.dataitems.BoolItem`**xscalemin**

Lower bound. Default: None.

**Type**`guidata.dataset.dataitems.FloatItem`**xscalemax**

Upper bound. Default: None.

**Type**`guidata.dataset.dataitems.FloatItem`**yscalelog**

Default: False.

**Type**`guidata.dataset.dataitems.BoolItem`**yscalemin**

Lower bound. Default: None.

**Type**`guidata.dataset.dataitems.FloatItem`**yscalemax**

Upper bound. Default: None.

**Type**`guidata.dataset.dataitems.FloatItem`alias of `SignalObj`

**classmethod** `PARAMCLASS.create`(*uuid*: *str*, *xydata*: *numpy.ndarray*, *metadata*: *dict*, *title*: *str*, *xlabel*: *str*, *xunit*: *str*, *ylabel*: *str*, *yunit*: *str*, *autoscale*: *bool*, *yscalelog*: *bool*, *xscalemin*: *float*, *xscalemax*: *float*, *yscalelog*: *bool*, *yscalemin*: *float*, *yscalemax*: *float*) → `cdl.core.model.signal.SignalObj`

Returns a new instance of `SignalObj` with the fields set to the given values.**Parameters**

- **uuid** (*str*) – Default: None.
- **xydata** (*numpy.ndarray*) – Default: None.
- **metadata** (*dict*) – Default: {}.
- **title** (*str*) – Signal title. Default: 'Untitled'.
- **xlabel** (*str*) – Title. Default: ''.
- **xunit** (*str*) – Default: ''.
- **ylabel** (*str*) – Title. Default: ''.
- **yunit** (*str*) – Default: ''.
- **autoscale** (*bool*) – Default: True.
- **yscalelog** (*bool*) – Default: False.
- **xscalemin** (*float*) – Lower bound. Default: None.
- **xscalemax** (*float*) – Upper bound. Default: None.
- **yscalelog** (*bool*) – Default: False.

- **yscalemin** (*float*) – Lower bound. Default: None.
- **yscalemax** (*float*) – Upper bound. Default: None.

**Returns**

New instance of `SignalObj`.

**IO\_REGISTRY**

alias of `SignalIORegistry`

**static get\_roieditor\_class()** → `Type[SignalROIEditor]`

Return ROI editor class

**get\_newparam\_from\_current** (*newparam*: `NewSignalParam` | `None` = `None`, *title*: `str` | `None` = `None`) → `NewSignalParam` | `None`

Get new object parameters from the current object.

**Parameters**

- **newparam** (*guidata.dataset.DataSet*) – new object parameters. If `None`, create a new one.
- **title** – new object title. If `None`, use the current object title, or the default title.

**Returns**

New object parameters

**new\_object** (*newparam*: `NewSignalParam` | `None` = `None`, *addparam*: `gds.DataSet` | `None` = `None`, *edit*: `bool` = `True`, *add\_to\_panel*: `bool` = `True`) → `SignalObj` | `None`

Create a new object (signal).

**Parameters**

- **newparam** (*guidata.dataset.DataSet*) – new object parameters
- **addparam** (*guidata.dataset.DataSet*) – additional parameters
- **edit** (*bool*) – Open a dialog box to edit parameters (default: `True`)
- **add\_to\_panel** (*bool*) – Add the new object to the panel (default: `True`)

**Returns**

New object

**toggle\_anti\_aliasing** (*state*: `bool`) → `None`

Toggle anti-aliasing on/off

**Parameters**

**state** – state of the anti-aliasing

**reset\_curve\_styles**() → `None`

Reset curve styles

### 3.8.3 Image panel

**class** `cdl.core.gui.panel.image.ImagePanel` (*parent: QW.QWidget, dockableplotwidget: DockablePlotWidget, panel\_toolbar: QW.QToolBar*)

Object handling the item list, the selected item properties and plot, specialized for Image objects

**PARAMCLASS**

Image object

**uuid**

Default: None.

**Type**

`guidata.dataset.dataitems.StringItem`

**data**

Default: None.

**Type**

`guidata.dataset.dataitems.FloatArrayItem`

**metadata**

Default: {}.

**Type**

`guidata.dataset.dataitems.DictItem`

**x0**

X<sub>0</sub>. Default: 0.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**y0**

Y<sub>0</sub>. Default: 0.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**dx**

x. Float, non zero. Default: 1.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**dy**

y. Float, non zero. Default: 1.0.

**Type**

`guidata.dataset.dataitems.FloatItem`

**title**

Image title. Default: 'Untitled'.

**Type**

`guidata.dataset.dataitems.StringItem`

**xlabel**

Title. Default: ''.

**Type**

`guidata.dataset.dataitems.StringItem`

**xunit**

Default: ''.

**Type**  
 guidata.dataset.dataitems.StringItem

**ylabel**  
 Title. Default: ‘’.

**Type**  
 guidata.dataset.dataitems.StringItem

**yunit**  
 Default: ‘’.

**Type**  
 guidata.dataset.dataitems.StringItem

**zlabel**  
 Title. Default: ‘’.

**Type**  
 guidata.dataset.dataitems.StringItem

**zunit**  
 Default: ‘’.

**Type**  
 guidata.dataset.dataitems.StringItem

**autoscale**  
 Default: True.

**Type**  
 guidata.dataset.dataitems.BoolItem

**xscalelog**  
 Default: False.

**Type**  
 guidata.dataset.dataitems.BoolItem

**xscalemin**  
 Lower bound. Default: None.

**Type**  
 guidata.dataset.dataitems.FloatItem

**xscalemax**  
 Upper bound. Default: None.

**Type**  
 guidata.dataset.dataitems.FloatItem

**yscalelog**  
 Default: False.

**Type**  
 guidata.dataset.dataitems.BoolItem

**yscalemin**  
 Lower bound. Default: None.

**Type**  
 guidata.dataset.dataitems.FloatItem

**yscalemax**  
 Upper bound. Default: None.

**Type**  
 guidata.dataset.dataitems.FloatItem

**zscalemín**

Lower bound. Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`

**zscalemáx**

Upper bound. Default: None.

**Type**

`guidata.dataset.dataitems.FloatItem`

alias of `ImageObj`

**classmethod** `PARAMCLASS.create`(*uuid: str, data: numpy.ndarray, metadata: dict, x0: float, y0: float, dx: float, dy: float, title: str, xlabel: str, xunit: str, ylabel: str, yunit: str, zlabel: str, zunit: str, autoscale: bool, xscalelog: bool, xscalemín: float, xscalemáx: float, yscalelog: bool, yscalemín: float, yscalemáx: float, zscalemín: float, zscalemáx: float*) → `cdl.core.model.image.ImageObj`

Returns a new instance of `ImageObj` with the fields set to the given values.

**Parameters**

- **uuid** (*str*) – Default: None.
- **data** (*numpy.ndarray*) – Default: None.
- **metadata** (*dict*) – Default: {}.
- **x0** (*float*) –  $X_0$ . Default: 0.0.
- **y0** (*float*) –  $Y_0$ . Default: 0.0.
- **dx** (*float*) –  $x$ . Float, non zero. Default: 1.0.
- **dy** (*float*) –  $y$ . Float, non zero. Default: 1.0.
- **title** (*str*) – Image title. Default: ‘Untitled’.
- **xlabel** (*str*) – Title. Default: ‘’.
- **xunit** (*str*) – Default: ‘’.
- **ylabel** (*str*) – Title. Default: ‘’.
- **yunit** (*str*) – Default: ‘’.
- **zlabel** (*str*) – Title. Default: ‘’.
- **zunit** (*str*) – Default: ‘’.
- **autoscale** (*bool*) – Default: True.
- **xscalelog** (*bool*) – Default: False.
- **xscálemín** (*float*) – Lower bound. Default: None.
- **xscálemáx** (*float*) – Upper bound. Default: None.
- **yscalelog** (*bool*) – Default: False.
- **yscálemín** (*float*) – Lower bound. Default: None.
- **yscálemáx** (*float*) – Upper bound. Default: None.
- **zscálemín** (*float*) – Lower bound. Default: None.
- **zscálemáx** (*float*) – Upper bound. Default: None.

**Returns**

New instance of ImageObj.

**IO\_REGISTRY**

alias of ImageIORegistry

**static get\_roieditor\_class()** → *Type[ImageROIEditor]*

Return ROI editor class

**plot\_lut\_changed(plot: BasePlot)** → *None*

The LUT of the plot has changed: updating image objects accordingly

**Parameters**

**plot** – Plot object

**get\_newparam\_from\_current**(*newparam: NewImageParam | None = None, title: str | None = None*) → *NewImageParam | None*

Get new object parameters from the current object.

**Parameters**

- **newparam** (*guidata.dataset.DataSet*) – new object parameters. If None, create a new one.
- **title** – new object title. If None, use the current object title, or the default title.

**Returns**

New object parameters

**new\_object**(*newparam: NewImageParam | None = None, addparam: gds.DataSet | None = None, edit: bool = True, add\_to\_panel: bool = True*) → *ImageObj | None*

Create a new object (image).

**Parameters**

- **newparam** (*Daguidata.dataset.datatypes.DataSettaSet*) – new object parameters
- **addparam** (*guidata.dataset.DataSet*) – additional parameters
- **edit** (*bool*) – Open a dialog box to edit parameters (default: True)
- **add\_to\_panel** (*bool*) – Add the object to the panel (default: True)

**Returns**

New object

**toggle\_show\_contrast**(*state: bool*) → *None*

Toggle show contrast option

### 3.8.4 Macro panel

**class** `cdl.core.gui.panel.macro.MacroTabs`(*parent=None*)

Macro tabwidget

**Parameters**

**parent** (*QWidget*) – Parent widget

**clear()** → *None*

Override Qt method

**contextMenuEvent**(*event*)

Override Qt method

**add\_tab**(*macro: Macro*) → *int*

Add tab

**Parameters**

**macro** – Macro object

**Returns**

Number of the tab (starting at 1)

**Return type**

*int*

**remove\_tab**(*number: int*) → *None*

Remove tab

**Parameters**

**number** – Number of the tab (starting at 1)

**get\_widget**(*number: int*) → *CodeEditor*

Return macro editor widget at number

**Parameters**

**number** – Number of the tab (starting at 1)

**Returns**

Macro editor widget

**set\_current\_number**(*number: int*) → *None*

Set current tab number

**Parameters**

**number** – Number of the tab (starting at 1)

**get\_current\_number**() → *int*

Return current tab number

**Returns**

Number of the tab (starting at 1)

**Return type**

*int*

**set\_tab\_title**(*number: int, name: str*) → *None*

Set tab title

**Parameters**

- **number** – Number of the tab (starting at 1)
- **name** – Macro name

**class** `cdl.core.gui.panel.macro.MacroPanel`(*parent: QWidget | None = None*)

Macro Panel widget

**Parameters**

**parent** (*QWidget*) – Parent widget

**update\_color\_mode**() → *None*

Update color mode according to the current theme



**get\_serializable\_name**(*obj: Macro*) → *str*

Return serializable name of object

**serialize\_to\_hdf5**(*writer: NativeH5Writer*) → *None*

Serialize whole panel to a HDF5 file

**Parameters**

**writer** – HDF5 writer

**deserialize\_from\_hdf5**(*reader: NativeH5Reader*) → *None*

Deserialize whole panel from a HDF5 file

**Parameters**

**reader** – HDF5 reader

**create\_object**() → *Macro*

Create object.

**Returns**

Macro object

**add\_object**(*obj: Macro*) → *None*

Add object.

**Parameters**

**obj** – Macro object

**remove\_all\_objects**() → *None*

Remove all objects

**setup\_actions**() → *None*

Setup macro menu actions

**get\_macro**(*number\_or\_title: int | str | None = None*) → *Macro | None*

Return macro at number (if number is None, return current macro)

**Parameters**

**number** – Number of the macro (starting at 1) or title of the macro. Defaults to None (current macro).

**Returns**

Macro object or None (if not found)

**get\_number\_from\_title**(*title: str*) → *int | None*

Return macro number from title

**Parameters**

**title** – Title of the macro

**Returns**

Number of the macro (starting at 1) or None (if not found)

**get\_number\_from\_macro**(*macro: Macro*) → *int | None*

Return macro number from macro object

**Parameters**

**macro** – Macro object

**Returns**

Number of the macro (starting at 1) or None (if not found)

**get\_macro\_titles()** → *list[str]*

Return list of macro titles

**macro\_contents\_changed()** → *None*

One of the macro contents has changed

**run\_macro**(*number\_or\_title: int | str | None = None*) → *None*

Run current macro

**Parameters**

**number** – Number of the macro (starting at 1). Defaults to *None* (run current macro, or does nothing if there is no macro).

**stop\_macro**(*number\_or\_title: int | str | None = None*) → *None*

Stop current macro

**Parameters**

**number** – Number of the macro (starting at 1). Defaults to *None* (run current macro, or does nothing if there is no macro).

**macro\_state\_changed**(*orig\_macro: Macro, state: bool*) → *None*

Macro state has changed (True: started, False: stopped)

**Parameters**

- **orig\_macro** – Macro object
- **state** – State of the macro

**add\_macro()** → *Macro*

Add macro, optionally with name

**Returns**

Macro object

**macro\_name\_changed**(*name: str*) → *None*

Macro name has been changed

**Parameters**

**name** – New name of the macro

**rename\_macro**(*number: int | None = None, title: str | None = None*) → *None*

Rename macro

**Parameters**

- **number** – Number of the macro (starting at 1). Defaults to *None*.
- **title** – Title of the macro. Defaults to *None*.

**export\_macro\_to\_file**(*number\_or\_title: int | str | None = None, filename: str | None = None*) → *None*

Export macro to file

**Parameters**

- **number\_or\_title** – Number of the macro (starting at 1) or title of the macro. Defaults to *None*.
- **filename** – Filename. Defaults to *None*.

**Raises**

**ValueError** – If title is not found

**import\_macro\_from\_file**(filename: *str* | *None* = *None*) → int

Import macro from file

**Parameters**

**filename** – Filename. Defaults to None.

**Returns**

Number of the macro (starting at 1)

**remove\_macro**(number\_or\_title: *int* | *str* | *None* = *None*) → *None*

Remove macro

**Parameters**

**number\_or\_title** – Number of the macro (starting at 1) or title of the macro. Defaults to None.

## 3.9 Action handler

The `cdl.core.gui.actionhandler` module handles all application actions (menus, toolbars, context menu). These actions point to DataLab panels, processors, objecthandler, ...

### 3.9.1 Utility classes

**class** `cdl.core.gui.actionhandler.SelectCond`

Signal or image select conditions

**static** **always**(selected\_groups: *list*[*ObjectGroup*], selected\_objects: *list*[*SignalObj* | *ImageObj*]) → bool

Always true

**static** **exactly\_one**(selected\_groups: *list*[*ObjectGroup*], selected\_objects: *list*[*SignalObj* | *ImageObj*]) → bool

Exactly one signal or image is selected

**static** **exactly\_one\_group**(selected\_groups: *list*[*ObjectGroup*], selected\_objects: *list*[*SignalObj* | *ImageObj*]) → bool

Exactly one group is selected

**static** **exactly\_one\_group\_or\_one\_object**(selected\_groups: *list*[*ObjectGroup*], selected\_objects: *list*[*SignalObj* | *ImageObj*]) → bool

Exactly one group or one signal or image is selected

**static** **at\_least\_one\_group\_or\_one\_object**(sel\_groups: *list*[*ObjectGroup*], sel\_objects: *list*[*SignalObj* | *ImageObj*]) → bool

At least one group or one signal or image is selected

**static** **at\_least\_one**(sel\_groups: *list*[*ObjectGroup*], sel\_objects: *list*[*SignalObj* | *ImageObj*]) → bool

At least one signal or image is selected

**static** **at\_least\_two**(sel\_groups: *list*[*ObjectGroup*], sel\_objects: *list*[*SignalObj* | *ImageObj*]) → bool

At least two signals or images are selected

**static** **with\_roi**(selected\_groups: *list*[*ObjectGroup*], selected\_objects: *list*[*SignalObj* | *ImageObj*]) → bool

At least one signal or image has a ROI

```
class cdl.core.gui.actionhandler.ActionCategory(value, names=None, *, module=None,
                                              qualname=None, type=None, start=1,
                                              boundary=None)
```

Action categories

### 3.9.2 Handler classes

```
class cdl.core.gui.actionhandler.SignalActionHandler(panel: SignalPanel | ImagePanel,
                                                    panel_toolbar: QW.QToolBar, view_toolbar:
                                                    QW.QToolBar)
```

Object handling signal panel GUI interactions: actions, menus, ...

**create\_first\_actions()**

Create actions that are added to the menus in the first place

**create\_last\_actions()**

Create actions that are added to the menus in the end

**add\_action**(action: QAction, select\_condition: Callable | None = None) → None

Add action to list of actions.

#### Parameters

- **action** – action to add
- **select\_condition** – condition to enable action. Defaults to None. If None, action is enabled if at least one object is selected.

**add\_to\_action\_list**(action: QAction, category: ActionCategory | None = None, pos: int | None = None, sep: bool = False) → None

Add action to list of actions.

#### Parameters

- **action** – action to add
- **category** – action category. Defaults to None. If None, action is added to the current category.
- **pos** – add action to menu at this position. Defaults to None. If None, action is added at the end of the list.
- **sep** – add separator before action in menu (or after if pos is positive). Defaults to False.

**create\_all\_actions()**

Create all actions

**new\_action**(title: str, position: int | None = None, separator: bool = False, triggered: Callable | None = None, toggled: Callable | None = None, shortcut: QShortcut | None = None, icon\_name: str | None = None, tip: str | None = None, select\_condition: Callable | str | None = None, context\_menu\_pos: int | None = None, context\_menu\_sep: bool = False, toolbar\_pos: int | None = None, toolbar\_sep: bool = False, toolbar\_category: ActionCategory | None = None) → QAction

Create new action and add it to list of actions.

#### Parameters

- **title** – action title

- **position** – add action to menu at this position. Defaults to None.
- **separator** – add separator before action in menu (or after if pos is positive). Defaults to False.
- **triggered** – triggered callback. Defaults to None.
- **toggled** – toggled callback. Defaults to None.
- **shortcut** – shortcut. Defaults to None.
- **icon\_name** – icon name. Defaults to None.
- **tip** – tooltip. Defaults to None.
- **select\_condition** – selection condition. Defaults to None. If str, must be the name of a method of SelectCond, i.e. one of “always”, “exactly\_one”, “exactly\_one\_group”, “at\_least\_one\_group\_or\_one\_object”, “at\_least\_one”, “at\_least\_two”, “with\_roi”.
- **context\_menu\_pos** – add action to context menu at this position. Defaults to None.
- **context\_menu\_sep** – add separator before action in context menu (or after if context\_menu\_pos is positive). Defaults to False.
- **toolbar\_pos** – add action to toolbar at this position. Defaults to None.
- **toolbar\_sep** – add separator before action in toolbar (or after if toolbar\_pos is positive). Defaults to False.
- **toolbar\_category** – toolbar category. Defaults to None. If toolbar\_pos is not None, this specifies the category of the toolbar. If None, defaults to ActionCategory.VIEW\_TOOLBAR if the current category is ActionCategory.VIEW, else to ActionCategory.PANEL\_TOOLBAR.

**Returns**

New action

**new\_category**(category: ActionCategory) → Generator[None, None, None]

Context manager for creating a new menu.

**Parameters**

**category** – Action category

**Yields**

None

**new\_menu**(title: str, icon\_name: str | None = None) → Generator[None, None, None]

Context manager for creating a new menu.

**Parameters**

- **title** – Menu title
- **icon\_name** – Menu icon name. Defaults to None.

**Yields**

None

**selected\_objects\_changed**(selected\_groups: list[ObjectGroup], selected\_objects: list[SignalObj | ImageObj]) → None

Update actions based on selected objects.

**Parameters**

- **selected\_groups** – selected groups

- **selected\_objects** – selected objects

**class** `cdl.core.gui.actionhandler.ImageActionHandler`(*panel: SignalPanel | ImagePanel, panel\_toolbar: QW.QToolBar, view\_toolbar: QW.QToolBar*)

Object handling image panel GUI interactions: actions, menus, ...

**create\_first\_actions()**

Create actions that are added to the menus in the first place

**create\_last\_actions()**

Create actions that are added to the menus in the end

**add\_action**(*action: QAction, select\_condition: Callable | None = None*) → *None*

Add action to list of actions.

#### Parameters

- **action** – action to add
- **select\_condition** – condition to enable action. Defaults to None. If None, action is enabled if at least one object is selected.

**add\_to\_action\_list**(*action: QAction, category: ActionCategory | None = None, pos: int | None = None, sep: bool = False*) → *None*

Add action to list of actions.

#### Parameters

- **action** – action to add
- **category** – action category. Defaults to None. If None, action is added to the current category.
- **pos** – add action to menu at this position. Defaults to None. If None, action is added at the end of the list.
- **sep** – add separator before action in menu (or after if pos is positive). Defaults to False.

**create\_all\_actions()**

Create all actions

**new\_action**(*title: str, position: int | None = None, separator: bool = False, triggered: Callable | None = None, toggled: Callable | None = None, shortcut: QShortcut | None = None, icon\_name: str | None = None, tip: str | None = None, select\_condition: Callable | str | None = None, context\_menu\_pos: int | None = None, context\_menu\_sep: bool = False, toolbar\_pos: int | None = None, toolbar\_sep: bool = False, toolbar\_category: ActionCategory | None = None*) → *QAction*

Create new action and add it to list of actions.

#### Parameters

- **title** – action title
- **position** – add action to menu at this position. Defaults to None.
- **separator** – add separator before action in menu (or after if pos is positive). Defaults to False.
- **triggered** – triggered callback. Defaults to None.
- **toggled** – toggled callback. Defaults to None.
- **shortcut** – shortcut. Defaults to None.

- **icon\_name** – icon name. Defaults to None.
- **tip** – tooltip. Defaults to None.
- **select\_condition** – selection condition. Defaults to None. If str, must be the name of a method of SelectCond, i.e. one of “always”, “exactly\_one”, “exactly\_one\_group”, “at\_least\_one\_group\_or\_one\_object”, “at\_least\_one”, “at\_least\_two”, “with\_roi”.
- **context\_menu\_pos** – add action to context menu at this position. Defaults to None.
- **context\_menu\_sep** – add separator before action in context menu (or after if context\_menu\_pos is positive). Defaults to False.
- **toolbar\_pos** – add action to toolbar at this position. Defaults to None.
- **toolbar\_sep** – add separator before action in toolbar (or after if toolbar\_pos is positive). Defaults to False.
- **toolbar\_category** – toolbar category. Defaults to None. If toolbar\_pos is not None, this specifies the category of the toolbar. If None, defaults to ActionCategory.VIEW\_TOOLBAR if the current category is ActionCategory.VIEW, else to ActionCategory.PANEL\_TOOLBAR.

**Returns**

New action

**new\_category**(category: ActionCategory) → Generator[None, None, None]

Context manager for creating a new menu.

**Parameters**

**category** – Action category

**Yields**

None

**new\_menu**(title: str, icon\_name: str | None = None) → Generator[None, None, None]

Context manager for creating a new menu.

**Parameters**

- **title** – Menu title
- **icon\_name** – Menu icon name. Defaults to None.

**Yields**

None

**selected\_objects\_changed**(selected\_groups: list[ObjectGroup], selected\_objects: list[SignalObj | ImageObj]) → None

Update actions based on selected objects.

**Parameters**

- **selected\_groups** – selected groups
- **selected\_objects** – selected objects

## 3.10 Object view

The `cdl.core.gui.objectview` module provides widgets to display object (signal/image) trees.

---

**Note:** This module provides tree widgets to display signals, images and groups. It is important to note that, by design, the user can only select either individual signals/images or groups, but not both at the same time. This is an important design choice, as it allows to simplify the user experience, and to avoid potential confusion between the two types of selection.

---

### 3.10.1 Simple object tree

**class** `cdl.core.gui.objectview.SimpleObjectTree`(parent: *QW.QWidget*, objmodel: *ObjectModel*)

Base object handling panel list widget, object (sig/ima) lists

**initialize\_from**(sobjlist: *SimpleObjectTree*) → *None*

Init from another SimpleObjectList, without making copies of objects

**iter\_items**(item: *QTreeWidgetItem* | *None* = *None*) → *Iterator*[*QTreeWidgetItem*]

Recursively iterate over all items

**get\_item\_from\_id**(item\_id) → *QTreeWidgetItem*

Return *QTreeWidgetItem* from id (stored in item's data)

**get\_current\_item\_id**(object\_only: *bool* = *False*) → *str* | *None*

Return current item id

**set\_current\_item\_id**(uuid: *str*, extend: *bool* = *False*) → *None*

Set current item by id

**get\_current\_group\_id**() → *str*

Return current group ID

**get\_sel\_group\_items**() → *list*[*QTreeWidgetItem*]

Return selected group items

**get\_sel\_group\_uuids**() → *list*[*str*]

Return selected group uuids

**get\_sel\_object\_items**() → *list*[*QTreeWidgetItem*]

Return selected object items

**get\_sel\_object\_uuids**(include\_groups: *bool* = *False*) → *list*[*str*]

Return selected objects uuids.

**Parameters**

**include\_groups** – If True, also return objects from selected groups.

**Returns**

List of selected objects uuids.

**get\_sel\_objects**(include\_groups: *bool* = *False*) → *list*[*SignalObj* | *ImageObj*]

Return selected objects.

If include\_groups is True, also return objects from selected groups.



**get\_sel\_groups()** → list[ObjectGroup]  
Return selected groups

**populate\_tree()** → None  
Populate tree with objects

**update\_tree()** → None  
Update tree

**add\_group\_item(group: ObjectGroup)** → None  
Add group item

**add\_object\_item(obj: SignalObj | ImageObj, group\_id: str, set\_current: bool = True)** → None  
Add item

**update\_item(uuid: str)** → None  
Update item

**remove\_item(oid: str, refresh: bool = True)** → None  
Remove item

**item\_double\_clicked(item: QTreeWidgetItem)** → None  
Item was double-clicked: open a pop-up plot dialog

**contextMenuEvent(event: QContextMenuEvent)** → None  
Override Qt method

### 3.10.2 Get object dialog

**class** cdl.core.gui.objectview.GetObjectsDialog(parent: QW.QWidget, panel: BaseDataPanel, title: str, comment: str = "", nb\_objects: int = 1, minimum\_size: tuple[int, int] | None = None)

Dialog box showing groups and objects (signals or images) to select one, or more.

#### Parameters

- **parent** – parent widget
- **panel** – data panel
- **title** – dialog title
- **comment** – optional dialog comment
- **nb\_objects** – number of objects to select (default: 1)
- **minimum\_size** – minimum size (width, height)

**get\_selected\_objects()** → list[SignalObj | ImageObj]  
Return selected objects

### 3.10.3 Object view

```
class cdl.core.gui.objectview.ObjectView(parent: QW.QWidget, objmodel: ObjectModel)
```

Object handling panel list widget, object (sig/ima) lists

```
    paintEvent(event)
```

Reimplement Qt method

```
    dragEnterEvent(event: QDragEnterEvent) → None
```

Reimplement Qt method

```
    dragLeaveEvent(event: QDragLeaveEvent) → None
```

Reimplement Qt method

```
    dragMoveEvent(event: QDragMoveEvent) → None
```

Reimplement Qt method

```
    get_all_group_uuids() → list[str]
```

Return all group uuids, in a list ordered by group position in the tree

```
    get_all_object_uuids() → dict[str, list[str]]
```

Return all object uuids, in a dictionary that maps group uuids to the list of object uuids in each group, in the correct order

```
    dropEvent(event: QDropEvent) → None
```

Reimplement Qt method

```
    get_current_object() → SignalObj | ImageObj | None
```

Return current object

```
    set_current_object(obj: SignalObj | ImageObj) → None
```

Set current object

```
    item_selection_changed() → None
```

Refreshing the selection of objects and groups, emitting the SIG\_SELECTION\_CHANGED signal which triggers the update of the object properties panel, the plot items and the actions of the toolbar and menu bar.

This method is called when the user selects or deselects items in the tree. It is also called when the user clicks on an item that was already selected.

This method emits the SIG\_SELECTION\_CHANGED signal.

```
    select_objects(selection: list[SignalObj | ImageObj | int | str]) → None
```

Select multiple objects

**Parameters**

**selection** – list of objects, object numbers (1 to N) or object uuids

```
    select_groups(groups: list[ObjectGroup | int | str] | None = None) → None
```

Select multiple groups

**Parameters**

**groups** – list of groups, group numbers (1 to N), group names or None (select all groups). Defaults to None.

```
    move_up()
```

Move selected objects/groups up

**move\_down()**

Move selected objects/groups down

## 3.11 Plot handler

The `cdl.core.gui.plothandler` module provides plot handlers for signal and image panels, that is, classes handling *PlotPy* plot items for representing signals and images.

### 3.11.1 Signal plot handler

**class** `cdl.core.gui.plothandler.SignalPlotHandler`(*panel*: `BaseDataPanel`, *plotwidget*: `PlotWidget`)

Object handling signal plot items, plot dialogs, plot options

**toggle\_anti\_aliasing**(*state*: `bool`) → `None`

Toggle anti-aliasing

**Parameters**

**state** – if True, enable anti-aliasing

**get\_current\_plot\_options**() → `PlotOptions`

Return standard signal/image plot options

**add\_shapes**(*oid*: `str`, *do\_autoscale*: `bool` = `False`) → `None`

Add geometric shape items associated to computed results and annotations, for the object with the given uuid

**cleanup\_dataview**() → `None`

Clean up data view

**clear**() → `None`

Clear plot items

**get**(*key*: `str`, *default*: `TypePlotItem` | `None` = `None`) → `TypePlotItem` | `None`

Return item associated to object uuid. If the key is not found, default is returned if given, otherwise None is returned.

**get\_obj\_from\_item**(*item*: `TypePlotItem`) → `TypeObj` | `None`

Return object associated to plot item

**Parameters**

**item** – plot item

**Returns**

Object associated to plot item

**reduce\_shown\_oids**(*oids*: `list[str]`) → `list[str]`

Reduce the number of shown objects to visible items only. The base implementation is to show only the first selected item if the option “Show first only” is enabled.

**Parameters**

**oids** – list of object uuids

**Returns**

Reduced list of object uuids

**refresh\_plot**(*what*: *str*, *update\_items*: *bool* = *True*, *force*: *bool* = *False*, *only\_visible*: *bool* = *True*, *only\_existing*: *bool* = *False*) → *None*

Refresh plot.

#### Parameters

- **what** – string describing the objects to refresh. Valid values are “selected” (refresh the selected objects), “all” (refresh all objects), “existing” (refresh existing plot items), or an object uuid.
- **update\_items** – if *True*, update the items. If *False*, only show the items (do not update them, except if the option “Use reference item LUT range” is enabled and more than one item is selected). Defaults to *True*.
- **force** – if *True*, force refresh even if auto refresh is disabled. Defaults to *False*.
- **only\_visible** – if *True*, only refresh visible items. Defaults to *True*. Visible items are the ones that are not hidden by other items or the items except the first one if the option “Show first only” is enabled. This is useful for images, where the last image is the one that is shown. If *False*, all items are refreshed.
- **only\_existing** – if *True*, only refresh existing items. Defaults to *False*. Existing items are the ones that have already been created and are associated to the object uuid. If *False*, create new items for the objects that do not have an item yet.

#### Raises

**ValueError** – if *what* is not a valid value

**remove\_all\_shape\_items**() → *None*

Remove all geometric shapes associated to result items

**remove\_item**(*oid*: *str*) → *None*

Remove plot item associated to object uuid

**set\_auto\_refresh**(*auto\_refresh*: *bool*) → *None*

Set auto refresh mode.

#### Parameters

**auto\_refresh** – if *True*, refresh plot items automatically

**set\_show\_first\_only**(*show\_first\_only*: *bool*) → *None*

Set show first only mode.

#### Parameters

**show\_first\_only** – if *True*, show only the first selected item

**static update\_item\_according\_to\_ref\_item**(*item*: *TypePlotItem*, *ref\_item*: *TypePlotItem*) → *None*

Update plot item according to reference item

**update\_resultproperty\_from\_plot\_item**(*item*: *LabelItem*) → *None*

Update result property from plot item

### 3.11.2 Image plot handler

**class** `cdl.core.gui.plothandler.ImagePlotHandler`(*panel*: `BaseDataPanel`, *plotwidget*: `PlotWidget`)

Object handling image plot items, plot dialogs, plot options

**static** `update_item_according_to_ref_item`(*item*: `MaskedImageItem`, *ref\_item*: `MaskedImageItem`)  
→ `None`

Update plot item according to reference item

**reduce\_shown\_oids**(*oids*: `list[str]`) → `list[str]`

Reduce the number of shown objects to visible items only. The base implementation is to show only the first selected item if the option “Show first only” is enabled.

**Parameters**

**oids** – list of object uuids

**Returns**

Reduced list of object uuids

**refresh\_plot**(*what*: `str`, *update\_items*: `bool` = `True`, *force*: `bool` = `False`, *only\_visible*: `bool` = `True`,  
*only\_existing*: `bool` = `False`) → `None`

Refresh plot.

**Parameters**

- **what** – string describing the objects to refresh. Valid values are “selected” (refresh the selected objects), “all” (refresh all objects), “existing” (refresh existing plot items), or an object uuid.
- **update\_items** – if `True`, update the items. If `False`, only show the items (do not update them, except if the option “Use reference item LUT range” is enabled and more than one item is selected). Defaults to `True`.
- **force** – if `True`, force refresh even if auto refresh is disabled. Defaults to `False`.
- **only\_visible** – if `True`, only refresh visible items. Defaults to `True`. Visible items are the ones that are not hidden by other items or the items except the first one if the option “Show first only” is enabled. This is useful for images, where the last image is the one that is shown. If `False`, all items are refreshed.
- **only\_existing** – if `True`, only refresh existing items. Defaults to `False`. Existing items are the ones that have already been created and are associated to the object uuid. If `False`, create new items for the objects that do not have an item yet.

**Raises**

**ValueError** – if *what* is not a valid value

**cleanup\_dataview**() → `None`

Clean up data view

**get\_current\_plot\_options**() → `PlotOptions`

Return standard signal/image plot options

**add\_shapes**(*oid*: `str`, *do\_autoscale*: `bool` = `False`) → `None`

Add geometric shape items associated to computed results and annotations, for the object with the given uuid

**clear**() → `None`

Clear plot items

**get**(key: *str*, default: *TypePlotItem* | *None* = *None*) → *TypePlotItem* | *None*

Return item associated to object uuid. If the key is not found, default is returned if given, otherwise None is returned.

**get\_obj\_from\_item**(item: *TypePlotItem*) → *TypeObj* | *None*

Return object associated to plot item

**Parameters**

**item** – plot item

**Returns**

Object associated to plot item

**remove\_all\_shape\_items**() → *None*

Remove all geometric shapes associated to result items

**remove\_item**(oid: *str*) → *None*

Remove plot item associated to object uuid

**set\_auto\_refresh**(auto\_refresh: *bool*) → *None*

Set auto refresh mode.

**Parameters**

**auto\_refresh** – if True, refresh plot items automatically

**set\_show\_first\_only**(show\_first\_only: *bool*) → *None*

Set show first only mode.

**Parameters**

**show\_first\_only** – if True, show only the first selected item

**update\_resultproperty\_from\_plot\_item**(item: *LabelItem*) → *None*

Update result property from plot item

## 3.12 ROI editor

The `cdl.core.gui.roieditor` module provides the ROI editor widgets for signals and images.

### 3.12.1 Signal ROI editor

**class** `cdl.core.gui.roieditor.SignalROIEditor`(parent: *PlotDialog*, obj: *TypeObj*, extract: *bool*, item: *TypePlotItem* | *None* = *None*)

Signal ROI Editor

**get\_obj\_roi\_class**() → *type[SignalROI]*

Get object ROI class

**add\_tools\_to\_plot\_dialog**() → *None*

Add tools to plot dialog

**manually\_add\_roi**() → *None*

Manually add segment ROI

**create\_coordinate\_based\_roi\_actions**() → *list[QAction]*

Create coordinate-based ROI actions

**setup\_widget()** → *None*  
 Setup ROI editor widget

**update\_roi\_titles()**  
 Update ROI annotation titles

### 3.12.2 Image ROI editor

**class** `cdl.core.gui.roieditor.ImageROIEditor`(parent: *PlotDialog*, obj: *TypeObj*, extract: *bool*, item: *TypePlotItem* | *None* = *None*)

Image ROI Editor

**get\_obj\_roi\_class()** → *type[ImageROI]*  
 Get object ROI class

**add\_tools\_to\_plot\_dialog()** → *None*  
 Add tools to plot dialog

**manually\_add\_roi**(roi\_type: *Literal*['rectangle', 'circle', 'polygon']) → *None*  
 Manually add image ROI

**create\_coordinate\_based\_roi\_actions()** → *list*[*QAction*]  
 Create coordinate-based ROI actions

**setup\_widget()** → *None*  
 Setup ROI editor widget

**update\_roi\_titles()** → *None*  
 Update ROI annotation titles

## 3.13 Processor

The `cdl.core.gui.processor` package provides the **processor objects** for signals and images.

Processor objects are the bridge between the computation modules (in `cdl.computation`) and the GUI modules (in `cdl.core.gui`). They are used to call the computation functions and to update the GUI from inside the data panel objects.

When implementing a processing feature in DataLab, the steps are usually the following:

- Add an action in the `cdl.core.gui.actionhandler` module to trigger the processing feature from the GUI (e.g. a menu item or a toolbar button).
- Implement the computation function in the `cdl.computation` module (that would eventually call the algorithm from the `cdl.algorithms` module).
- Implement the processor object method in this package to call the computation function and eventually update the GUI.

The processor objects are organized in submodules according to their purpose.

The following submodules are available:

- `cdl.core.gui.processor.base`: Common processing features
- `cdl.core.gui.processor.signal`: Signal processing features
- `cdl.core.gui.processor.image`: Image processing features

### 3.13.1 Common features

**class** `cdl.core.gui.processor.base.Worker`

Multiprocessing worker, to run long-running tasks in a separate process

**static** `create_pool()` → `None`

Create multiprocessing pool

**static** `terminate_pool(wait: bool = False)` → `None`

Terminate multiprocessing pool.

**Parameters**

**wait** – wait for all tasks to finish. Defaults to False.

**restart\_pool()** → `None`

Terminate and recreate the pool

**run**(*func: Callable, args: tuple[Any]*) → `None`

Run computation.

**Parameters**

- **func** – function to run
- **args** – arguments

**close()** → `None`

Close worker: close pool properly and wait for all tasks to finish

**is\_computation\_finished()** → `bool`

Return True if computation is finished.

**Returns**

True if computation is finished

**Return type**

`bool`

**get\_result()** → `CompOut`

Return computation result.

**Returns**

computation result

**Return type**

`CompOut`

`cdl.core.gui.processor.base.is_pairwise_mode()` → `bool`

Return True if operation mode is pairwise.

**Returns**

True if operation mode is pairwise

**Return type**

`bool`

**class** `cdl.core.gui.processor.base.BaseProcessor`(*panel: SignalPanel | ImagePanel, plotwidget: PlotWidget*)

Object handling data processing: operations, processing, analysis.

**Parameters**



- **panel** – panel
- **plotwidget** – plot widget

**close()**

Close processor properly

**set\_process\_isolation\_enabled**(*enabled: bool*) → *None*

Set process isolation enabled.

**Parameters**

**enabled** – enabled

**has\_param\_defaults**(*paramclass: type[DataSet]*) → *bool*

Return True if parameter defaults are available.

**Parameters**

**paramclass** – parameter class

**Returns**

True if parameter defaults are available

**Return type**

*bool*

**update\_param\_defaults**(*param: DataSet*) → *None*

Update parameter defaults.

**Parameters**

**param** – parameters

**init\_param**(*param: DataSet, paramclass: type[DataSet], title: str, comment: str | None = None*) → *tuple[bool, DataSet]*

Initialize processing parameters.

**Parameters**

- **param** – parameter
- **paramclass** – parameter class
- **title** – title
- **comment** – comment

**Returns**

Tuple (edit, param) where edit is True if parameters have been edited, False otherwise.

**compute\_11**(*func: Callable, param: DataSet | None = None, paramclass: DataSet | None = None, title: str | None = None, comment: str | None = None, edit: bool | None = None*) → *None*

Compute 11 function: 1 object in → 1 object out.

**Parameters**

- **func** – function
- **param** – parameter
- **paramclass** – parameter class
- **title** – title
- **comment** – comment
- **edit** – edit parameters

**compute\_1n**(*funcs*: *list*[*Callable*] | *Callable*, *params*: *list* | *None* = *None*, *title*: *str* | *None* = *None*, *edit*: *bool* | *None* = *None*) → *None*

Compute 1n function: 1 object in → n objects out.

#### Parameters

- **funcs** – list of functions
- **params** – list of parameters
- **title** – title
- **edit** – edit parameters

**handle\_output**(*compout*: *CompOut*, *context*: *str*, *progress*: *QW.QProgressDialog*) → *SignalObj* | *ImageObj* | *ResultShape* | *ResultProperties* | *None*

Handle computation output: if error, display error message, if warning, display warning message.

#### Parameters

- **compout** – computation output
- **context** – context (e.g. “Computing: Gaussian filter”)
- **progress** – progress dialog

#### Returns

a signal or image object, or a result shape object,  
or None if error

#### Return type

Output object

**compute\_10**(*func*: *Callable*, *param*: *DataSet* | *None* = *None*, *paramclass*: *DataSet* | *None* = *None*, *title*: *str* | *None* = *None*, *comment*: *str* | *None* = *None*, *edit*: *bool* | *None* = *None*) → dict[*str*, *ResultShape* | *ResultProperties*]

Compute 10 function: 1 object in → 0 object out (the result of this method is stored in original object’s metadata).

#### Parameters

- **func** – function to execute
- **param** – parameters. Defaults to None.
- **paramclass** – parameters class. Defaults to None.
- **title** – title of progress bar. Defaults to None.
- **comment** – comment. Defaults to None.
- **edit** – if True, edit parameters. Defaults to None.

#### Returns

object uuid, values: *ResultShape* or  
*ResultProperties* objects)

#### Return type

Dictionary of results (keys

**compute\_n1**(*name*: *str*, *func*: *Callable*, *param*: *DataSet* | *None* = *None*, *paramclass*: *DataSet* | *None* = *None*, *title*: *str* | *None* = *None*, *comment*: *str* | *None* = *None*, *func\_objs*: *Callable* | *None* = *None*, *edit*: *bool* | *None* = *None*) → *None*

Compute `n1` function:  $N(\geq 2)$  objects in  $\rightarrow$  1 object out.

#### Parameters

- **name** – name of function
- **func** – function to execute
- **param** – parameters. Defaults to None.
- **paramclass** – parameters class. Defaults to None.
- **title** – title of progress bar. Defaults to None.
- **comment** – comment. Defaults to None.
- **func\_objs** – function to execute on objects. Defaults to None.
- **edit** – if True, edit parameters. Defaults to None.

**compute\_n1n**(*obj2*: *Obj* | *list*[*Obj*] | *None*, *obj2\_name*: *str*, *func*: *Callable*, *param*: *gds.DataSet* | *None* = *None*, *paramclass*: *gds.DataSet* | *None* = *None*, *title*: *str* | *None* = *None*, *comment*: *str* | *None* = *None*, *edit*: *bool* | *None* = *None*)  $\rightarrow$  *None*

Compute `n1n` function:  $N(\geq 1)$  objects + 1 object in  $\rightarrow$   $N$  objects out.

---

**Note:** In pairwise mode, the function is executed on each pair of objects, so the logic is different:

$N(\geq 1)$  objects +  $N$  objects in  $\rightarrow$   $N$  objects out

In other words, the `n1n` function in single operand mode becomes a `nnn` function in pairwise mode.

---

Examples: subtract, divide

#### Parameters

- **obj2** – second object (or list of objects in case of pairwise operation mode)
- **obj2\_name** – name of second object
- **func** – function to execute
- **param** – parameters. Defaults to None.
- **paramclass** – parameters class. Defaults to None.
- **title** – title of progress bar. Defaults to None.
- **comment** – comment. Defaults to None.
- **edit** – if True, edit parameters. Defaults to None.

**abstract compute\_arithmetic**(*obj2*: *Obj* | *None* = *None*, *param*: *ArithmeticParam* | *None* = *None*)  $\rightarrow$  *None*

Compute arithmetic operation

**abstract compute\_sum**()  $\rightarrow$  *None*

Compute sum

**abstract compute\_normalize**(*param*: *NormalizeParam* | *None* = *None*)  $\rightarrow$  *None*

Normalize data

**abstract compute\_average**()  $\rightarrow$  *None*

Compute average

**abstract compute\_product()** → *None*

Compute product

**abstract compute\_difference(obj2: *Obj* | *list[Obj]* | *None = None*)** → *None*

Compute difference

**abstract compute\_quadratic\_difference(obj2: *Obj* | *list[Obj]* | *None = None*)** → *None*

Compute quadratic difference

**abstract compute\_division(obj2: *Obj* | *list[Obj]* | *None = None*)** → *None*

Compute division

**abstract compute\_swap\_axes()** → *None*

Swap data axes

**abstract compute\_inverse()** → *None*

Compute inverse

**abstract compute\_abs()** → *None*

Compute absolute value

**abstract compute\_re()** → *None*

Compute real part

**abstract compute\_im()** → *None*

Compute imaginary part

**abstract compute\_astype()** → *None*

Convert data type

**abstract compute\_log10()** → *None*

Compute Log10

**abstract compute\_exp()** → *None*

Compute exponential

**abstract compute\_calibration(param=*None*)** → *None*

Compute data linear calibration

**abstract compute\_clip(param: *ClipParam* | *None = None*)** → *None*

Compute maximum data clipping

**abstract compute\_gaussian\_filter(param: *GaussianParam* | *None = None*)** → *None*

Compute gaussian filter

**abstract compute\_moving\_average(param: *MovingAverageParam* | *None = None*)** → *None*

Compute moving average

**abstract compute\_moving\_median(param: *MovingMedianParam* | *None = None*)** → *None*

Compute moving median

**abstract compute\_wiener()** → *None*

Compute Wiener filter

**abstract compute\_fft()** → *None*

Compute iFFT

**abstract compute\_ifft()** → *None*  
 Compute FFT

**abstract compute\_addition\_constant**(*param: ConstantParam*) → *None*  
 Compute sum with a constant

**abstract compute\_difference\_constant**(*param: ConstantParam*) → *None*  
 Compute difference with a constant

**abstract compute\_product\_constant**(*param: ConstantParam*) → *None*  
 Compute product with a constant

**abstract compute\_division\_constant**(*param: ConstantParam*) → *None*  
 Compute division by a constant

**compute\_roi\_extraction**(*roi: TypeROI | None = None*) → *None*  
 Extract Region Of Interest (ROI) from data with:

- *cdl.computation.image.extract\_single\_roi()* for single ROI
- *cdl.computation.image.extract\_multiple\_roi()* for multiple ROIs

**edit\_regions\_of\_interest**(*extract: bool = False*) → *TypeROI | None*  
 Define Region Of Interest (ROI).

**Parameters**  
**extract** – If True, ROI is extracted from data. Defaults to False.

**Returns**  
 ROI object or None if ROI dialog has been canceled.

**delete\_regions\_of\_interest()** → *None*  
 Delete Regions Of Interest

**abstract compute\_stats()** → *dict[str, ResultShape]*  
 Compute data statistics

### 3.13.2 Signal processing features

**class** *cdl.core.gui.processor.signal*.**SignalProcessor**(*panel: SignalPanel | ImagePanel, plotwidget: PlotWidget*)

Object handling signal processing: operations, processing, analysis

**compute\_sum()** → *None*  
 Compute sum with *cdl.computation.signal.compute\_addition()*

**compute\_addition\_constant**(*param: ConstantParam | None = None*) → *None*  
 Compute sum with a constant with *cdl.computation.signal.compute\_addition\_constant()*

**compute\_average()** → *None*  
 Compute average with *cdl.computation.signal.compute\_addition()* and divide by the number of signals

**compute\_product()** → *None*  
 Compute product with *cdl.computation.signal.compute\_product()*

**compute\_product\_constant**(*param*: ConstantParam | *None* = *None*) → *None*  
Compute product with a constant with `cdl.computation.signal.compute_product_constant()`

**compute\_swap\_axes**() → *None*  
Swap data axes with `cdl.computation.signal.compute_swap_axes()`

**compute\_inverse**() → *None*  
Compute inverse

**compute\_abs**() → *None*  
Compute absolute value with `cdl.computation.signal.compute_abs()`

**compute\_re**() → *None*  
Compute real part with `cdl.computation.signal.compute_re()`

**compute\_im**() → *None*  
Compute imaginary part with `cdl.computation.signal.compute_im()`

**compute\_astype**(*param*: DataTypeSParam | *None* = *None*) → *None*  
Convert data type with `cdl.computation.signal.compute_astype()`

**compute\_log10**() → *None*  
Compute Log10 with `cdl.computation.signal.compute_log10()`

**compute\_exp**() → *None*  
Compute Log10 with `cdl.computation.signal.compute_exp()`

**compute\_sqrt**() → *None*  
Compute square root with `cdl.computation.signal.compute_sqrt()`

**compute\_power**(*param*: PowerParam | *None* = *None*) → *None*  
Compute power with `cdl.computation.signal.compute_power()`

**compute\_arithmetic**(*obj2*: SignalObj | *None* = *None*, *param*: ArithmeticParam | *None* = *None*) → *None*  
Compute arithmetic operation between two signals with `cdl.computation.signal.compute_arithmetic()`

**compute\_difference**(*obj2*: SignalObj | list[SignalObj] | *None* = *None*) → *None*  
Compute difference between two signals with `cdl.computation.signal.compute_difference()`

**compute\_difference\_constant**(*param*: ConstantParam | *None* = *None*) → *None*  
Compute difference with a constant with `cdl.computation.signal.compute_difference_constant()`

**compute\_quadratic\_difference**(*obj2*: SignalObj | list[SignalObj] | *None* = *None*) → *None*  
Compute quadratic difference between two signals with `cdl.computation.signal.compute_quadratic_difference()`

**compute\_division**(*obj2*: SignalObj | list[SignalObj] | *None* = *None*) → *None*  
Compute division between two signals with `cdl.computation.signal.compute_division()`

**compute\_division\_constant**(*param*: ConstantParam | *None* = *None*) → *None*  
Compute division by a constant with `cdl.computation.signal.compute_division_constant()`

**compute\_peak\_detection**(*param*: PeakDetectionParam | *None* = *None*) → *None*  
Detect peaks from data with `cdl.computation.signal.compute_peak_detection()`

**compute\_reverse\_x()** → None  
Reverse X axis with `cdl.computation.signal.compute_reverse_x()`

**compute\_cartesian2polar**(*param*: AngleUnitParam | None = None) → None  
Convert cartesian to polar coordinates with `cdl.computation.signal.compute_cartesian2polar()`

**compute\_polar2cartesian**(*param*: AngleUnitParam | None = None) → None  
Convert polar to cartesian coordinates with `cdl.computation.signal.compute_polar2cartesian()`.

**compute\_normalize**(*param*: NormalizeParam | None = None) → None  
Normalize data with `cdl.computation.signal.compute_normalize()`

**compute\_derivative()** → None  
Compute derivative with `cdl.computation.signal.compute_derivative()`

**compute\_integral()** → None  
Compute integral with `cdl.computation.signal.compute_integral()`

**compute\_calibration**(*param*: XYCalibrateParam | None = None) → None  
Compute data linear calibration with `cdl.computation.signal.compute_calibration()`

**compute\_clip**(*param*: ClipParam | None = None) → None  
Compute maximum data clipping with `cdl.computation.signal.compute_clip()`

**compute\_offset\_correction**(*param*: ROIIDParam | None = None) → None  
Compute offset correction with `cdl.computation.signal.compute_offset_correction()`

**compute\_gaussian\_filter**(*param*: GaussianParam | None = None) → None  
Compute gaussian filter with `cdl.computation.signal.compute_gaussian_filter()`

**compute\_moving\_average**(*param*: MovingAverageParam | None = None) → None  
Compute moving average with `cdl.computation.signal.compute_moving_average()`

**compute\_moving\_median**(*param*: MovingMedianParam | None = None) → None  
Compute moving median with `cdl.computation.signal.compute_moving_median()`

**compute\_wiener()** → None  
Compute Wiener filter with `cdl.computation.signal.compute_wiener()`

**compute\_lowpass**(*param*: LowPassFilterParam | None = None) → None  
Compute high-pass filter with `cdl.computation.signal.compute_filter()`

**compute\_highpass**(*param*: HighPassFilterParam | None = None) → None  
Compute high-pass filter with `cdl.computation.signal.compute_filter()`

**compute\_bandpass**(*param*: BandPassFilterParam | None = None) → None  
Compute band-pass filter with `cdl.computation.signal.compute_filter()`

**compute\_bandstop**(*param*: BandStopFilterParam | None = None) → None  
Compute band-stop filter with `cdl.computation.signal.compute_filter()`

**compute\_zero\_padding**(*param*: ZeroPadding1DParam | None = None) → None  
Compute zero padding with `cdl.computation.signal.compute_zero_padding()`

**compute\_fft**(*param*: FFTParam | None = None) → None  
Compute FFT with `cdl.computation.signal.compute_fft()`

**compute\_ifft**(*param*: FFTParam | None = None) → None  
Compute iFFT with `cdl.computation.signal.compute_ifft()`

**compute\_magnitude\_spectrum**(*param*: SpectrumParam | None = None) → None  
Compute magnitude spectrum with `cdl.computation.signal.compute_magnitude_spectrum()`

**compute\_phase\_spectrum**() → None  
Compute phase spectrum with `cdl.computation.signal.compute_phase_spectrum()`

**compute\_psd**(*param*: SpectrumParam | None = None) → None  
Compute power spectral density with `cdl.computation.signal.compute_psd()`

**compute\_interpolation**(*obj2*: SignalObj | None = None, *param*: InterpolationParam | None = None)  
Compute interpolation with `cdl.computation.signal.compute_interpolation()`

**compute\_resampling**(*param*: ResamplingParam | None = None)  
Compute resampling with `cdl.computation.signal.compute_resampling()`

**compute\_detrending**(*param*: DetrendingParam | None = None)  
Compute detrending with `cdl.computation.signal.compute_detrending()`

**compute\_XY\_mode**(*obj2*: SignalObj | None = None) → None  
Compute XY mode with `cdl.computation.signal.compute_XY_mode.()`

**compute\_convolution**(*obj2*: SignalObj | None = None) → None  
Compute convolution with `cdl.computation.signal.compute_convolution()`

**compute\_windowing**(*param*: WindowingParam | None = None) → None  
Compute windowing with `cdl.computation.signal.compute_windowing()`

**compute\_allan\_variance**(*param*: AllanVarianceParam | None = None) → None  
Compute Allan variance with `cdl.computation.signal.compute_allan_variance()`

**compute\_allan\_deviation**(*param*: AllanVarianceParam | None = None) → None  
Compute Allan deviation with `cdl.computation.signal.compute_allan_deviation()`

**compute\_overlapping\_allan\_variance**(*param*: AllanVarianceParam | None = None) → None  
Compute overlapping Allan variance with `cdl.computation.signal.compute_overlapping_allan_variance()`

**compute\_modified\_allan\_variance**(*param*: AllanVarianceParam | None = None) → None  
Compute modified Allan variance with `cdl.computation.signal.compute_modified_allan_variance()`

**compute\_hadamard\_variance**(*param*: AllanVarianceParam | None = None) → None  
Compute Hadamard variance with `cdl.computation.signal.compute_hadamard_variance()`

**compute\_total\_variance**(*param*: AllanVarianceParam | None = None) → None  
Compute total variance with `cdl.computation.signal.compute_total_variance()`

**compute\_time\_deviation**(*param*: AllanVarianceParam | None = None) → None  
Compute time deviation with `cdl.computation.signal.compute_time_deviation()`

**compute\_all\_stability**(*param*: AllanVarianceParam | None = None) → None  
Compute all stability analysis features using the following functions:

- `cdl.computation.signal.compute_allan_variance()`
- `cdl.computation.signal.compute_allan_deviation()`



- `cdl.computation.signal.compute_overlapping_allan_variance()`
- `cdl.computation.signal.compute_modified_allan_variance()`
- `cdl.computation.signal.compute_hadamard_variance()`
- `cdl.computation.signal.compute_total_variance()`
- `cdl.computation.signal.compute_time_deviation()`

**compute\_polyfit**(*param*: PolynomialFitParam | None = None) → None

Compute polynomial fitting curve

**compute\_fit**(*title*: str, *fitdlgfunc*: Callable) → None

Compute fitting curve using an interactive dialog

#### Parameters

- **title** – Title of the dialog
- **fitdlgfunc** – Fitting dialog function

**compute\_multigaussianfit**() → None

Compute multi-Gaussian fitting curve using an interactive dialog

**compute\_fwhm**(*param*: FWHMParam | None = None) → dict[str, ResultShape]

Compute FWHM with `cdl.computation.signal.compute_fwhm()`

**compute\_fw1e2**() → dict[str, ResultShape]

Compute FW at  $1/e^2$  with `cdl.computation.signal.compute_fw1e2()`

**compute\_full\_width\_at\_y**(*param*: OrdinateParam | None = None) → dict[str, ResultShape]

Compute full width at a given y with `cdl.computation.signal.compute_full_width_at_y()`

**compute\_stats**() → dict[str, ResultProperties]

Compute data statistics with `cdl.computation.signal.compute_stats()`

**compute\_histogram**(*param*: HistogramParam | None = None) → dict[str, ResultShape]

Compute histogram with `cdl.computation.signal.compute_histogram()`

**compute\_contrast**() → dict[str, ResultProperties]

Compute contrast with `cdl.computation.signal.compute_contrast()`

**compute\_x\_at\_minmax**() → dict[str, ResultProperties]

Compute x at min/max with `cdl.computation.signal.compute_x_at_minmax()`

**compute\_x\_at\_y**(*param*: OrdinateParam | None = None) → dict[str, ResultProperties]

Compute x at y with `cdl.computation.signal.compute_x_at_y()`.

**compute\_y\_at\_x**(*param*: AbscissaParam | None = None) → dict[str, ResultProperties]

Compute y at x with `cdl.computation.signal.compute_y_at_x()`.

**compute\_sampling\_rate\_period**() → dict[str, ResultProperties]

Compute sampling rate and period (mean and std) with `cdl.computation.signal.compute_sampling_rate_period()`

**compute\_bandwidth\_3db**() → None

Compute bandwidth at -3dB with `cdl.computation.signal.compute_bandwidth_3db()`

**compute\_dynamic\_parameters**(*param*: DynamicParam | None = None) → dict[str, ResultProperties]

Compute Dynamic Parameters (ENOB, SINAD, THD, SFDR, SNR) with `cdl.computation.signal.compute_dynamic_parameters()`

### 3.13.3 Image processing features

**class** `cdl.core.gui.processor.image.ImageProcessor`(*panel*: `SignalPanel` | `ImagePanel`, *plotwidget*: `PlotWidget`)

Object handling image processing: operations, processing, analysis

**compute\_normalize**(*param*: `NormalizeParam` | `None = None`) → `None`

Normalize data with `cdl.computation.image.compute_normalize()`

**compute\_sum**() → `None`

Compute sum with `cdl.computation.image.compute_addition()`

**compute\_addition\_constant**(*param*: `ConstantParam` | `None = None`) → `None`

Compute sum with a constant using `cdl.computation.image.compute_addition_constant()`

**compute\_average**() → `None`

Compute average with `cdl.computation.image.compute_addition()` and dividing by the number of images

**compute\_product**() → `None`

Compute product with `cdl.computation.image.compute_product()`

**compute\_product\_constant**(*param*: `ConstantParam` | `None = None`) → `None`

Compute product with a constant using `cdl.computation.image.compute_product_constant()`

**compute\_logp1**(*param*: `LogP1Param` | `None = None`) → `None`

Compute base 10 logarithm using `cdl.computation.image.compute_logp1()`

**compute\_rotate**(*param*: `RotateParam` | `None = None`) → `None`

Rotate data arbitrarily using `cdl.computation.image.compute_rotate()`

**compute\_rotate90**() → `None`

Rotate data 90° with `cdl.computation.image.compute_rotate90()`

**compute\_rotate270**() → `None`

Rotate data 270° with `cdl.computation.image.compute_rotate270()`

**compute\_fliph**() → `None`

Flip data horizontally using `cdl.computation.image.compute_fliph()`

**compute\_flipv**() → `None`

Flip data vertically with `cdl.computation.image.compute_flipv()`

**distribute\_on\_grid**(*param*: `GridParam` | `None = None`) → `None`

Distribute images on a grid

**reset\_positions**() → `None`

Reset image positions

**compute\_resize**(*param*: `ResizeParam` | `None = None`) → `None`

Resize image with `cdl.computation.image.compute_resize()`

**compute\_binning**(*param*: `BinningParam` | `None = None`) → `None`

Binning image with `cdl.computation.image.compute_binning()`

**compute\_line\_profile**(*param*: `LineProfileParam` | `None = None`) → `None`

Compute profile along a vertical or horizontal line with `cdl.computation.image.compute_line_profile()`

**compute\_segment\_profile**(*param*: SegmentProfileParam | *None* = *None*)  
 Compute profile along a segment with `cdl.computation.image.compute_segment_profile()`

**compute\_average\_profile**(*param*: AverageProfileParam | *None* = *None*) → *None*  
 Compute average profile with `cdl.computation.image.compute_average_profile()`

**compute\_radial\_profile**(*param*: RadialProfileParam | *None* = *None*) → *None*  
 Compute radial profile with `cdl.computation.image.compute_radial_profile()`

**compute\_histogram**(*param*: HistogramParam | *None* = *None*) → *None*  
 Compute histogram with `cdl.computation.image.compute_histogram()`

**compute\_swap\_axes**() → *None*  
 Swap data axes with `cdl.computation.image.compute_swap_axes()`.

**compute\_inverse**() → *None*  
 Compute inverse

**compute\_abs**() → *None*  
 Compute absolute value with `cdl.computation.image.compute_abs()`

**compute\_re**() → *None*  
 Compute real part with `cdl.computation.image.compute_re()`

**compute\_im**() → *None*  
 Compute imaginary part with `cdl.computation.image.compute_im()`

**compute\_astype**(*param*: DataTypeIParam | *None* = *None*) → *None*  
 Convert data type with `cdl.computation.image.compute_astype()`

**compute\_log10**() → *None*  
 Compute Log10 with `cdl.computation.image.compute_log10()`

**compute\_exp**() → *None*  
 Compute Log10 with `cdl.computation.image.compute_exp()`

**compute\_arithmetic**(*obj2*: ImageObj | *None* = *None*, *param*: ArithmeticParam | *None* = *None*) → *None*  
 Compute arithmetic operation between two images with `cdl.computation.image.compute_arithmetic()`

**compute\_difference**(*obj2*: ImageObj | list[ImageObj] | *None* = *None*) → *None*  
 Compute difference between two images with `cdl.computation.image.compute_difference()`

**compute\_difference\_constant**(*param*: ConstantParam | *None* = *None*) → *None*  
 Compute difference with a constant with `cdl.computation.image.compute_difference_constant()`

**compute\_quadratic\_difference**(*obj2*: ImageObj | list[ImageObj] | *None* = *None*) → *None*  
 Compute quadratic difference between two images with `cdl.computation.image.compute_quadratic_difference()`

**compute\_division**(*obj2*: ImageObj | list[ImageObj] | *None* = *None*) → *None*  
 Compute division between two images with `cdl.computation.image.compute_division()`

**compute\_division\_constant**(*param*: ConstantParam | *None* = *None*) → *None*  
 Compute division by a constant with `cdl.computation.image.compute_division_constant()`

**compute\_flatfield**(*obj2*: ImageObj | *None* = *None*, *param*: FlatFieldParam | *None* = *None*) → *None*  
Compute flat field correction with `cdl.computation.image.compute_flatfield()`

**compute\_calibration**(*param*: ZCalibrateParam | *None* = *None*) → *None*  
Compute data linear calibration with `cdl.computation.image.compute_calibration()`

**compute\_clip**(*param*: ClipParam | *None* = *None*) → *None*  
Compute maximum data clipping with `cdl.computation.image.compute_clip()`

**compute\_offset\_correction**(*param*: ROI2DParam | *None* = *None*) → *None*  
Compute offset correction with `cdl.computation.image.compute_offset_correction()`

**compute\_gaussian\_filter**(*param*: GaussianParam | *None* = *None*) → *None*  
Compute gaussian filter with `cdl.computation.image.compute_gaussian_filter()`

**compute\_moving\_average**(*param*: MovingAverageParam | *None* = *None*) → *None*  
Compute moving average with `cdl.computation.image.compute_moving_average()`

**compute\_moving\_median**(*param*: MovingMedianParam | *None* = *None*) → *None*  
Compute moving median with `cdl.computation.image.compute_moving_median()`

**compute\_wiener**() → *None*  
Compute Wiener filter with `cdl.computation.image.compute_wiener()`

**compute\_zero\_padding**(*param*: ZeroPadding2DParam | *None* = *None*) → *None*  
Compute zero padding with `cdl.computation.image.compute_zero_padding()`

**compute\_fft**(*param*: FFTParam | *None* = *None*) → *None*  
Compute FFT with `cdl.computation.image.compute_fft()`

**compute\_ifft**(*param*: FFTParam | *None* = *None*) → *None*  
Compute iFFT with `cdl.computation.image.compute_ifft()`

**compute\_magnitude\_spectrum**(*param*: SpectrumParam | *None* = *None*) → *None*  
Compute magnitude spectrum with `cdl.computation.image.compute_magnitude_spectrum()`

**compute\_phase\_spectrum**() → *None*  
Compute phase spectrum with `cdl.computation.image.compute_phase_spectrum()`

**compute\_psd**(*param*: SpectrumParam | *None* = *None*) → *None*  
Compute Power Spectral Density (PSD) with `cdl.computation.image.compute_psd()`

**compute\_butterworth**(*param*: ButterworthParam | *None* = *None*) → *None*  
Compute Butterworth filter with `cdl.computation.image.compute_butterworth()`

**compute\_threshold**(*param*: ThresholdParam | *None* = *None*) → *None*  
Compute parametric threshold with `cdl.computation.image.threshold.compute_threshold()`

**compute\_threshold\_isodata**() → *None*  
Compute threshold using Isodata algorithm with `cdl.computation.image.threshold.compute_threshold_isodata()`

**compute\_threshold\_li**() → *None*  
Compute threshold using Li algorithm with `cdl.computation.image.threshold.compute_threshold_li()`

**compute\_threshold\_mean()** → None

Compute threshold using Mean algorithm with `cdl.computation.image.threshold.compute_threshold_mean()`

**compute\_threshold\_minimum()** → None

Compute threshold using Minimum algorithm with `cdl.computation.image.threshold.compute_threshold_minimum()`

**compute\_threshold\_otsu()** → None

Compute threshold using Otsu algorithm with `cdl.computation.image.threshold.compute_threshold_otsu()`

**compute\_threshold\_triangle()** → None

Compute threshold using Triangle algorithm with `cdl.computation.image.threshold.compute_threshold_triangle()`

**compute\_threshold\_yen()** → None

Compute threshold using Yen algorithm with `cdl.computation.image.threshold.compute_threshold_yen()`

**compute\_all\_threshold()** → None

Compute all threshold algorithms using the following functions:

- `cdl.computation.image.threshold.compute_threshold_isodata()`
- `cdl.computation.image.threshold.compute_threshold_li()`
- `cdl.computation.image.threshold.compute_threshold_mean()`
- `cdl.computation.image.threshold.compute_threshold_minimum()`
- `cdl.computation.image.threshold.compute_threshold_otsu()`
- `cdl.computation.image.threshold.compute_threshold_triangle()`
- `cdl.computation.image.threshold.compute_threshold_yen()`

**compute\_adjust\_gamma**(*param*: AdjustGammaParam | None = None) → None

Compute gamma correction with `cdl.computation.image.exposure.compute_adjust_gamma()`

**compute\_adjust\_log**(*param*: AdjustLogParam | None = None) → None

Compute log correction with `cdl.computation.image.exposure.compute_adjust_log()`

**compute\_adjust\_sigmoid**(*param*: AdjustSigmoidParam | None = None) → None

Compute sigmoid correction with `cdl.computation.image.exposure.compute_adjust_sigmoid()`

**compute\_rescale\_intensity**(*param*: RescaleIntensityParam | None = None) → None

Rescale image intensity levels with `:py:func`cdl.computation.image.exposure.compute_rescale_intensity``

**compute\_equalize\_hist**(*param*: EqualizeHistParam | None = None) → None

Histogram equalization with `cdl.computation.image.exposure.compute_equalize_hist()`

**compute\_equalize\_adapthist**(*param*: EqualizeAdaptHistParam | None = None) → None

Adaptive histogram equalization with `cdl.computation.image.exposure.compute_equalize_adapthist()`

**compute\_denoise\_tv**(*param*: DenoiseTVParam | None = None) → None

Compute Total Variation denoising with `cdl.computation.image.restoration.compute_denoise_tv()`

**compute\_denoise\_bilateral**(*param*: DenoiseBilateralParam | *None* = *None*) → *None*  
Compute bilateral filter denoising with `cdl.computation.image.restoration.compute_denoise_bilateral()`

**compute\_denoise\_wavelet**(*param*: DenoiseWaveletParam | *None* = *None*) → *None*  
Compute Wavelet denoising with `cdl.computation.image.restoration.compute_denoise_wavelet()`

**compute\_denoise\_tophat**(*param*: MorphologyParam | *None* = *None*) → *None*  
Denoise using White Top-Hat with `cdl.computation.image.restoration.compute_denoise_tophat()`

**compute\_all\_denoise**(*params*: list | *None* = *None*) → *None*  
Compute all denoising filters using the following functions:

- `cdl.computation.image.restoration.compute_denoise_tv()`
- `cdl.computation.image.restoration.compute_denoise_bilateral()`
- `cdl.computation.image.restoration.compute_denoise_wavelet()`
- `cdl.computation.image.restoration.compute_denoise_tophat()`

**compute\_white\_tophat**(*param*: MorphologyParam | *None* = *None*) → *None*  
Compute White Top-Hat with `cdl.computation.image.morphology.compute_white_tophat()`

**compute\_black\_tophat**(*param*: MorphologyParam | *None* = *None*) → *None*  
Compute Black Top-Hat with `cdl.computation.image.morphology.compute_black_tophat()`

**compute\_erosion**(*param*: MorphologyParam | *None* = *None*) → *None*  
Compute Erosion with `cdl.computation.image.morphology.compute_erosion()`

**compute\_dilation**(*param*: MorphologyParam | *None* = *None*) → *None*  
Compute Dilation with `cdl.computation.image.morphology.compute_dilation()`

**compute\_opening**(*param*: MorphologyParam | *None* = *None*) → *None*  
Compute morphological opening with `cdl.computation.image.morphology.compute_opening()`

**compute\_closing**(*param*: MorphologyParam | *None* = *None*) → *None*  
Compute morphological closing with `cdl.computation.image.morphology.compute_closing()`

**compute\_all\_morphology**(*param*: MorphologyParam | *None* = *None*) → *None*  
Compute all morphology filters using the following functions:

- `cdl.computation.image.morphology.compute_white_tophat()`
- `cdl.computation.image.morphology.compute_black_tophat()`
- `cdl.computation.image.morphology.compute_erosion()`
- `cdl.computation.image.morphology.compute_dilation()`
- `cdl.computation.image.morphology.compute_opening()`
- `cdl.computation.image.morphology.compute_closing()`

**compute\_canny**(*param*: CannyParam | *None* = *None*) → *None*  
Compute Canny filter with `cdl.computation.image.edges.compute_canny()`

**compute\_roberts**() → *None*  
Compute Roberts filter with `cdl.computation.image.edges.compute_roberts()`



**compute\_prewitt()** → None

Compute Prewitt filter with `cdl.computation.image.edges.compute_prewitt()`

**compute\_prewitt\_h()** → None

Compute Prewitt filter (horizontal) with `cdl.computation.image.edges.compute_prewitt_h()`

**compute\_prewitt\_v()** → None

Compute Prewitt filter (vertical) with `cdl.computation.image.edges.compute_prewitt_v()`

**compute\_sobel()** → None

Compute Sobel filter with `cdl.computation.image.edges.compute_sobel()`

**compute\_sobel\_h()** → None

Compute Sobel filter (horizontal) with `cdl.computation.image.edges.compute_sobel_h()`

**compute\_sobel\_v()** → None

Compute Sobel filter (vertical) with `cdl.computation.image.edges.compute_sobel_v()`

**compute\_scharr()** → None

Compute Scharr filter with `cdl.computation.image.edges.compute_scharr()`

**compute\_scharr\_h()** → None

Compute Scharr filter (horizontal) with `cdl.computation.image.edges.compute_scharr_h()`

**compute\_scharr\_v()** → None

Compute Scharr filter (vertical) with `cdl.computation.image.edges.compute_scharr_v()`

**compute\_farid()** → None

Compute Farid filter with `cdl.computation.image.edges.compute_farid()`

**compute\_farid\_h()** → None

Compute Farid filter (horizontal) with `cdl.computation.image.edges.compute_farid_h()`

**compute\_farid\_v()** → None

Compute Farid filter (vertical) with `cdl.computation.image.edges.compute_farid_v()`

**compute\_laplace()** → None

Compute Laplace filter with `cdl.computation.image.edges.compute_laplace()`

**compute\_all\_edges()** → None

Compute all edges filters using the following functions:

- `cdl.computation.image.edges.compute_roberts()`
- `cdl.computation.image.edges.compute_prewitt()`
- `cdl.computation.image.edges.compute_prewitt_h()`
- `cdl.computation.image.edges.compute_prewitt_v()`
- `cdl.computation.image.edges.compute_sobel()`
- `cdl.computation.image.edges.compute_sobel_h()`
- `cdl.computation.image.edges.compute_sobel_v()`
- `cdl.computation.image.edges.compute_scharr()`
- `cdl.computation.image.edges.compute_scharr_h()`
- `cdl.computation.image.edges.compute_scharr_v()`
- `cdl.computation.image.edges.compute_farid()`

- `cdl.computation.image.edges.compute_farid_h()`
- `cdl.computation.image.edges.compute_farid_v()`
- `cdl.computation.image.edges.compute_laplace()`

**compute\_stats()** → dict[str, *ResultProperties*]

Compute data statistics with `cdl.computation.image.compute_stats()`

**compute\_centroid()** → dict[str, *ResultShape*]

Compute image centroid with `cdl.computation.image.compute_centroid()`

**compute\_enclosing\_circle()** → dict[str, *ResultShape*]

Compute minimum enclosing circle with `cdl.computation.image.compute_enclosing_circle()`

**compute\_peak\_detection()**(*param*: *Peak2DDetectionParam* | *None* = *None*) → dict[str, *ResultShape*]

Compute 2D peak detection with `cdl.computation.image.compute_peak_detection()`

**compute\_contour\_shape()**(*param*: *ContourShapeParam* | *None* = *None*) → dict[str, *ResultShape*]

Compute contour shape fit with `cdl.computation.image.detection.compute_contour_shape()`

**compute\_hough\_circle\_peaks()**(*param*: *HoughCircleParam* | *None* = *None*) → dict[str, *ResultShape*]

Compute peak detection based on a circle Hough transform with `cdl.computation.image.compute_hough_circle_peaks()`

**compute\_blob\_dog()**(*param*: *BlobDOGParam* | *None* = *None*) → dict[str, *ResultShape*]

Compute blob detection using Difference of Gaussian method with `cdl.computation.image.detection.compute_blob_dog()`

**compute\_blob\_doh()**(*param*: *BlobDOHParam* | *None* = *None*) → dict[str, *ResultShape*]

Compute blob detection using Determinant of Hessian method with `cdl.computation.image.detection.compute_blob_doh()`

**compute\_blob\_log()**(*param*: *BlobLOGParam* | *None* = *None*) → dict[str, *ResultShape*]

Compute blob detection using Laplacian of Gaussian method with `cdl.computation.image.detection.compute_blob_log()`

**compute\_blob\_opencv()**(*param*: *BlobOpenCVParam* | *None* = *None*) → dict[str, *ResultShape*]

Compute blob detection using OpenCV with `cdl.computation.image.detection.compute_blob_opencv()`

## 3.14 Docks

The `cdl.core.gui.docks` module provides the dockable widgets for the DataLab main window.

### 3.14.1 Plot widget

**class** `cdl.core.gui.docks.DataLabPlotWidget`(*plot\_type*: *PlotType*)

DataLab PlotWidget

This class is a subclass of `plotpy.plot.PlotWidget` that provides a customized widget for DataLab, with a specific set of tools and a customized appearance.

#### Parameters

**plot\_type** – Plot type



**register\_tools()** → [None](#)

Register the plotting tools according to the plot type

### 3.14.2 Dockable plot widget

**class** `cdl.core.gui.docks.DockablePlotWidget`(*parent: QWidget, plot\_type: PlotType*)

Docked plotting widget

#### Parameters

- **parent** – Parent widget
- **plot\_type** – Plot type

**setup\_layout()** → [None](#)

Setup layout

**update\_toolbar\_position()** → [None](#)

Update toolbar position

**setup\_plotwidget()** → [None](#)

Setup plotting widget

**update\_color\_mode()** → [None](#)

Update plot widget styles according to application color mode

**get\_plot()** → [BasePlot](#)

Return plot instance

**update\_watermark(plot: BasePlot)** → [None](#)

Update watermark visibility

**visibility\_changed(enable: bool)** → [None](#)

DockWidget visibility has changed

## 3.15 HDF5 I/O

The `cdl.core.gui.h5io` module provides the HDF5 file open/save into/from DataLab data model/main window.

**class** `cdl.core.gui.h5io.H5InputOutput`(*mainwindow: CDLMainWindow*)

Object handling HDF5 file open/save into/from DataLab data model/main window

#### Parameters

**mainwindow** – Main window

**save\_file(filename: str)** → [None](#)

Save all signals and images from DataLab model into a HDF5 file

**open\_file(filename: str, import\_all: bool, reset\_all: bool)** → [None](#)

Open HDF5 file

**import\_files(filename: list[str], import\_all: bool, reset\_all: bool)** → [None](#)

Import HDF5 files

**import\_dataset\_from\_file(filename: str, dsetname: str)** → [None](#)

Import dataset from HDF5 file



## CONTRIBUTING

DataLab is **your** platform. If you want it to be improved, you **can** contribute to the project, whether you are a developer or not.

There are many ways to contribute to DataLab project, depending on how much time you have, your experience with open source projects, and your skills.

### 4.1 Share your ideas and experiences

Besides the classic bug reports and feature requests, you can share your ideas and experiences for improving DataLab. In particular, we are very interested in your feedback on the documentation and tutorials. Moreover, if you have a use case that you would like to share with the community, please let us know.

Without coding, you can contribute to DataLab project by:

- [Reporting a bug](#)
- [Suggesting an enhancement](#)
- [Suggesting a documentation topic](#)
- [Suggesting a tutorial topic](#)

### 4.2 Share your scientific/technical knowledge

Your technical or scientific knowledge is also very valuable to us, as you may directly contribute to the documentation or tutorials. Or, better, if you want to write a tutorial, we will be happy to help you.

Again without coding, you can contribute to DataLab project by:

- [Writing documentation](#)
- [Writing a tutorial](#)

## 4.3 Contribute to new features

Even if you are not a developer, you can contribute to the project by:

- Testing new features
- Writing and submitting new Plugins
- Writing and submitting macro-commands

## 4.4 Develop new features

If you are a developer, you can contribute to the core of the project by fixing bugs or implementing new features.

See [Contribute code](#) section for more information.

### 4.4.1 Contribute code

This page explains how you can contribute code to the project.

**See also:**

The [Coding guidelines](#) page presents the coding guidelines that you should follow when contributing to the project.

### Fork the project

The first step is to fork the project on GitHub. You can do that by visiting [DataLab GitHub project](#) and clicking on the “Fork” button on the top right corner of the page.

Once you have forked the project, you will have a copy of the project in your own GitHub account. Then you can clone the project on your computer and start working on it.

### Submit a pull request

Once you have made some changes, you can submit a pull request to the original project. To do that, go to your forked project on GitHub and click on the “Pull request” button on the top right corner of the page.

Then you will have to fill a form to describe your pull request. Once you have submitted the pull request, the project maintainers will review your changes and merge them if they are satisfied.

During the review process, the project maintainers will check that your code follows the coding guidelines and that it does not break the existing tests. If your code does not follow the coding guidelines, you will have to fix it before your pull request can be merged.

## 4.4.2 Coding guidelines

### Generic coding guidelines

We follow the [PEP 8](#) coding style.

In particular, we are especially strict about the following guidelines:

- Limit all lines to a maximum of 79 characters.
- Respect the naming conventions (classes, functions, variables, etc.).
- Use specific exceptions instead of the generic [Exception](#).

To enforce these guidelines, the following tools are mandatory:

- [ruff](#) for code formatting and static code analysis.
- [pylint](#) for static code analysis.

#### ruff

If you are using [Visual Studio Code](#), the project settings will automatically format your code with *ruff* on save (you may also run the task “Run Ruff” to run *ruff* on the project).

To run *ruff*, run the following command:

```
ruff check
```

#### pylint

To run *pylint*, run the following command:

```
pylint datalab
```

If you are using [Visual Studio Code](#) on Windows, you may run the task “Run Pylint” to run *pylint* on the project.

---

**Note:** A *pylint* rating greater than 9/10 is required to merge a pull request.

---

### Specific coding guidelines

In addition to the generic coding guidelines, we have the following specific guidelines:

- Write docstrings for all classes, methods and functions. The docstrings should follow the [Google style](#).
- Add typing annotations for all functions and methods. The annotations should use the future syntax (`from __future__ import annotations`)
- Try to keep the code as simple as possible. If you have to write a complex piece of code, try to split it into several functions or classes.
- Add as many comments as possible. The code should be self-explanatory, but it is always useful to add some comments to explain the general idea of the code, or to explain some tricky parts.
- Do not use `from module import *` statements, even in the `__init__` module of a package.

- Avoid using mixins (multiple inheritance) when possible. It is often possible to use composition instead of inheritance.
- Avoid using `__getattr__` and `__setattr__` methods. They are often used to implement lazy initialization, but this can be done in a more explicit way.

### 4.4.3 Git Workflow

This document describes the Git workflow used in the DataLab project, based on a `main` branch, a `develop` branch, and feature-specific branches. It also defines how bug fixes are managed.

---

**Note:** This workflow is a simplified version of the [Gitflow Workflow](#). It has been adapted to suit the needs of the DataLab project at the current stage of development. In the near future, we may consider adopting a more complex workflow, e.g. by adding release branches.

---

## Branching Model

### Main Branches

- `main`: Represents the stable, production-ready version of the project.
- `develop`: Used for ongoing development and integration of new features.

### Feature Branches

- `develop/feature_name`: Used for the development of new features.
  - Created from `develop`.
  - Merged back into `develop` once completed.
  - Deleted after merging.

### Bug Fix Branches

- `fix/xxx`: Used for general bug fixes that are not urgent.
  - Created from `develop`.
  - Merged back into `develop` once completed.
  - Deleted after merging.
- `hotfix/xxx`: Used for urgent production-critical fixes.
  - Created from `main`.
  - Merged back into `main`.
  - The fix is then cherry-picked into `develop`.
  - Deleted after merging.

---

**Note:** Hotfixes (high-priority fixes) will be integrated in the next maintenance release (X.Y.Z -> Z+1), while fixes (low-priority fixes) will be integrated in the next feature release (X.Y -> Y+1).

---

### Workflow for New Features

1. Create a new feature branch from develop:

```
git checkout develop
git checkout -b develop/feature_name
```

2. Develop the feature and commit changes.
3. Merge the feature branch back into develop:

```
git checkout develop
git merge --no-ff develop/feature_name
```

4. Delete the feature branch:

```
git branch -d develop/feature_name
```

**Warning:** Do not leave feature branches unmerged for too long. Regularly rebase them on develop to minimize conflicts.

### Workflow for Regular Bug Fixes

1. Create a bug fix branch from develop:

```
git checkout develop
git checkout -b fix/bug_description
```

2. Apply the fix and commit changes.
3. Merge the fix branch back into develop:

```
git checkout develop
git merge --no-ff fix/bug_description
```

4. Delete the fix branch:

```
git branch -d fix/bug_description
```

**Warning:** Do not create a fix/xxx branch from a develop/feature\_name branch. Always branch from develop to ensure fixes are correctly propagated.

```
# Incorrect:
git checkout develop/feature_name
git checkout -b fix/wrong_branch
```

```
# Correct:
git checkout develop
git checkout -b fix/correct_branch
```

## Workflow for Critical Hotfixes

1. Create a hotfix branch from main:

```
git checkout main
git checkout -b hotfix/critical_bug
```

2. Apply the fix and commit changes.
3. Merge the fix back into main:

```
git checkout main
git merge --no-ff hotfix/critical_bug
```

4. Cherry-pick the fix into develop:

```
git checkout develop
git cherry-pick <commit_hash>
```

5. Delete the hotfix branch:

```
git branch -d hotfix/critical_bug
```

**Warning:** Do not merge `fix/xxx` or `hotfix/xxx` directly into `main` without following the workflow. Ensure hotfixes are cherry-picked into `develop` to avoid losing fixes in future releases.

## Best Practices

- Regularly **rebase feature branches** on `develop` to stay up to date:

```
git checkout develop/feature_name
git rebase develop
```

- Avoid long-lived branches to minimize merge conflicts.
- Ensure bug fixes in `main` are **always cherry-picked** to `develop`.
- Clearly differentiate between `fix/xxx` (non-urgent fixes) and `hotfix/xxx` (critical production fixes).



## Takeaway

This workflow ensures a structured yet flexible development process while keeping `main` stable and `develop` always updated with the latest changes.

It also ensures that bug fixes are correctly managed and propagated across branches.

### 4.4.4 Setting up Development Environment

Getting started with DataLab development is easy.

Here is what you will need:

1. An integrated development environment (IDE) for Python. We recommend [Spyder](#) or [Visual Studio Code](#), but any IDE will do.
2. A Python distribution. We recommend [WinPython](#), on Windows, or [Anaconda](#), on Linux or Mac. But, again, any Python distribution will do.
3. A clean project structure (see below).
4. Test data (see below).
5. Environment variables (see below).
6. Third-party software (see below).

## Development Environment

If you are using [Spyder](#), thank you for supporting the scientific open-source Python community!

If you are using Visual Studio Code, that's also an excellent choice (for other reasons). We recommend installing the following extensions:

Extension	Description
<a href="#">gettext</a>	Gettext syntax highlighting
<a href="#">Pylance</a>	Python language server
<a href="#">Python</a>	Python extension
<a href="#">reStructuredText Syntax highlighting</a>	reStructuredText syntax highlighting
<a href="#">Ruff</a>	Extremely fast Python linter and code formatter
<a href="#">Todo Tree</a>	Todo tree
<a href="#">Insert GUID</a>	Insert GUID
<a href="#">XML Tools</a>	XML Tools

## Python Environment

DataLab requires the following :

- Python (e.g. WinPython)
- Additional Python packages

Installing all required packages :

```
pip install --upgrade -r dev\requirements.txt
```

See [Installation](#) for more details on reference Python and Qt versions.

If you are using [WinPython](#), thank you for supporting the scientific open-source Python community!

The following table lists the currently officially used Python distributions:

Python version	Status	WinPython version
3.9	OK	3.9.10.0
3.10	OK	3.10.11.1
3.11	OK	3.11.5.0
3.12	OK	3.12.3.0
3.13	OK	3.13.2.0

We strongly recommend using the `.dot` versions of WinPython which are lightweight and can be customized to your needs (using `pip install -r requirements.txt`).

We also recommend using a dedicated WinPython instance for DataLab.

## Test data

DataLab test data are located in different folders, depending on their nature or origin.

Required data for unit tests are located in “`cdl\data\tests`” (public data).

A second folder `%CDL_DATA%` (optional) may be defined for additional tests which are still under development (or for confidential data).

## Specific environment variables

Enable the “debug” mode (no stdin/stdout redirection towards internal console):

```
@REM Mode DEBUG
set DEBUG=1
```

Building PDF documentation requires LaTeX. On Windows, the following environment:

```
@REM LaTeX executable must be in Windows PATH, for mathematical equations rendering
@REM Example with MiKTeX :
set PATH=C:\\Apps\\miktex-portable\\texmf\\install\\miktex\\bin\\x64;%PATH%
```

Visual Studio Code configuration used in `launch.json` and `tasks.json` (examples) :

```
@REM Development environment
set CDL_PYTHONEXE=C:\\python-3.9.10.amd64\\python.exe
@REM Folder containing additional working test data
set CDL_DATA=C:\\Dev\\Projets\\CDL_data
```

Visual Studio Code `.env` file:

- This file is used to set environment variables for the application.
- It is used to set the `PYTHONPATH` environment variable to the root of the project.
- This is required to be able to import the project modules from within VS Code.
- To create this file, copy the `.env.template` file to `.env` (and eventually add your own paths).

## Windows installer

The Windows installer is built using [WiX Toolset V4.0.5](#). Using the WiX Toolset requires [.NET SDK V6.0](#) minimum. You may install .NET SDK using `winget`:

```
winget install Microsoft.DotNet.SDK.8
```

Once .NET SDK is installed, the WiX Toolset can be installed and configured using the following commands:

```
dotnet tool install --global wix --version 4.0.5
wix extension add WixToolset.UI.wixext/4.0.5
```

First, you need to generate the installer script from a generic template:

```
python wix/makewxs.py
```

Building the installer is done using the following command:

```
wix build .\wix\DataLab.wxs -ext WixToolset.UI.wixext
```

## Third-party Software

The following software may be required for maintaining the project:

Software	Description
<a href="#">gettext</a>	Translations
<a href="#">Git</a>	Version control system
<a href="#">ImageMagick</a>	Image manipulation utilities
<a href="#">Inkscape</a>	Vector graphics editor
<a href="#">MikTeX</a>	LaTeX distribution on Windows

## 4.4.5 Roadmap

This document outlines the current and future development plans for DataLab:

- It includes information about *funding sources*, *planned milestones*, and *future evolution*.
- It also provides a summary of *past milestones* to give context to the project's evolution.

## Funding

As an open-source project, DataLab relies on the support of various organizations and grants to fund its development and maintenance. The project's roadmap is shaped by the needs and priorities of its users, as well as the availability of resources. Funded work is prioritized and scheduled accordingly, while community contributions are welcomed and encouraged.

From the project's inception in 2023, the development of DataLab has been funded by the following grants and organizations:

Func ing	Description
	<p><a href="#">CEA</a> - French Alternative Energies and Atomic Energy Commission:• DataLab was initially created for analyzing data from CEA's Laser Megajoule (LMJ) facility• Interfaced with the LMJ control system, DataLab is used to process and visualize signals and images acquired with plasma diagnostics (devices such as cameras, digitizers, spectrometers, etc.)• It is also used for R&amp;D activities around the LMJ facility (e.g., metrology, data analysis, etc.)• CEA is the major investor in DataLab and the main contributor to the project: CEA scientists and engineers are actively involved in the roadmap</p> <p><a href="#">CODRA</a>, a software engineering company and software publisher, has supported DataLab's open-source journey since its inception:• Open-source project management and communication (social media, website, etc.)• Conferences and events: <a href="#">SciPy 2024</a>, <a href="#">PyData Paris 2024</a>, <a href="#">Open Source Experience 2024</a>, etc.• Documentation: tutorials, videos, and more</p> <p><a href="#">NLnet Foundation</a>, as part of the NGI0 Commons Fund backed by the European Commission, funded the <a href="#">redesign of DataLab's core architecture</a> — a major overhaul scheduled for inclusion in the 2.0 release (December 2025):• The goal is to decouple the data model, computation, and I/O from the UI• This will enable DataLab to be used as a library in other software</p>

## Planned Milestones

The following tasks are planned for future releases of DataLab. The timeline and specific details may change based on user feedback, funding availability, and other factors.

This section outlines the planned features and enhancements for DataLab, along with their expected release dates.

Those features and enhancements are funded by the following organizations (see [Funding](#) for more details):

- [CEA](#): French Alternative Energies and Atomic Energy Commission
- [CODRA](#): software engineering company and software publisher
- [NLnet](#): NLnet Foundation, as part of the NGI0 Commons Fund

*Note: The milestones and dates presented below are indicative and subject to change.*

Mile- stone	Description
2.0 2025/	This release introduces the <a href="#">redesign of DataLab's core architecture</a> :• Core: decoupling data model, computation & I/O from UI• Validation: full migration of test infra, automated & manual testing• Docs & Training: installation guides, API docs, user manuals, onboarding materials
1.020	This release consolidates the features introduced in the V0.x series, and also integrates:• Common: Fourier tools, noise generation, multi-file saving, ...• Image: background subtraction, local smoothing, filtering, sub-image extraction, resampling, ...• Signal: new formats (.SIG, .IMA), signal generators, curve fitting, advanced filtering, ...

## Future Milestones

The following tasks are long-term goals for DataLab. They are not scheduled for any specific release and may evolve over time based on user needs and project direction.

The following table summarizes the future evolutions and maintenance plans for DataLab that are detailed in the sections below.

Mile-stone type	Description
Future Evolutions	<ul style="list-style-type: none"> <li>• Support for data acquisition</li> <li>• Web frontend</li> <li>• Support for time series</li> <li>• Database connectors</li> <li>• Jupyter plugin for interactive data analysis</li> <li>• Spyder plugin for interactive data analysis</li> <li>• Jupyter kernel interface for DataLab</li> </ul>
Maintenance	<ul style="list-style-type: none"> <li>• Transition to gRPC for remote control</li> <li>• Drop Qt5 support and migrate to Qt6</li> </ul>
Other Tasks	<ul style="list-style-type: none"> <li>• Create a DataLab plugin template</li> </ul>

## Support for Data Acquisition

Adding support for data acquisition would be a major step forward in making DataLab not only a platform for signal and image processing, but also a **versatile tool for real-time experimental workflows**. The idea would be to allow users to acquire data directly from various hardware devices (e.g., cameras, digitizers, spectrometers, etc.) **within DataLab itself**.

Such a feature would enable **seamless integration between acquisition and analysis**, allowing users to process and visualize data immediately after it is captured, without needing to switch tools or export/import files.

While no formal design has been established yet, a viable approach could be to:

- Introduce a **new plugin family dedicated to data acquisition**, following the modular architecture of DataLab;
- Define a **generic API for acquisition plugins**, ensuring flexibility and compatibility across device types;
- **Leverage existing solutions** such as [PyMoDAQ](#), a Python-based data acquisition framework already compatible with a wide range of laboratory instruments.

A potential **collaboration with the PyMoDAQ development team** could be explored, to benefit from their ecosystem and avoid duplicating efforts. This would also help foster interoperability and promote open standards in the Python scientific instrumentation community.

## Web Frontend

As DataLab's modular architecture evolves, a natural next step is to provide a **web-based frontend** to complement the existing desktop application.

A web frontend would allow users to:

- **Run DataLab remotely** (e.g. from a server or cloud platform) and access it via a browser;
- Perform processing and visualization tasks without needing a local Python environment;
- Facilitate **collaborative data analysis**, sharing sessions or results with colleagues;
- Integrate with JupyterHub, dashboards, or lab management tools for centralized usage.

This frontend could be built on top of the upcoming DataLab-core library, exposing its features through a web interface — possibly leveraging tools like **JupyterLab extensions**, **Panel**, or **Dash**, depending on the chosen stack.

While still exploratory, this direction would increase **accessibility, portability, and scalability** of DataLab, especially in academic, industrial, and cloud-based environments.

## Support for Time Series

DataLab currently focuses on generic signal and image processing, but many use cases — especially in scientific instrumentation and experimental physics — involve **time series data**.

Adding dedicated support for time series would make it easier to:

- Handle signals with associated timestamps or non-uniform sampling;
- Perform time-aware processing and visualization (e.g., event alignment, time-based filtering, resampling);
- Integrate with external systems generating time-indexed data.

This feature is tracked in [Issue #27](#), where potential use cases and design considerations are discussed.

Introducing robust time series handling would broaden DataLab’s applicability in domains such as data logging, slow control, and real-time monitoring.

## Database Connectors

Currently, DataLab operates primarily on files and in-memory data structures. Adding **connectors to databases** would significantly extend its capabilities, especially for users dealing with large, structured, or historical datasets.

This feature would allow DataLab to:

- **Query and load data** from SQL or NoSQL databases (e.g. PostgreSQL, SQLite, MongoDB, etc.);
- Support metadata-driven workflows, where experiments or datasets are referenced from a database;
- **Store analysis results** or annotations back into a database for traceability and reproducibility;
- Facilitate integration into lab information management systems (LIMS) or enterprise data infrastructures.

The design could include:

- A **plugin-based system** for supporting various database backends;
- A simple configuration interface for connection settings and query management;
- Integration with pandas or SQLAlchemy for flexible data exchange.

This evolution would help bridge the gap between **data acquisition, analysis, and long-term storage**, enabling more robust scientific data workflows.

## Jupyter Plugin for Interactive Data Analysis

Although DataLab can already be remotely controlled from a Jupyter notebook — thanks to its existing remote control capabilities (see [Remote controlling](#)) — the user experience could be greatly enhanced by developing a **dedicated Jupyter plugin**.

This plugin would provide a **tighter integration** between Jupyter and DataLab, offering the following features:

- Use DataLab as a **Jupyter kernel**, enabling direct access to its processing capabilities from within a notebook;

- **Display numerical results from DataLab in Jupyter**, and vice versa — for example, importing results computed in a Jupyter notebook into the DataLab interface;
- Allow **interactive data manipulation**: use DataLab for efficient signal and image operations, and Jupyter for custom or home-made data analysis routines;
- Bridge scripting and GUI workflows, making DataLab more attractive for scientific users familiar with Jupyter environments.

Technically, this plugin could rely on the **Jupyter kernel interface** to expose DataLab’s capabilities in an interactive programming context.

Such integration would reinforce DataLab’s role in the scientific Python ecosystem and facilitate reproducible, notebook-driven analysis workflows.

### Spyder Plugin for Interactive Data Analysis

A plugin for the [Spyder IDE](#) would address the **same use cases as the Jupyter plugin**, providing seamless integration between DataLab and Spyder’s interactive environment.

This plugin would allow users to:

- Interact with DataLab directly from Spyder;
- Visualize or send data between the DataLab GUI and the Spyder console;
- Use DataLab’s processing capabilities in real time while scripting custom analysis workflows in Spyder.

As with the Jupyter integration, this plugin could be implemented by leveraging the **Jupyter kernel interface** (see [Remote controlling](#)), which Spyder already supports internally.

Such a plugin would enhance DataLab’s usability for scientists and engineers who prefer Spyder’s integrated development environment for exploratory analysis.

### Jupyter Kernel Interface for DataLab

Implementing a native **Jupyter kernel interface** for DataLab would provide a more integrated way to use it from other environments such as **Jupyter notebooks, Spyder, or Visual Studio Code**.

This interface would allow:

- Direct control of DataLab from third-party tools that support Jupyter kernels;
- **Two-way data exchange**: e.g., displaying DataLab results inside a notebook, or visualizing Jupyter-generated data inside the DataLab GUI;
- Tighter integration into scripting workflows and IDEs beyond simple remote control.

However, based on initial exploration, implementing a full Jupyter kernel may be **non-trivial and potentially time-consuming**. Given that remote control already enables communication between DataLab and Jupyter (see [Remote controlling](#)), the added value of a full kernel integration should be carefully evaluated against its complexity and maintenance cost.

This option remains open for discussion depending on user demand and development resources.

## Maintenance Plan

### 2026: Transition to gRPC for Remote Control

DataLab currently relies on XML-RPC for remote control, which may become a limitation for more advanced or high-performance use cases. If the need arises for a more **efficient, robust, and extensible communication protocol**, a switch to **gRPC** is under consideration.

This improvement is tracked in [Issue #18](#).

### 2025: Drop Qt5 Support and Migrate to Qt6

With the **end-of-life of Qt5 scheduled for mid-2025**, DataLab will fully migrate to **Qt6**. This transition is expected to be **straightforward**, thanks to:

- The use of the `qtpy` abstraction layer;
- The fact that the `PlotPyStack` library is already compatible with Qt6.

This change will ensure compatibility with future versions of Qt and modern Python environments.

## Other Tasks

### Create a DataLab Plugin Template

To encourage community contributions and facilitate the development of extensions, a **template for creating DataLab plugins** is planned.

This template would:

- Provide a ready-to-use scaffold with best practices;
- Help new developers quickly understand the plugin architecture;
- Promote consistency and modularity across third-party plugins.

The task is tracked in [Issue #26](#).

## Past Milestones

From version 0.9 to 0.19, DataLab has undergone significant development and enhancements. The project has evolved from a simple data analysis tool to a powerful platform for processing and visualizing signals and images.

Those enhancements have been made possible thanks to the support of the following organizations (see [Funding](#) for more details):

- **CEA**: French Alternative Energies and Atomic Energy Commission
- **CODRA**: software engineering company and software publisher

The following table summarizes the major past milestones of DataLab, including the release dates and a brief description of the features or enhancements introduced in each version.



Mile-stone	Description
0.1920:	• Open all signals or images from a folder (recursively)• ROI editor: add options to create ROIs from coordinates
0.1820:	• New pairwise operand mode (the operation is done on each pair of signals/images)• New polygonal ROI feature• Support Windows 7 SP1 to Windows 11 with a single installer
0.1720:	• Introduce ROI support across all processing features• Add arithmetic operations on signals and images• New Ubuntu package for native installation on Linux• New Conda package for all platforms (Windows, Linux, MacOS)
0.1620:	• New validation process for signal and image features• Add support for binary images (unlocking new processing features)
0.1520:	• New MSI installer for the stand-alone version on Windows• Add support for large text/CSV files (> 1 GB)• Add auto downsampling feature
0.1420:	• HDF5 browser: multiple file support, detailed info on groups and attributes• New Debian package for native installation on Linux
0.1220:	• Add a tour and demo feature• Add tutorials to the documentation• Add a Text file import assistant
0.1120:	• Add features for reordering signals and images (e.g. drag-and-drop)• Add 1D convolution, interpolation, resampling and detrending features
0.1020:	• Develop a very simple DataLab plugin to demonstrate the plugin system• Allow to disable the automatic refresh when doing multiple processing steps• Serialize curve and image styles in HDF5 files• Improve curve readability
12	
0.9202:	• Run computations in a separate process• Add a plugin system: API for third-party extensions• Add a macro-command system• Add an XML-RPC server to allow DataLab remote control
11	

## 4.4.6 Changelog

See DataLab [roadmap page](#) for future and past milestones.

### DataLab Version 0.20.0

New features and enhancements:

- ANDOR SIF Images:
  - Added support for background images in ANDOR SIF files
  - This closes [Issue #178](#) - Add support for ANDOR SIF files with background image
- Array editor (results, signal and image data, ...):
  - New “Copy all” button in the array editor dialog box, to copy all the data in the clipboard, including row and column headers
  - New “Export” button in the array editor dialog box, to export the data in a CSV file, including row and column headers
  - New “Paste” button in the array editor dialog box, to paste the data from the clipboard into the array editor (this feature is not available for read-only data, such as analysis results)
  - The features above require guidata v3.9.0 or later
  - This closes [Issue #174](#), [Issue #175](#) and [Issue #176](#)
- Fourier analysis features (“Processing” menu):
  - New “Zero padding” feature

- Implementation for signals:
  - \* Choose a zero padding strategy (Next power of 2, Double the length, Triple the length, Custom length)
  - \* Or manually set the zero padding length (if “Custom length” is selected)
- Implementation for images:
  - \* Choose a zero padding strategy (Next power of 2, Next multiple of 64, Custom length)
  - \* Or manually set the zero padding row and column lengths (if “Custom length” is selected)
  - \* Set the position of the zero padding (bottom-right, centered)
- This closes [Issue #170](#) - Fourier analysis: add zero padding feature for signals and images
- Region of Interest (ROI) editor:
  - This concerns the “Edit Regions of Interest” feature for both signals and images
  - New behavior:
    - \* Signals: the range ROI selection tool is now active by default, and the user can select right away the range of the signal to be used as a ROI
    - \* Images: the rectangular ROI selection tool is now active by default, and the user can select right away the rectangular ROI to be used as a ROI
    - \* This closes [Issue #154](#) - ROI editor: activate ROI selection tool by default, so that the user can select right away the area to be used as a ROI
  - Added the “Select tool” to editor’s toolbar, to allow the user to switch between the “Select” and “Draw” tools easily without having to use the plot toolbar on the top of the window
- Signal processing features (“Processing” menu):
  - New “X-Y mode” feature: this feature simulates the behavior of the X-Y mode of an oscilloscope, i.e. it allows to plot one signal as a function of another signal (e.g. X as a function of Y)
  - New abscissa and ordinate find features:
    - \* “First abscissa at y=...” feature: this feature allows to find the first abscissa value of a signal at a given y value (e.g. the abscissa value of a signal at y=0)
    - \* “Ordinate at x=...” feature: this feature allows to find the ordinate value of a signal at a given x value (e.g. the ordinate value of a signal at x=0)
    - \* Each feature has its own dialog box, which allows to set the y or x value to be used for the search with a slider or a text box
    - \* This closes [Issue #125](#) and [Issue #126](#)
  - New full width at given y feature:
    - \* The “Full width at y=...” feature allows to find the full width of a signal at a given y value (e.g. the full width of a signal at y=0)
    - \* A specific dialog box allows to set the y value to be used for the search with a slider or a text box
    - \* This closes [Issue #127](#)
- Public API (local or remote):
  - Add `group_id` and `set_current` arguments to `add_signal`, `add_image` and `add_object` methods:
    - \* This concerns the `LocalProxy`, `AbstractCDLControl`, `RemoteClient`, `RemoteServer` and `CDLMainWindow` classes

- \* `group_id` argument allows to specify the group ID where the signal or image should be added (if not specified, the signal or image is added to the current group)
- \* `set_current` argument allows to specify if the signal or image should be set as current after being added (default is `True`)
- \* This closes [Issue #151](#) - Public API: add a keyword `group_id` to `add_signal` and `add_image`

## DataLab Version 0.19.2

Bug fixes:

- Fixed [Issue #172](#) - Image profiles: when moving/resizing image, profile plots are not refreshed (fixed in PlotPy v2.7.4)
- Fixed [Issue #173](#) - Phase spectrum: add unit (degree) and function reference (`numpy.angle`) to the documentation
- Fixed [Issue #177](#) - “Open from directory” feature: unexpected group name (a group named “.” is created instead of the root folder name)
- Fixed [Issue #169](#) - Signal / Fourier analysis: magnitude spectrum feature does not work as expected with logarithmic scale enabled
- Fixed [Issue #168](#) - Average profile visualization: empty profile is displayed when the target rectangular area is outside the image area (this has been fixed upstream, in PlotPy v2.7.4, and so requires the latest version of PlotPy)

## DataLab Version 0.19.1

Bug fixes:

- Pairwise operation mode:
  - Fixed an unexpected behavior when using the pairwise operation mode with functions that take a single second operand (e.g. for images: difference, division, arithmetic operations, and flatfield correction)
  - If only one set of operands was selected in a single group, a warning message was displayed “In pairwise mode, you need to select objects in at least two groups.”, which is correct for functions that are symmetric (e.g. addition, multiplication, etc.), but not for functions that are not symmetric (e.g. difference, division, etc.).
  - This is now fixed: the warning message is only displayed for functions that are symmetric (e.g. addition, multiplication, etc.).
  - This closes [Issue #157](#) - Pairwise operation mode: unexpected behavior with functions that take a single second operand
- Fixed [Issue #152](#) - Ignore nan values for image normalization, flatfield correction, offset correction, and centroid computation
- Fixed [Issue #153](#) - Ignore nan values for signal normalization and statistics computations (both analysis result and interactive tool)
- Fixed [Issue #158](#) - When editing ROI of a list of images, the first image of the selection is shown (instead of the last as in the image panel)
- Fixed [Issue #159](#) - When selecting multiple images just after opening an HDF5 file, the “View in a new window” feature does not work (`KeyError` exception)
- Fixed [Issue #160](#) - When selecting multiple images and clearing ROI in ROI editor, only the first image is affected
- Fixed [Issue #161](#) - Refresh image items only if necessary (when editing ROI, pasting/deleting metadata)

- Fixed [Issue #162](#) - View in a new window: when displaying multiple images, the item list panel should be visible
- Fixed [Issue #163](#) - Open from directory: expected one group per folder when loading multiple files
- Fixed [Issue #164](#) - Open from directory: unsupported files should be ignored when loading files recursively, to avoid warning popup dialog boxes
- Fixed [Issue #165](#) - When opening a file, the default signal/image title must be set to the file name, instead of the relative path to the file name

## DataLab Version 0.19.0

New features and enhancements:

- Image operation features (“Operations” menu):
  - Renamed “Rotation” submenu to “Flip or rotation”
  - New “Flip diagonally” feature
- Signal processing features (“Processing” menu):
  - New “Convert to Cartesian coordinates” feature
  - New “Convert to polar coordinates” feature
- Signal analysis features (“Analysis” menu):
  - Renamed “X values at min/max” to “Abscissa of the minimum and maximum”
  - New “Abscissa at y=...” feature
- New “Open from directory” feature:
  - This feature allows to open multiple files from a directory at once, recursively (only the files with the supported extensions by the current panel are opened)
  - Add “Open from directory” action to the “File” menu for both Signal and Image panels
  - Add support for folders when dropping files in the Signal and Image panels
- Add 1/x operation to the “Operations” menu for both Signal and Image panels:
  - This feature relies on the `numpy.reciprocal` function, and handles the case where the denominator is zero by catching warnings and replacing the `np.inf` values with `np.nan` values
  - Add `compute_inverse` method for image and signal processors
  - This closes [Issue #143](#) - New feature: 1/x for signals and images
- Public API (local or remote):
  - Add `add_group` method with `title` and `select` arguments to create a new group in a data panel (e.g. Signal or Image panel) and eventually select it after creation:
    - \* Method was added to the following classes: `AbstractCDLControl`, `BaseDataPanel` and `RemoteClient`
    - \* This closes the following issues:
      - [Issue #131](#) - `BaseDataPanel.add_group`: add `select` argument
      - [Issue #47](#) - Remote proxy / Public API: add `add_group` method
  - `AbstractCDLControl.get_object_uuids`: add an optional `group` argument (group ID, title or number) to eventually filter the objects by group (this closes [Issue #130](#))

- When opening an HDF5 file, the confirmation dialog box asking if current workspace should be cleared has a new possible answer “Ignore”:
  - Choosing “Ignore” will prevent the confirmation dialog box from being displayed again, and will choose the current setting (i.e. clear or not the workspace) for all subsequent file openings
  - Added a new “Clear workspace before loading HDF5 file” option in the “Settings” dialog box, to allow the user to change the current setting (i.e. clear or not the workspace) for all subsequent file openings
  - Added a new “Ask before clearing workspace” option in the “Settings” dialog box, to allow the user to disable or re-enable the confirmation dialog box asking if current workspace should be cleared when opening an HDF5 file
  - This closes [Issue #146](#) - Ask before clearing workspace when opening HDF5 file: add “Ignore” option to prevent dialog from being displayed again
- Object and group title renaming:
  - Removed “Rename group” feature from the “Edit” menu and context menu
  - Added “Rename object” feature to the “Edit” menu and context menu, with F2 shortcut, to rename the title of the selected object or group
  - This closes [Issue #148](#) - Rename signal/image/group title by pressing F2
- Region of Interest editor:
  - Regrouped the graphical actions (new rectangular ROI, new circular ROI, new polygonal ROI) in a single menu “Graphical ROI”
  - Added new “Coordinate-based ROI” menu to create a ROI using manual input of the coordinates:
    - \* For signals, the ROI is defined by the start and end coordinates
    - \* For images:
      - The rectangular ROI is defined by the top-left and bottom-right coordinates
      - The circular ROI is defined by the center and radius coordinates
      - The polygonal ROI is not supported yet
    - \* This closes [Issue #145](#) - ROI editor: add manual input of the coordinates

#### Bug fixes:

- Fixed [Issue #141](#) - Image analysis: mask nan values when computing statistics, for example
- Fixed [Issue #144](#) - Average profile extraction: ValueError when selection rectangle is larger than the image

### DataLab Version 0.18.2

#### General information:

- Python 3.13 is now supported, since the availability of the scikit-image V0.25 (see [Issue #104](#) - Python 3.13: `KeyError: 'area_bbox'`)

#### Enhancements:

- Added new “Keep results after computation” option in “Processing” section:
  - Before this change, when applying a processing feature (e.g. a filter, a threshold, etc.) on a signal or an image, the analysis results were removed from the object

- This new option allows to keep the analysis results after applying a processing feature on a signal or an image. Even if the analysis results are not updated, they might be relevant in some use cases (e.g. when using the 2D peak detection feature on an image, and then applying a filter on the image, or summing two images, etc.)

**Bug fixes:**

- Fixed [Issue #138](#) - Image colormaps were no longer stored in metadata (and serialized in HDF5 files) since PlotPy v2.6.3 (this commit, specifically: [PlotPyStack/PlotPy@a37af8a](#))
- Fixed [Issue #137](#) - Arithmetic operations and signal interpolation: dialog box with parameters is not displayed
- Fixed [Issue #136](#) - When processing a signal or an image, the analysis result is kept from original object
  - Before this fix, when processing a signal or an image (e.g. when applying a filter, a threshold, etc.), the analysis result was kept from the original object, and was not updated with the new data. Thus the analysis result was not meaningful anymore, and was misleading the user.
  - This is now fixed: the analysis result is now removed when processing a signal or an image. However it is not recalculated automatically, because there is no way to know which analysis result should be recalculated (e.g. if the user has applied a filter, should the FWHM be recalculated?) - besides, the current implementation of the analysis features does not allow to recalculate the analysis results automatically when the data is modified. The user has to recalculate the analysis results manually if needed.
- Fixed [Issue #132](#) - Plot analysis results: “One curve per result title” mode ignores ROIs
  - Before this fix, the “One curve per result title” mode was ignoring ROIs, and was plotting the selected result for all objects (signals or images) without taking into account the ROI defined on the objects
  - This is now fixed: the “One curve per result title” mode now takes into account the ROI defined on the objects, and plots the selected result for each object (signal or image) and for each ROI defined on the object
- Fixed [Issue #128](#) - Support long object titles in Signal and Image panels
- Fixed [Issue #133](#) - Remove specific analysis results from metadata clipboard during copy operation
- Fixed [Issue #135](#) - Allow to edit ROI on multiple signals or images at once
  - Before this fix, the ROI editor was disabled when multiple signals or images were selected
  - This is now fixed: the ROI editor is now enabled when multiple signals or images are selected, and the ROI is applied to all selected signals or images (only the ROI of the first selected signal or image is taken into account)
  - This new behavior is consistent with the ROI extraction feature, which allows to extract the ROI on multiple signals or images at once, based on the ROI defined on the first selected signal or image
- Image ROI features:
  - Fixed [Issue #120](#) - ROI extraction on multiple images: defined ROI should not be saved in the first selected object. The design choice is to save the defined ROI neither in the first nor in any of the selected objects: the ROI is only used for the extraction, and is not saved in any object
  - Fixed [Issue #121](#) - `AttributeError` when extracting multiple ROIs on a single image, if more than one image is selected
  - Fixed [Issue #122](#) - Image masks are not refreshed when removing metadata except for the active image
  - Fixed [Issue #123](#) - Image masks are not refreshed when pasting metadata on multiple images, except for the last image
- Text and CSV files:

- Enhance text file reading by detecting data headers (using a list of typical headers from scientific instruments) and by allowing to skip the header when reading the file
- Ignore encoding errors when reading files in both open feature and import wizard, hence allowing to read files with special characters without raising an exception
- Fixed [Issue #124](#) - Text files: support locale decimal separator (different than .)
- Signal analysis features: fixed duplicate results when no ROI is defined
- Fixed [Issue #113](#) - Call to `RemoteClient.open_h5_files` (and `import_h5_file`) fails without passing the optional arguments
- Fixed [Issue #116](#) - `KeyError` exception when trying to remove a group after opening an HDF5 file

## DataLab Version 0.18.1

### Enhancements:

- FWHM computation now raises an exception when less than two points are found with zero-crossing method
- Improved result validation for array-like results by checking the data type of the result

### Bug fixes:

- Fixed [Issue #106](#) - Analysis: coordinate shifted results on images with ROIs and shifted origin
- Fixed [Issue #107](#) - Wrong indices when extracting a profile from an image with a ROI
- Fixed [Issue #111](#) - Proxy `add_object` method does not support signal/image metadata (e.g. ROI)
- Test data plugin / “Create 2D noisy gauss image”: fixed amplitude calculation in `cdl.tests.data.create_2d_random` for non-integer data types

### Documentation:

- Fixed path separators in plugin directory documentation
- Corrected left and right area descriptions in workspace documentation
- Updated Google style link in contributing guidelines
- Fixed various French translations in the documentation

## DataLab Version 0.18.0

### General information:

- PlotPy v2.7 is required for this release.
- Dropped support for Python 3.8.
- Python 3.13 is not supported yet, due to the fact that some dependencies are not compatible with this version.

### New features and enhancements:

- New operation mode feature:
  - Added “Operation mode” feature to the “Processing” tab in the “Settings” dialog box
  - This feature allows to choose between “single” and “pairwise” operation modes for all basic operations (addition, subtraction, multiplication, division, etc.):
    - \* “Single” mode: single operand mode (default mode: the operation is done on each object independently)

- \* “Pairwise” mode: pairwise operand mode (the operation is done on each pair of objects)
- This applies to both signals and images, and to computations taking  $N$  inputs
- Computations taking  $N$  inputs are the ones where:
  - \*  $N(\geq 2)$  objects in give  $N$  objects out
  - \*  $N(\geq 1)$  object(s) + 1 object in give  $N$  objects out
- New ROI (Region Of Interest) features:
  - New polygonal ROI feature
  - Complete redesign of the ROI editor user interfaces, improving ergonomics and consistency with the rest of the application
  - Major internal refactoring of the ROI system to make it more robust (more tests) and easier to maintain
- Implemented [Issue #102](#) - Launch DataLab using `datalab` instead of `cdl`. Note that the `cdl` command is still available for backward compatibility.
- Implemented [Issue #101](#) - Configuration: set default image interpolation to anti-aliasing (5 instead of 0 for nearest). This change is motivated by the fact that a performance improvement was made in PlotPy v2.7 on Windows, which allows to use anti-aliasing interpolation by default without a significant performance impact.
- Implemented [Issue #100](#) - Use the same installer and executable on Windows 7 SP1, 8, 10, 11. Before this change, a specific installer was required for Windows 7 SP1, due to the fact that Python 3.9 and later versions are not supported on this platform. A workaround was implemented to make DataLab work on Windows 7 SP1 with Python 3.9.

Bug fixes:

- Fixed [Issue #103](#) - `proxy.add_annotations_from_items`: circle shape color seems to be ignored.

## DataLab Version 0.17.1

PlotPy v2.6.2 is required for this release.

New features and enhancements:

- Image View:
  - Before this release, when selecting a high number of images (e.g. when selecting a group of images), the application was very slow because all the images were displayed in the image view, even if they were all superimposed on the same image
  - The workaround was to enable the “Show first only” option
  - Now, to improve performance, if multiple images are selected, only the last image of the selection is displayed in the image view if this last image has no transparency and if the other images are completely covered by this last image
- Clarification: action “Show first only” was renamed to “Show first object only”, and a new icon was added to the action
- API: added `width` and `height` properties to `ImageObj` class (returns the width and height of the image in physical units)
- Windows launcher “start.pyw”: writing a log file “`datalab_error.log`” when an exception occurs at startup

Bug fixes:

- Changing the color theme now correctly updates all DataLab’s user interface components without the need to restart the application



Other changes:

- OpenCV is now an optional dependency:
  - This change is motivated by the fact that the OpenCV conda package is not maintained on Windows (at least), which leads to an error when installing DataLab with conda
  - When OpenCV is not installed, only the “OpenCV blob detection” feature won’t work, and a warning message will be displayed when trying to use this feature

## DataLab Version 0.17.0

PlotPy v2.6 is required for this release.

New features and enhancements:

- Menu “Computing” was renamed to “Analysis” for both Signal and Image panels, to better reflect the nature of the features in this menu
- Regions Of Interest (ROIs) are now taken into account everywhere in the application where it makes sense, and not only for the old “Computing” menu (now “Analysis”) features. This closes [Issue #93](#). If a signal or an image has an ROI defined:
  - Operations are done on the ROI only (except if the operation changes the data shape, or the pixel size for images)
  - Processing features are done on the ROI only (if the destination object data type is compatible with the source object data type, which excludes thresholding, for instance)
  - Analysis features are done on the ROI only, like before
- As a consequence of previous point, and for clarity:
  - The “Edit Regions of interest” and “Remove all Regions of interest” features have been moved from the old “Computing” (now “Analysis”) menu to the “Edit” menu where all metadata-related features are located
  - The “Edit Regions of interest” action has been added to both Signal and Image View vertical toolbars (in second position, after the “View in a new window” action)
- Following the bug fix on image data type conversion issues with basic operations, a new “Arithmetic operation” feature has been added to the “Operations” menu for both Signal and Image panels. This feature allows to perform linear operations on signals and images, with the following operations:
  - Addition:  $\text{obj3} = (\text{obj1} + \text{obj2}) * a + b$
  - Subtraction:  $\text{obj3} = (\text{obj1} - \text{obj2}) * a + b$
  - Multiplication:  $\text{obj3} = (\text{obj1} * \text{obj2}) * a + b$
  - Division:  $\text{obj3} = (\text{obj1} / \text{obj2}) * a + b$
- Improved “View in a new window” and “ROI editor” dialog boxes size management: default size won’t be larger than DataLab’s main window size
- ROI editor:
  - Added toolbars for both Signal and Image ROI editors, to allow to zoom in and out, and to reset the zoom level easily
  - Rearranged the buttons in the ROI editor dialog box for better ergonomics and consistency with the Annotations editor (“View in a new window” dialog box)
- Application color theme:
  - Added support for color theme (auto, light, dark) in the “Settings” dialog box

- The color theme is applied without restarting the application

Bug fixes:

- Intensity profile / Segment profile extraction:
  - When extracting a profile on an image with a ROI defined, the associated PlotPy feature show a warning message ('UserWarning: Warning: converting a masked element to nan.') but the profile is correctly extracted and displayed, with NaN values where the ROI is not defined.
  - NaN values are now removed from the profile before plotting it
- Simple processing features with a one-to-one mapping with a Python function (e.g. `numpy.absolute`, `numpy.log10`, etc.) and without parameters: fix result object title which was systematically ending with "]" (the character that usually precedes the list of parameters)
- Butterworth filter: fix cutoff frequency ratio default value and valid range
- Fix actions refresh issue in Image View vertical toolbar:
  - When starting DataLab with the Signal Panel active, switching to the Image View was showing "View in a new window" or "Edit Regions of interest" actions enabled in the vertical toolbar, even if no image was displayed in the Image View
  - The Image View vertical toolbar is now correctly updated at startup
- View in a new window: cross section tools (intensity profiles) stayed disabled unless the user selected an image through the item list - this is now fixed
- Image View: "Show contrast panel" toolbar button was not enabled at startup, and was only enabled when at least one image was displayed in the Image View - it is now always enabled, as expected
- Image data type conversion:
  - Previously, the data type conversion feature was common to signal and image processing features, i.e. a simple conversion of the data type using NumPy's `astype` method
  - This was not sufficient for image processing features, in particular for integer images, because even if the result was correct from a numerical point of view, underflow or overflow could be legitimately seen as a bug from a mathematical point of view
  - The image data type conversion feature now relies on the internal `clip_astype` function, which clips the data to the valid range of the target data type before converting it (in the case of integer images)
- Image ROI extraction issues:
  - Multiple regressions were introduced in version 0.16.0:
    - \* Single circular ROI extraction was not working as expected (a rectangular ROI was extracted, with unexpected coordinates)
    - \* Multiple circular ROI extraction lead to a rectangular ROI extraction
    - \* Multiple ROI extraction was no longer cropping the image to the overall bounding box of the ROIs
  - These issues are now fixed, and unit tests have been added to prevent regressions:
    - \* An independent test algorithm has been implemented to check the correctness of the ROI extraction in all cases mentioned above
    - \* Tests cover both single and multiple ROI extraction, with circular and rectangular ROIs
- Overflow and underflow issues in some operations on integer images:

- When processing integer images, some features were causing overflow or underflow issues, leading to unexpected results (correct results from a numerical point of view, but not from a mathematical point of view)
- This issue only concerned basic operations (addition, subtraction, multiplication, division, and constant operations) - all the other features were already working as expected
- This is now fixed as result output are now floating point images
- Unit tests have been added to prevent regressions for all these operations

### DataLab Version 0.16.4

This is a minor maintenance release.

Bug fixes:

- Requires PlotPy v2.4.1 or later to fix the following issues related to the contrast adjustment feature:
  - A regression was introduced in an earlier version of PlotPy: levels histogram was no longer removed from contrast adjustment panel when the associated image was removed from the plot
  - This is now fixed: when an image is removed, the histogram is removed as well and the contrast panel is refreshed (which was not the case even before the regression)
- Ignore `AssertionError` in `config_unit_test.py` when executing test suite on WSL

Documentation:

- Fix class reference in `Wrap11Func` documentation

### DataLab Version 0.16.3

Bug fixes:

- Fixed [Issue #84](#) - Build issues with V0.16.1: signal name conflict, ...
  - This issue was intended to be fixed in version 0.16.2, but the fix was not complete
  - Thanks to [@rolandmas](#) for reporting the issue and for the help in investigating the problem and testing the fix
- Fixed [Issue #85](#) - Test data paths may be added multiple times to `cdl.utils.tests.TST_PATH`
  - This issue is related to [Issue #84](#)
  - Adding the test data paths multiple times to `cdl.utils.tests.TST_PATH` was causing the test data to be loaded multiple times, which lead to some tests failing (a simple workaround was added to V0.16.2: this issue is now fixed)
  - Thanks again to [@rolandmas](#) for reporting the issue in the context of the Debian packaging
- Fixed [Issue #86](#) - Average of N integer images overflows data type
- Fixed [Issue #87](#) - Image average profile extraction: `AttributeError` when trying to edit profile parameters
- Fixed [Issue #88](#) - Image segment profile: point coordinates inversion

## DataLab Version 0.16.2

This release requires PlotPy v2.4.0 or later, which brings the following bug fixes and new features:

- New contrast adjustment features and bug fixes:
  - New layout: the vertical toolbar (which was constrained in a small area on the right side of the panel) is now a horizontal toolbar at the top of the panel, beside the title
  - New “Set range” button: allows the user to set manually the minimum and maximum values of the histogram range
  - Fixed histogram update issues when no image was currently selected (even if the an image was displayed and was selected before)
  - Histogram range was not updated when either the minimum or maximum value was set using the “Minimum value” or “Maximum value” buttons (which have been renamed to “Min.” and “Max.” in this release)
  - Histogram range was not updated when the “Set full range” button was clicked, or when the LUT range was modified using the “Scales / LUT range” form in “Properties” group box
- Image view context menu: new “Reverse X axis” feature

Minor new features and enhancements:

- Image file types:
  - Added native support for reading .SPE, .GEL, .NDPI and .REC image files
  - Added support for any `imageio`-supported file format through configuration file (entry `imageio_formats` may be customized to complement the default list of supported formats: see [documentation](#) for more details)

Bug fixes:

- Image Fourier analysis:
  - Fixed logarithmic scale for the magnitude spectrum (computing dB instead of natural logarithm)
  - Fixed PSD computation with logarithmic scale (computing dB instead of natural logarithm)
  - Updated the documentation to explicitly mention that the logarithmic scale is in dB
- Fixed [Issue #82](#) - Macros are not renamed in DataLab after exporting them to Python scripts
- `ResultProperties` object can now be added to `SignalObj` or `ImageObj` metadata even outside a Qt event loop (because the label item is no longer created right away)
- Progress bar is now automatically closed as expected when an error occurs during a long operation (e.g. when opening a file)
- Difference, division...: dialog box for the second operand selection was allowing to select a group (only a signal or an image should be selected)
- When doing an operation which involves an object (signal or image) with higher order number than the current object (e.g. when subtracting an image with an image from a group below the current image), the resulting object's title now correctly refers to the order numbers of the objects involved in the operation (e.g., to continue with the subtraction example mentioned above, the resulting object's title was previously referring to the order number before the insertion of the resulting image)
- Added support for additional test data folder thanks to the `CDL_DATA` environment variable (useful for testing purposes, and especially in the context of Debian packaging)

## DataLab Version 0.16.1

Since version 0.16.0, many validation functions have been added to the test suite. The percentage of validated compute functions has increased from 37% to 84% in this release.

NumPy 2.0 support has been added with this release.

Minor new features and enhancements:

- Signal and image moving average and median filters:
  - Added “Mode” parameter to choose the mode of the filter (e.g. “reflect”, “constant”, “nearest”, “mirror”, “wrap”)
  - The default mode is “reflect” for moving average and “nearest” for moving median
  - This allows to handle edge effects when filtering signals and images

Bug fixes:

- Fixed Canny edge detection to return binary image as `uint8` instead of `bool` (for consistency with other image processing features)
- Fixed Image normalization: lower bound was wrongly set for `maximum` method
- Fixed `ValueError` when computing PSD with logarithmic scale
- Fixed Signal derivative algorithm: now using `numpy.gradient` instead of a custom implementation
- Fixed SciPy’s `cumtrapz` deprecation: use `cumulative_trapezoid` instead
- Curve selection now shows the individual points of the curve (before, only the curve line width was broadened)
- Windows installer: add support for unstable releases (e.g., 0.16.1.dev0), thus allowing to easily install the latest development version of DataLab on Windows
- Fixed [Issue #81](#) - When opening files, show progress dialog only if necessary
- Fixed [Issue #80](#) - Plotting results: support for two use cases
  - The features of the “Analysis” menu produce *results* (scalars): blob detection (circle coordinates), 2D peak detection (point coordinates), etc. Depending on the feature, result tables are displayed in the “Results” dialog box, and the results are also stored in the signal or image metadata: each line of the result table is an individual result, and each column is a property of the result - some results may consist only of a single individual result (e.g., image centroid or curve FWHM), while others may consist of multiple individual results (e.g., blob detection, contour detection, etc.).
  - Before this change, the “Plot results” feature only supported plotting the first individual result of a result table, as a function of the index (of the signal or image objects) or any of the columns of the result table. This was not sufficient for some use cases, where the user wanted to plot multiple individual results of a result table.
  - Now, the “Plot results” feature supports two use cases:
    - \* “One curve per result title”: Plotting the first individual result of a result table, as before
    - \* “One curve per object (or ROI) and per result title”: Plotting all individual results of a result table, as a function of the index (of the signal or image objects) or any of the columns of the result table
  - The selection of the use case is done in the “Plot results” dialog box
  - The default use case is “One curve per result title” if the result table has only one line, and “One curve per object (or ROI) and per result title” otherwise

## DataLab Version 0.16.0

New features and enhancements:

- Major user interface overhaul:
  - The menu bar and toolbars have been reorganized to make the application more intuitive and easier to use
  - Operations and processing features have been regrouped in submenus
  - All visualization-related actions are now grouped in the plot view vertical toolbar
  - Clarified the “Annotations” management (new buttons, toolbar action...)
- New validation process for signal and image features:
  - Before this release, DataLab’s validation process was exclusively done from the programmer’s point of view, by writing unit tests and integration tests, thus ensuring that the code was working as expected (i.e. that no exception was raised and that the behavior was correct)
  - With this release, a new validation process has been introduced, from the user’s point of view, by adding new validation functions (marked with the `@pytest.mark.validation` decorator) in the test suite
  - A new “Validation” section in the documentation explains how validation is done and contains a list of all validation functions with the statistics of the validation process (generated from the test suite)
  - The validation process is a work in progress and will be improved in future versions
- “Properties” group box:
  - Added “Scales” tab, to show and set the plot scales:
    - \* X, Y for signals
    - \* X, Y, Z (LUT range) for images
- View options:
  - New “Show first only” option in the “View” menu, to show only the first curve (or image) when multiple curves (or images) are displayed in the plot view
  - New (movable) label for FWHM computations, additional to the existing segment annotation
- I/O features:
  - Added support for reading and writing .MAT files (MATLAB format)
  - Create a new group when opening a file containing multiple signals or images (e.g. CSV file with multiple curves)
- Add support for binary images
- Signal ROI extraction: added new dialog box to manually edit the ROI lower and upper bounds after defining the ROI graphically

New **Signal** operations, processing and analysis features:

Menu	Submenu	Features
New	New signal	Exponential, pulse, polynomial, experimental (manual input)
Operations		Exponential, Square root, Power
Operations	Operations with a constant	+, -, *, /
Processing	Axis Transformation	Reverse X-axis
Processing	Level Adjustment	Offset correction
Processing	Fourier analysis	Power spectrum, Phase spectrum, Magnitude spectrum, Power spectral density
Processing	Frequency filters	Low-pass, High-pass, Band-pass, Band-stop
Processing		Windowing (Hanning, Hamming, Blackman, Blackman-Harris, Nuttall, Flat-top...)
Processing	Fit	Linear fit, Sinusoidal fit, Exponential fit, CDF fit
Analysis		FWHM (Zero-crossing method), X value @ min/max, Sampling period/frequency, Dynamic parameters (ENOB, SNR, SINAD, THD, SFDR), -3dB bandwidth, Contrast

New **Image** operations, processing and analysis features:

Menu	Submenu	Features
Operations		Exponential
Operations	Intensity profiles	Profile along a segment
Operations	Operations with a constant	+, -, *, /
Processing	Level Adjustment	Normalization, Clipping, Offset correction
Processing	Fourier analysis	Power spectrum, Phase spectrum, Magnitude spectrum, Power spectral density
Processing	Thresholding	Parametric, ISODATA, Li, Mean, Minimum, Otsu, Triangle, Yen

Bug fixes:

- Fixed a performance issue due to an unnecessary refresh of the plot view when adding a new signal or image
- Fixed [Issue #77](#) - Intensity profiles: unable to accept dialog the second time
- Fixed [Issue #75](#) - View in a new window: curve anti-aliasing is not enabled by default

- Annotations visibility is now correctly saved and restored:
  - Before this release, when modifying the annotations visibility in the separate plot view, the visibility was not saved and restored when reopening the plot view
  - This has been [fixed upstream](#) in PlotPy (v2.3.3)

## **DataLab Version 0.15.1**

Bug fixes:

- Fixed [Issue #68](#) - Slow loading of even simple plots:
  - On macOS, the user experience was degraded when handling even simple plots
  - This was due to the way macOS handles the pop-up windows, e.g. when refreshing the plot view (“Creating plot items” progress bar), hence causing a very annoying flickering effect and a global slowdown of the application
  - This is now fixed by showing the progress bar only after a short delay (1s), that is when it is really needed (i.e. for long operations)
  - Thanks to [@marcel-goldschen-ohm](#) for the very thorough feedback and the help in testing the fix
- Fixed [Issue #69](#) - Annotations should be read-only in Signal/Image View
  - Regarding the annotations, DataLab’s current behavior is the following:
    - \* Annotations are created only when showing the signal/image in a separate window (double-click on the object, or “View” > “View in a new window”)
    - \* When displaying the objects in either the “Signal View” or the “Image View”, the annotations should be read-only (i.e. not movable, nor resizable or deletable)
  - However, some annotations were still deletable in the “Signal View” and the “Image View”: this is now fixed
  - Note that the fact that annotations can’t be created in the “Signal View” or the “Image View” is a limitation of the current implementation, and may be improved in future versions

## **DataLab Version 0.15.0**

New installer for the stand-alone version on Windows:

- The stand-alone version on Windows is now distributed as an MSI installer (instead of an EXE installer)
- This avoids the false positive detection of the stand-alone version as a potential threat by some antivirus software
- The program will install files and shortcuts:
  - For current user, if the user has no administrator privileges
  - For all users, if the user has administrator privileges
  - Installation directory may be customized
- MSI installer allows to integrate DataLab’s installation seamlessly in an organization’s deployment system

New features and enhancements:

- Added support for large text/CSV files:
  - Files over 1 GB (and with reasonable number of lines) can now be imported as signals or images without crashing the application or even slowing it down



- The file is read by chunks and, for signals, the data is downsampled to a reasonable number of points for visualization
- Large files are supported when opening a file (or dragging and dropping a file in the Signal Panel) and when importing a file in the Text Import Wizard
- Auto downsampling feature:
  - Added “Auto downsampling” feature to signal visualization settings (see “Settings” dialog box)
  - This feature allows to automatically downsample the signal data for visualization when the number of points is too high and would lead to a slow rendering
  - The downsampling factor is automatically computed based on the configured maximum number of points to display
  - This feature is enabled by default and may be disabled in the signal visualization settings
- CSV format handling:
  - Improved support for CSV files with a header row (column names)
  - Added support for CSV files with empty columns
- Open/save file error handling:
  - Error messages are now more explicit when opening or saving a file fails
  - Added a link to the folder containing the file in the error message
- Added “Plugins and I/O formats” page to the Installation and Configuration Viewer (see “Help” menu)
- Reset DataLab configuration:
  - In some cases, it may be useful to reset the DataLab configuration file to its default values (e.g. when the configuration file is corrupted)
  - Added new `--reset` command line option to remove the configuration folder
  - Added new “Reset DataLab” Start Menu shortcut to the Windows installer

#### Bug fixes:

- Fixed [Issue #64](#) - HDF5 browser does not show datasets with 1x1 size:
  - HDF5 datasets with a size of 1x1 were not shown in the HDF5 browser
  - Even if those datasets should not be considered as signals or images, they are now shown in the HDF5 browser (but not checkable, i.e. not importable as signals or images)

### DataLab Version 0.14.2

API changes required for fixing support for multiple signals loading feature:

- Merged `open_object` and `open_objects` methods to `load_from_files` in proxy classes, main window and data panels
- For consistency’s sake: merged `save_object` and `save_objects` into `save_to_files`
- To sum up, those changes lead to the following situation:
  - `load_from_files`: load a sequence of objects from multiple files
  - `save_to_files`: save a sequence of objects to multiple files (at the moment, it only supports saving a single object to a single file, but it may be extended in the future to support saving multiple objects to a single file)

Bug fixes:

- Fixed [Issue #61](#) - Text file import wizard: application crash when importing a multiple curve text file:
  - This issue concerns a use case where the text file contains multiple curves
  - This is now fixed and an automatic test has been added to prevent regressions

## DataLab Version 0.14.1

New domain name: [datalab-platform.com](https://datalab-platform.com)

New features:

- Added support for colormap inversion in Image View:
  - New “Invert colormap” entry in plot context menu, image parameters, and in the default image view settings
  - This requires PlotPy v2.3 or later
- HDF5 Browser:
  - Added “Show array” button at the corner of the “Group” and “Attributes” tabs, to show the array in a separate window (useful for copy/pasting data to other applications, for instance)
  - Attributes: added support for more scalar data types
- Testability and maintainability:
  - DataLab’s unit tests are now using [pytest](#). This has required a lot of work for the transition, especially to readapt the tests so that they may be executed in the same process. For instance, a particular attention has been given to sandboxing the tests, so that they do not interfere with each other.
  - Added continuous integration (CI) with GitHub Actions
  - For this release, test coverage is 87%
- Text file import assistant:
  - Drastically improved the performance of the array preview when importing large text files (no more progress bar, and the preview is now displayed almost instantaneously)

Bug fixes:

- XML-RPC server was not shut down properly when closing DataLab
- Fixed test-related issues: some edge cases were hidden by the old test suite, and have been revealed by the transition to [pytest](#). This has led to some bug fixes and improvements in the code.
- On Linux, when running a computation on a signal or an image, and on rare occasions, the computation was stuck as if it was running indefinitely. Even though the graphical user interface was still responsive, the computation was not progressing and the user had to cancel the operation and restart it. This was due to the start method of the separate process used for the computation (default method was “fork” on Linux). This is now fixed by using the “spawn” method instead, which is the recommended method for latest versions of Python on Linux when multithreading is involved.
- Fixed [Issue #60](#) - `OSError: Invalid HDF5 file [...] when trying to open an HDF5 file with an extension other than “.h5”`
- Image Region of Interest (ROI) extraction: when modifying the image bounds in the confirmation dialog box, the ROI was not updated accordingly until the operation was run again
- Deprecation issues:
  - Fixed `scipy.ndimage.filters` deprecation warning

- Fixed `numpy.fromstring` deprecation warning

## DataLab Version 0.14.0

New features:

- New “Histogram” feature in “Analysis” menu:
  - Added histogram computation feature for both signals and images
  - The histogram is computed on the regions of interest (ROI) if any, or on the whole signal/image if no ROI is defined
  - Editable parameters: number of bins, lower and upper bounds
- HDF5 browser:
  - Improved tree view layout (more compact and readable)
  - Multiple files can now be opened at once, using the file selection dialog box
  - Added tabs with information below the graphical preview:
    - \* Group info: path, textual preview, etc.
    - \* Attributes info: name, value
  - Added “Show only supported data” check box: when checked, only supported data (signals and images) are shown in the tree view
  - Added “Show values” check box, to show/hide the values in the tree view
- Macro Panel:
  - Macro commands are now numbered, starting from 1, like signals and images
- Remote control API (RemoteProxy and LocalProxy):
  - `get_object_titles` method now accepts “macro” as panel name and returns the list of macro titles
  - New `run_macro`, `stop_macro` and `import_macro_from_file` methods

Bug fixes:

- Stand-alone version - Integration in Windows start menu:
  - Fixed “Uninstall” shortcut (unclickable due to a generic name)
  - Translated “Browse installation directory” and “Uninstall” shortcuts
- Fixed [Issue #55](#) - Changing image bounds in Image View has no effect on the associated image object properties
- Fixed [Issue #56](#) - “Test data” plugin: `AttributeError: 'NoneType' object has no attribute 'data'` when canceling “Create image with peaks”
- Fixed [Issue #57](#) - Circle and ellipse result shapes are not transformed properly
- Curve color and style cycle:
  - Before this release, this cycle was handled by the same mechanism either for the Signal Panel or the HDF5 Browser, which was not the expected behavior
  - Now, the cycle is handled separately: the HDF5 Browser or the Text Import Wizard use always the same color and style for curves, and they don’t interfere with the Signal Panel cycle

## DataLab Version 0.12.0

### Clarity-Enhanced Interface Update:

- The tabs used to switch between the data panels (signals and images) and the visualization components (“Curve panel” and “Image panel”) have been renamed to “Signal Panel” and “Image Panel” (instead of “Signals” and “Images”)
- The visualization components have been renamed to “Signal View” and “Image View” (instead of “Curve panel” and “Image panel”)
- The data panel toolbar has been renamed to “Signal Toolbar” and “Image Toolbar” (instead of “Signal Processing Toolbar” and “Image Processing Toolbar”)
- Ergonomics improvements: the “Signal Panel” and “Image Panel” are now displayed on the left side of the main window, and the “Signal View” and “Image View” are displayed on the right side of the main window. This reduces the distance between the list of objects (signals and images) and the associated actions (toolbars and menus), and makes the interface more intuitive and easier to use

### New tour and demo feature:

- When starting DataLab for the first time, an optional tour is now shown to the user to introduce the main features of the application
- The tour can be started again at any time from the “?” menu
- Also added a new “Demo” feature to the “?” menu

### New Binder environment to test DataLab online without installing anything

#### Documentation:

- New text tutorials are available:
  - Measuring Laser Beam Size
  - DataLab and Spyder: a perfect match
- “Getting started” section: added more explanations and links to the tutorials
- New “Contributing” section explaining how to contribute to DataLab, whether you are a developer or not
- New “Macros” section explaining how to use the macro commands feature
- Added “Copy” button to code blocks in the documentation

#### New features:

- New “Text file import assistant” feature:
  - This feature allows to import text files as signals or images
  - The user can define the source (clipboard or text file)
  - Then, it is possible to define the delimiter, the number of rows to skip, the destination data type, etc.
- Added menu on the “Signal Panel” and “Image Panel” tabs corner to quickly access the most used features (e.g. “Add”, “Remove”, “Duplicate”, etc.)
- Intensity profile extraction feature:
  - Added graphical user interface to extract intensity profiles from images, for both line and averaged profiles
  - Parameters are still directly editable by the user (“Edit profile parameters” button)
  - Parameters are now stored from one profile extraction to another
- Statistics feature:

- Added  $\langle y \rangle$  /  $\langle y \rangle$  to the signal “Statistics” result table (in addition to the mean, median, standard deviation, etc.)
- Added peak-to-peak to the signal and image “Statistics” result table
- Curve fitting feature: fit results are now stored in a dictionary in the signal metadata (instead of being stored individually in the signal metadata)
- Window state:
  - The toolbars and dock widgets state (visibility, position, etc.) are now stored in the configuration file and restored at startup (size and position were already stored and restored)
  - This implements part of [Issue #30](#) - Save/restore main window layout

#### Bug fixes:

- Fixed [Issue #41](#) - Radial profile extraction: unable to enter user-defined center coordinates
- Fixed [Issue #49](#) - Error when trying to open a (UTF-8 BOM) text file as an image
- Fixed [Issue #51](#) - Unexpected dimensions when adding new ROI on an image with X/Y arbitrary units (not pixels)
- Improved plot item style serialization management:
  - Before this release, the plot item style was stored in the signal/image metadata only when saving the workspace to an HDF5 file. So, when modifying the style of a signal/image from the “Parameters” button (view toolbar), the style was not kept in some cases (e.g. when duplicating the signal/image).
  - Now, the plot item style is stored in the signal/image metadata whenever the style is modified, and is restored when reloading the workspace
- Handled ComplexWarning cast warning when adding regions of interest (ROI) to a signal with complex data

### DataLab Version 0.11.0

#### New features:

- Signals and images may now be reordered in the tree view:
  - Using the new “Move up” and “Move down” actions in the “Edit” menu (or using the corresponding toolbar buttons):
  - This fixes [Issue #22](#) - Add “move up/down” actions in “Edit” menu, for signals/images and groups
- Signals and images may also be reordered using drag and drop:
  - Signals and images can be dragged and dropped inside their own panel to change their order
  - Groups can also be dragged and dropped inside their panel
  - The feature also supports multi-selection (using the standard Ctrl and Shift modifiers), so that multiple signals/images/groups can be moved at once, not necessarily with contiguous positions
  - This fixes [Issue #17](#) - Add Drag and Drop feature to Signals/Images tree views
- New 1D interpolation features:
  - Added “Interpolation” feature to signal panel’s “Processing” menu
  - Methods available: linear, spline, quadratic, cubic, barycentric and PCHIP
  - Thanks to [@marcel-goldschen-ohm](#) for the contribution to spline interpolation
  - This fixes [Issue #20](#) - Add 1D interpolation features
- New 1D resampling feature:

- Added “Resampling” feature to signal panel’s “Processing” menu
- Same interpolation methods as for the “Interpolation” feature
- Possibility to specify the resampling step or the number of points
- This fixes [Issue #21](#) - Add 1D resampling feature
- New 1D convolution feature:
  - Added “Convolution” feature to signal panel’s “Operation” menu
  - This fixes [Issue #23](#) - Add 1D convolution feature
- New 1D detrending feature:
  - Added “Detrending” feature to signal panel’s “Processing” menu
  - Methods available: linear or constant
  - This fixes [Issue #24](#) - Add 1D detrending feature
- 2D analysis results:
  - Before this release, 2D analysis results such as contours, blobs, etc. were stored in image metadata dictionary as coordinates (x0, y0, x1, y1, ...) even for circles and ellipses (i.e. the coordinates of the bounding rectangles).
  - For convenience, the circle and ellipse coordinates are now stored in image metadata dictionary as (x0, y0, radius) and (x0, y0, a, b, theta) respectively.
  - These results are also shown as such in the “Results” dialog box (either at the end of the computing process or when clicking on the “Show results” button).
  - This fixes [Issue #32](#) - Contour detection: show circle (x, y, r) and ellipse (x, y, a, b, theta) instead of (x0, y0, x1, y1, ...)
- 1D and 2D analysis results:
  - Additionally to the previous enhancement, more analysis results are now shown in the “Results” dialog box
  - This concerns both 1D (FWHM...) and 2D analysis results (contours, blobs...):
    - \* Segment results now also show length (L) and center coordinates (Xc, Yc)
    - \* Circle and ellipse results now also show area (A)
- Added “Plot results” entry in “Analysis” menu:
  - This feature allows to plot analysis results (1D or 2D)
  - It creates a new signal with X and Y axes corresponding to user-defined parameters (e.g. X = indices and Y = radius for circle results)
- Increased default width of the object selection dialog box:
  - The object selection dialog box is now wider by default, so that the full signal/image/group titles may be more easily readable
- Delete metadata feature:
  - Before this release, the feature was deleting all metadata, including the Regions Of Interest (ROI) metadata, if any.
  - Now a confirmation dialog box is shown to the user before deleting all metadata if the signal/image has ROI metadata: this allows to keep the ROI metadata if needed.

- Image profile extraction feature: added support for masked images (when defining regions of interest, the areas outside the ROIs are masked, and the profile is extracted only on the unmasked areas, or averaged on the unmasked areas in the case of average profile extraction)
- Curve style: added “Reset curve styles” in “View” menu. This feature allows to reset the curve style cycle to its initial state.
- Plugin base classe `PluginBase`:
  - Added `edit_new_signal_parameters` method for showing a dialog box to edit parameters for a new signal
  - Added `edit_new_image_parameters` method for showing a dialog box to edit parameters for a new image (updated the `cdl_testdata.py` plugin accordingly)
- Signal and image computations API (`cdl.computations`):
  - Added wrappers for signal and image 1 -> 1 computations
  - These wrappers aim at simplifying the creation of a basic computation function operating on DataLab’s native objects (`SignalObj` and `ImageObj`) from a function operating on NumPy arrays
  - This simplifies DataLab’s internals and makes it easier to create new computing features inside plugins
  - See the `cdl_custom_func.py` example plugin for a practical use case
- Added “Radial profile extraction” feature to image panel’s “Operation” menu:
  - This feature allows to extract a radially averaged profile from an image
  - The profile is extracted around a user-defined center (x0, y0)
  - The center may also be computed (centroid or image center)
- Automated test suite:
  - Since version 0.10, DataLab’s proxy object has a `toggle_auto_refresh` method to toggle the “Auto-refresh” feature. This feature may be useful to improve performance during the execution of test scripts
  - Test scenarios on signals and images are now using this feature to improve performance
- Signal and image metadata:
  - Added “source” entry to the metadata dictionary, to store the source file path when importing a signal or an image from a file
  - This field is kept while processing the signal/image, in order to keep track of the source file path

#### Documentation:

- New [Tutorial section](#) in the documentation:
  - This section provides a set of tutorials to learn how to use DataLab
  - The following video tutorials are available:
    - \* Quick demo
    - \* Adding your own features
  - The following text tutorials are available:
    - \* Processing a spectrum
    - \* Detecting blobs on an image
    - \* Measuring Fabry-Perot fringes
    - \* Prototyping a custom processing pipeline

- New [API section](#) in the documentation:
  - This section explains how to use DataLab as a Python library, by covering the following topics:
    - \* How to use DataLab algorithms on NumPy arrays
    - \* How to use DataLab computation features on DataLab objects (signals and images)
    - \* How to use DataLab I/O features
    - \* How to use proxy objects to control DataLab remotely
  - This section also provides a complete API reference for DataLab objects and features
  - This fixes [Issue #19](#) - Add API documentation (data model, functions on arrays or signal/image objects, ...)

Bug fixes:

- Fixed [Issue #29](#) - Polynomial fit error: `QDialog [...]` argument 1 has an unexpected type `'SignalProcessor'`
- Image ROI extraction feature:
  - Before this release, when extracting a single circular ROI from an image with the “Extract all ROIs into a single image object” option enabled, the result was a single image without the ROI mask (the ROI mask was only available when extracting ROI with the option disabled)
  - This was leading to an unexpected behavior, because one could interpret the result (a square image without the ROI mask) as the result of a single rectangular ROI
  - Now, when extracting a single circular ROI from an image with the “Extract all ROIs into a single image object” option enabled, the result is a single image with the ROI mask (as if the option was disabled)
  - This fixes [Issue #31](#) - Single circular ROI extraction: automatically switch to `extract_single_roi` function
- Analysis on circular ROI:
  - Before this release, when running computations on a circular ROI, the results were unexpected in terms of coordinates (results seemed to be computed in a region located above the actual ROI).
  - This was due to a regression introduced in an earlier release.
  - Now, when defining a circular ROI and running computations on it, the results are computed on the actual ROI
  - This fixes [Issue #33](#) - Analysis on circular ROI: unexpected results
- Contour detection on ROI:
  - Before this release, when running contour detection on a ROI, some contours were detected outside the ROI (it may be due to a limitation of the scikit-image `find_contours` function).
  - Now, thanks a workaround, the erroneous contours are filtered out.
  - A new test module `cdl.tests.features.images.contour_fabryperot_app` has been added to test the contour detection feature on a Fabry-Perot image (thanks to [@emarin2642](#) for the contribution)
  - This fixes [Issue #34](#) - Contour detection: unexpected results outside ROI
- Analysis result merging:
  - Before this release, when doing a 1->N computation (sum, average, product) on a group of signals/images, the analysis results associated to each signal/image were merged into a single result, but only the type of result present in the first signal/image was kept.



- Now, the analysis results associated to each signal/image are merged into a single result, whatever the type of result is.
- Fixed [Issue #36](#) - “Delete all” action enable state is sometimes not refreshed
- Image X/Y swap: when swapping X and Y axes, the regions of interest (ROI) were not removed and not swapped either (ROI are now removed, until we implement the swap feature, if requested)
- “Properties” group box: the “Apply” button was enabled by default, even when no property was modified, which was confusing for the user (the “Apply” button is now disabled by default, and is enabled only when a property is modified)
- Fixed proxy `get_object` method when there is no object to return (None is returned instead of an exception)
- Fixed `IndexError: list index out of range` when performing some operations or computations on groups of signals/images (e.g. “ROI extraction”, “Peak detection”, “Resize”, etc.)
- Drag and drop from a file manager: filenames are now sorted alphabetically

### DataLab Version 0.10.1

*Note:* V0.10.0 was almost immediately replaced by V0.10.1 due to a last minute bug fix

New features:

- Features common to signals and images:
  - Added “Real part” and “Imaginary part” features to “Operation” menu
  - Added “Convert data type” feature to “Operation” menu
- Features added following user requests (12/18/2023 meetup @ CEA):
  - Curve and image styles are now saved in the HDF5 file:
    - \* Curve style covers the following properties: color, line style, line width, marker style, marker size, marker edge color, marker face color, etc.
    - \* Image style covers the following properties: colormap, interpolation, etc.
    - \* Those properties were already persistent during the working session, but were lost when saving and reloading the HDF5 file
    - \* Now, those properties are saved in the HDF5 file and are restored when reloading the HDF5 file
  - New profile extraction features for images:
    - \* Added “Line profile” to “Operations” menu, to extract a profile from an image along a row or a column
    - \* Added “Average profile” to “Operations” menu, to extract the average profile on a rectangular area of an image, along a row or a column
  - Image LUT range (contrast/brightness settings) is now saved in the HDF5 file:
    - \* As for curve and image styles, the LUT range was already persistent during the working session, but was lost when saving and reloading the HDF5 file
    - \* Now, the LUT range is saved in the HDF5 file and is restored when reloading it
  - Added “Auto-refresh” and “Refresh manually” actions in “View” menu (and main toolbar):
    - \* When “Auto-refresh” is enabled (default), the plot view is automatically refreshed when a signal/image is modified, added or removed. Even though the refresh is optimized, this may lead to performance issues when working with large datasets.

- \* When disabled, the plot view is not automatically refreshed. The user must manually refresh the plot view by clicking on the “Refresh manually” button in the main toolbar or by pressing the standard refresh key (e.g. “F5”).
  - Added `toggle_auto_refresh` method to DataLab proxy object:
    - \* This method allows to toggle the “Auto-refresh” feature from a macro-command, a plugin or a remote control client.
    - \* A context manager `context_no_refresh` is also available to temporarily disable the “Auto-refresh” feature from a macro-command, a plugin or a remote control client. Typical usage:
- ```
with proxy.context_no_refresh():  
    # Do something without refreshing the plot view  
    proxy.compute_fft() # (...)    ``
```
- Improved curve readability:
  - \* Until this release, the curve style was automatically set by cycling through **PlotPy** predefined styles
  - \* However, some styles are not suitable for curve readability (e.g. “cyan” and “yellow” colors are not readable on a white background, especially when combined with a “dashed” line style)
  - \* This release introduces a new curve style management with colors which are distinguishable and accessible, even to color vision deficiency people
- Added “Curve anti-aliasing” feature to “View” menu (and toolbar):
  - This feature allows to enable/disable curve anti-aliasing (default: enabled)
  - When enabled, the curve rendering is smoother but may lead to performance issues when working with large datasets (that’s why it can be disabled)
- Added `toggle_show_titles` method to DataLab proxy object. This method allows to toggle the “Show graphical object titles” feature from a macro-command, a plugin or a remote control client.
- Remote client is now checking the server version and shows a warning message if the server version may not be fully compatible with the client version.

#### Bug fixes:

- Image contour detection feature (“Analysis” menu):
  - The contour detection feature was not taking into account the “shape” parameter (circle, ellipse, polygon) when computing the contours. The parameter was stored but really used only when calling the feature a second time.
  - This unintentional behavior led to an `AssertionError` when choosing “polygon” as the contour shape and trying to compute the contours for the first time.
  - This is now fixed (see [Issue #9](#) - Image contour detection: `AssertionError` when choosing “polygon” as the contour shape)
- Keyboard shortcuts:
  - The keyboard shortcuts for “New”, “Open”, “Save”, “Duplicate”, “Remove”, “Delete all” and “Refresh manually” actions were not working properly.
  - Those shortcuts were specific to each signal/image panel, and were working only when the panel on which the shortcut was pressed for the first time was active (when activated from another panel, the shortcut was not working and a warning message was displayed in the console, e.g. `QAction::event: Ambiguous shortcut overload: Ctrl+C`)

- Besides, the shortcuts were not working at startup (when no panel had focus).
- This is now fixed: the shortcuts are now working whatever the active panel is, and even at startup (see [Issue #10](#) - Keyboard shortcuts not working properly: QAction::event: Ambiguous shortcut overload: Ctrl+C)
- “Show graphical object titles” and “Auto-refresh” actions were not working properly:
  - The “Show graphical object titles” and “Auto-refresh” actions were only working on the active signal/image panel, and not on all panels.
  - This is now fixed (see [Issue #11](#) - “Show graphical object titles” and “Auto-refresh” actions were working only on current signal/image panel)
- Fixed [Issue #14](#) - Saving/Reopening HDF5 project without cleaning-up leads to ValueError
- Fixed [Issue #15](#) - MacOS: 1. pip install cdl error - 2. Missing menus:
  - Part 1: pip install cdl error on MacOS was actually an issue from **PlotPy** (see [this issue](#)), and has been fixed in PlotPy v2.0.3 with an additional compilation flag indicating to use C++11 standard
  - Part 2: Missing menus on MacOS was due to a PyQt/MacOS bug regarding dynamic menus
- HDF5 file format: when importing an HDF5 dataset as a signal or an image, the dataset attributes were systematically copied to signal/image metadata: we now only copy the attributes which match standard data types (integers, floats, strings) to avoid errors when serializing/deserializing the signal/image object
- Installation/configuration viewer: improved readability (removed syntax highlighting)
- PyInstaller specification file: added missing skimage data files manually in order to continue supporting Python 3.8 (see [Issue #12](#) - Stand-alone version on Windows 7: missing api-ms-win-core-path-l1-1-0.dll)
- Fixed [Issue #13](#) - ArchLinux: qt.qpa.plugin: Could not load the Qt platform plugin "xcb" in "" even though it was found

## DataLab Version 0.9.2

### Bug fixes:

- Region of interest (ROI) extraction feature for images:
  - ROI extraction was not working properly when the “Extract all ROIs into a single image object” option was enabled if there was only one defined ROI. The result was an image positioned at the origin (0, 0) instead of the expected position (x0, y0) and the ROI rectangle itself was not removed as expected. This is now fixed (see [Issue #6](#) - ‘Extract multiple ROI’ feature: unexpected result for a single ROI)
  - ROI rectangles with negative coordinates were not properly handled: ROI extraction was raising a ValueError exception, and the image mask was not displayed properly. This is now fixed (see [Issue #7](#) - Image ROI extraction: ValueError: zero-size array to reduction operation minimum which has no identity)
  - ROI extraction was not taking into account the pixel size (dx, dy) and the origin (x0, y0) of the image. This is now fixed (see [Issue #8](#) - Image ROI extraction: take into account pixel size)
- Macro-command console is now read-only:
  - The macro-command panel Python console is currently not supporting standard input stream (stdin) and this is intended (at least for now)
  - Set Python console read-only to avoid confusion

## DataLab Version 0.9.1

### Bug fixes:

- French translation is not available on Windows/Stand alone version:
  - Locale was not properly detected on Windows for stand-alone version (frozen with `pyinstaller`) due to an issue with `locale.getlocale()` (function returning `None` instead of the expected locale on frozen applications)
  - This is ultimately a `pyinstaller` issue, but a workaround has been implemented in `guidata` V3.2.2 (see [guidata issue #68](#) - Windows: gettext translation is not working on frozen applications)
  - [Issue #2](#) - French translation is not available on Windows Stand alone version
- Saving image to JPEG2000 fails for non integer data:
  - JPEG2000 encoder does not support non integer data or signed integer data
  - Before, DataLab was showing an error message when trying to save incompatible data to JPEG2000: this was not a consistent behavior with other standard image formats (e.g. PNG, JPG, etc.) for which DataLab was automatically converting data to the appropriate format (8-bit unsigned integer)
  - Current behavior is now consistent with other standard image formats: when saving to JPEG2000, DataLab automatically converts data to 8-bit unsigned integer or 16-bit unsigned integer (depending on the original data type)
  - [Issue #3](#) - Save image to JPEG2000: 'OSError: encoder error -2 when writing image file'
- Windows stand-alone version shortcuts not showing in current user start menu:
  - When installing DataLab on Windows from a non-administrator account, the shortcuts were not showing in the current user start menu but in the administrator start menu instead (due to the elevated privileges of the installer and the fact that the installer does not support installing shortcuts for all users)
  - Now, the installer *does not* ask for elevated privileges anymore, and shortcuts are installed in the current user start menu (this also means that the current user must have write access to the installation directory)
  - In future releases, the installer will support installing shortcuts for all users if there is a demand for it (see [Issue #5](#))
  - [Issue #4](#) - Windows: stand-alone version shortcuts not showing in current user start menu
- Installation and configuration window for stand-alone version:
  - Do not show ambiguous error message 'Invalid dependencies' anymore
  - Dependencies are supposed to be checked when building the stand-alone version
- Added PDF documentation to stand-alone version:
  - The PDF documentation was missing in previous release
  - Now, the PDF documentation (in English and French) is included in the stand-alone version

## DataLab Version 0.9.0

New dependencies:

- DataLab is now powered by [PlotPyStack](#):
  - [PythonQwt](#)
  - [guidata](#)
  - [PlotPy](#)
- [opencv-python](#) (algorithms for image processing)

New reference platform:

- DataLab is validated on Windows 11 with Python 3.11 and PyQt 5.15
- DataLab is also compatible with other OS (Linux, MacOS) and other Python-Qt bindings and versions (Python 3.8-3.12, PyQt6, PySide6)

New features:

- DataLab is a platform:
  - Added support for plugins
    - \* Custom processing features available in the “Plugins” menu
    - \* Custom I/O features: new file formats can be added to the standard I/O features for signals and images
    - \* Custom HDF5 features: new HDF5 file formats can be added to the standard HDF5 import feature
    - \* More features to come...
  - Added remote control feature: DataLab can be controlled remotely via a TCP/IP connection (see [Remote control](#))
  - Added macro commands: DataLab can be controlled via a macro file (see [Macro commands](#))
- General features:
  - Added settings dialog box (see “Settings” entry in “File” menu):
    - \* General settings
    - \* Visualization settings
    - \* Processing settings
    - \* Etc.
  - New default layout: signal/image panels are on the right side of the main window, visualization panels are on the left side with a vertical toolbar
- Signal/Image features:
  - Added process isolation: each signal/image is processed in a separate process, so that DataLab does not freeze anymore when processing large signals/images
  - Added support for groups: signals and images can be grouped together, and operations can be applied to all objects in a group, or between groups
  - Added warning and error dialogs with detailed traceback links to the source code (warnings may be optionally ignored)
  - Drastically improved performance when selecting objects
  - Optimized performance when showing large images

- Added support for dropping files on signal/image panel
- Added “Analysis parameters” group box to show last result input parameters
- Added “Copy titles to clipboard” feature in “Edit” menu
- For every single processing feature (operation, processing and analysis menus), the entered parameters (dialog boxes) are stored in cache to be used as defaults the next time the feature is used
- Signal processing:
  - Added support for optional FFT shift (see Settings dialog box)
- Image processing:
  - Added pixel binning operation (X/Y binning factors, operation: sum, mean...)
  - Added “Distribute on a grid” and “Reset image positions” in operation menu
  - Added Butterworth filter
  - Added exposure processing features:
    - \* Gamma correction
    - \* Logarithmic correction
    - \* Sigmoid correction
  - Added restoration processing features:
    - \* Total variation denoising filter (TV Chambolle)
    - \* Bilateral filter (denoising)
    - \* Wavelet denoising filter
    - \* White Top-Hat denoising filter
  - Added morphological transforms (disk footprint):
    - \* White Top-Hat
    - \* Black Top-Hat
    - \* Erosion
    - \* Dilation
    - \* Opening
    - \* Closing
  - Added edge detection features:
    - \* Roberts filter
    - \* Prewitt filter (vertical, horizontal, both)
    - \* Sobel filter (vertical, horizontal, both)
    - \* Scharr filter (vertical, horizontal, both)
    - \* Farid filter (vertical, horizontal, both)
    - \* Laplace filter
    - \* Canny filter
  - Contour detection: added support for polygonal contours (in addition to circle and ellipse contours)

- Added circle Hough transform (circle detection)
- Added image intensity levels rescaling
- Added histogram equalization
- Added adaptative histogram equalization
- Added blob detection methods:
  - \* Difference of Gaussian
  - \* Determinant of Hessian method
  - \* Laplacian of Gaussian
  - \* Blob detection using OpenCV
- Result shapes and annotations are now transformed (instead of removed) when executing one of the following operations:
  - \* Rotation (arbitrary angle,  $+90^\circ$ ,  $-90^\circ$ )
  - \* Symetry (vertical/horizontal)
- Added support for optional FFT shift (see Settings dialog box)
- Console: added configurable external editor (default: VSCode) to follow the traceback links to the source code

---

**Note:** DataLab was created by [CODRA/Pierre Raybaut](#) in 2023. It is developed and maintained by DataLab Platform Developers.

---





## PYTHON MODULE INDEX

### C

- `cdl.algorithms`, 201
- `cdl.algorithms.coordinates`, 218
- `cdl.algorithms.datatypes`, 217
- `cdl.algorithms.image`, 211
- `cdl.algorithms.signal`, 202
- `cdl.computation`, 302
- `cdl.computation.base`, 303
- `cdl.computation.image`, 336
- `cdl.computation.image.detection`, 372
- `cdl.computation.image.edges`, 369
- `cdl.computation.image.exposure`, 359
- `cdl.computation.image.morphology`, 367
- `cdl.computation.image.restoration`, 364
- `cdl.computation.image.threshold`, 357
- `cdl.computation.signal`, 309
- `cdl.core.gui`, 401
- `cdl.core.gui.actionhandler`, 427
- `cdl.core.gui.docks`, 456
- `cdl.core.gui.h5io`, 457
- `cdl.core.gui.main`, 402
- `cdl.core.gui.objectview`, 432
- `cdl.core.gui.panel`, 409
- `cdl.core.gui.panel.base`, 409
- `cdl.core.gui.panel.image`, 420
- `cdl.core.gui.panel.macro`, 423
- `cdl.core.gui.panel.signal`, 417
- `cdl.core.gui.plothandler`, 435
- `cdl.core.gui.processor`, 439
- `cdl.core.gui.processor.base`, 440
- `cdl.core.gui.processor.image`, 450
- `cdl.core.gui.processor.signal`, 445
- `cdl.core.gui.roieditor`, 438
- `cdl.obj`, 264
- `cdl.param`, 220
- `cdl.plugins`, 130
- `cdl.proxy`, 382