DataLab

Release 0.10.1

Dec 22, 2023

CONTENTS:

DataLab is a **generic signal and image processing software** with unique features designed to meet industrial requirements (see *Key strengths*: Extensibility, Interoperability, ...). It is based on Python scientific libraries (such as NumPy, SciPy or scikit-image) and Qt graphical user interfaces (thanks to the powerful PlotPyStack - mostly the guidata and PlotPy libraries).

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.

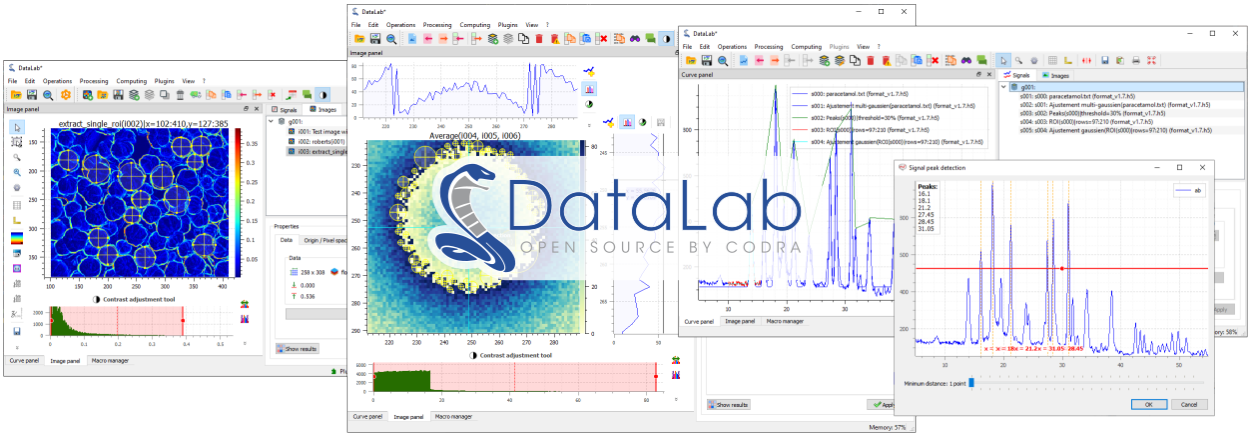


Fig. 1: Signal and image visualization in DataLab

DataLab *Main features* are available not only using the **stand-alone application** (easily installed thanks to the Windows installer or the Python package) but also by **embedding it into your own application** (see the "embedded tests" for detailed examples of how to do so).

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

Note: DataLab was created by Codra/Pierre Raybaut in 2023. It is developed and maintained by DataLab open-source project team with the support of Codra.

External resources:

Home	DataLab home page
PyPI	Python Package Index
GitHub	Bug reports and feature requests

CHAPTER

ONE

GETTING STARTED

1.1 DataLab in a nutshell

DataLab is an open platform for signal and image processing. Its functional scope is intentionally broad. With its many functions, some of them technically advanced, DataLab enables the processing and visualization of all types of scientific data. As a result, scientific, industrial, and innovation stakeholders can have access to an easy-to-use tool that is simple to adapt and offers the reliability of industrial-grade software.

1.2 What are the applications for Datalab?

1.2.1 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 signals acquired on a scientific facility
- Automatic detection laser spot positions in a scene
- Instrument alignment through image processing
- Automatic pattern detection and geometric corrections

1.2.2 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.

Note: DataLab can also be controlled from your familiar development environment (e.g., Visual Studio Code, Spyder, ...) to perform calculations using your processing functions while leveraging the advanced features of DataLab.

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.

1.3 How does DataLab work?

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.

1.3.1 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

1.3.2 Key strengths

DataLab highlights four key strengths:

1. **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.
2. **Interoperability**: DataLab can also be embedded in your own application. For example, within data processing software, machine-level control systems, or test bench applications.
3. **Automation**: a high-level public API allows for full remote control of DataLab to open and process data.
4. **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.

Researchers, engineers, scientists, you will undoubtedly benefit from the capabilities of DataLab. Its open-source software model will also allow you to reinvest your achievements in the open-source community, of which any reputable publisher should be an active member.

Installation

Dependencies

Note: The DataLab Windows installer package already include all those required libraries as well as Python itself.

The `cdl` package requires the following Python modules:

Name	Version	Summary
Python	>=3.8, <4	
h5py	>= 3.0	
NumPy	>= 1.21	
SciPy	>= 1.7	
scikit-image	>= 0.18	
opencv-python-headless	>= 4.5	
PyWavelets	>= 1.1	
psutil	>= 5.5	
guidata	>= 3.2	
PlotPy	>= 2.0	
QtPy	>= 1.9	
PyQt5	>=5.11	Python bindings for the Qt cross platform application toolkit

Optional modules for development:

Name	Version	Summary
black		The uncompromising code formatter.
isort		A Python utility / library to sort Python imports.
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:

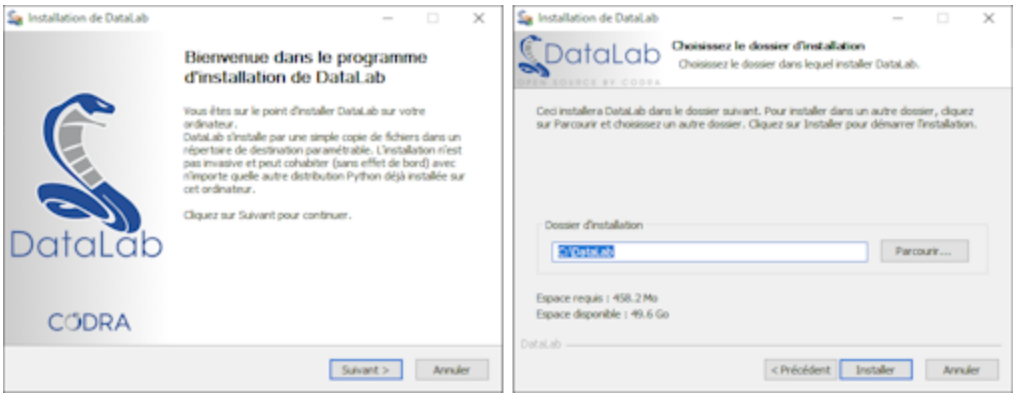
Name	Version	Summary
PyQt5		Python bindings for the Qt cross platform application toolkit
sphinx		Python documentation generator
sphinx_intl		Sphinx utility that make it easy to translate and to apply translation.
myst_parser		An extended [CommonMark](https://spec.commonmark.org/) compliant parser,
pydata-sphinx-theme		Bootstrap-based Sphinx theme from the PyData community

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

How to install

Windows installer:

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!



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

Wheel package:

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-2.0.2-py2.py3-none-any.whl
```

Source package:

Installing DataLab directly from the source package is straigthforward:

```
$ python setup.py install
```

Overview

This page presents briefly DataLab key features.

Data visualization key features

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
	✓	X/Y raw/averaged profiles
✓	✓	Annotations

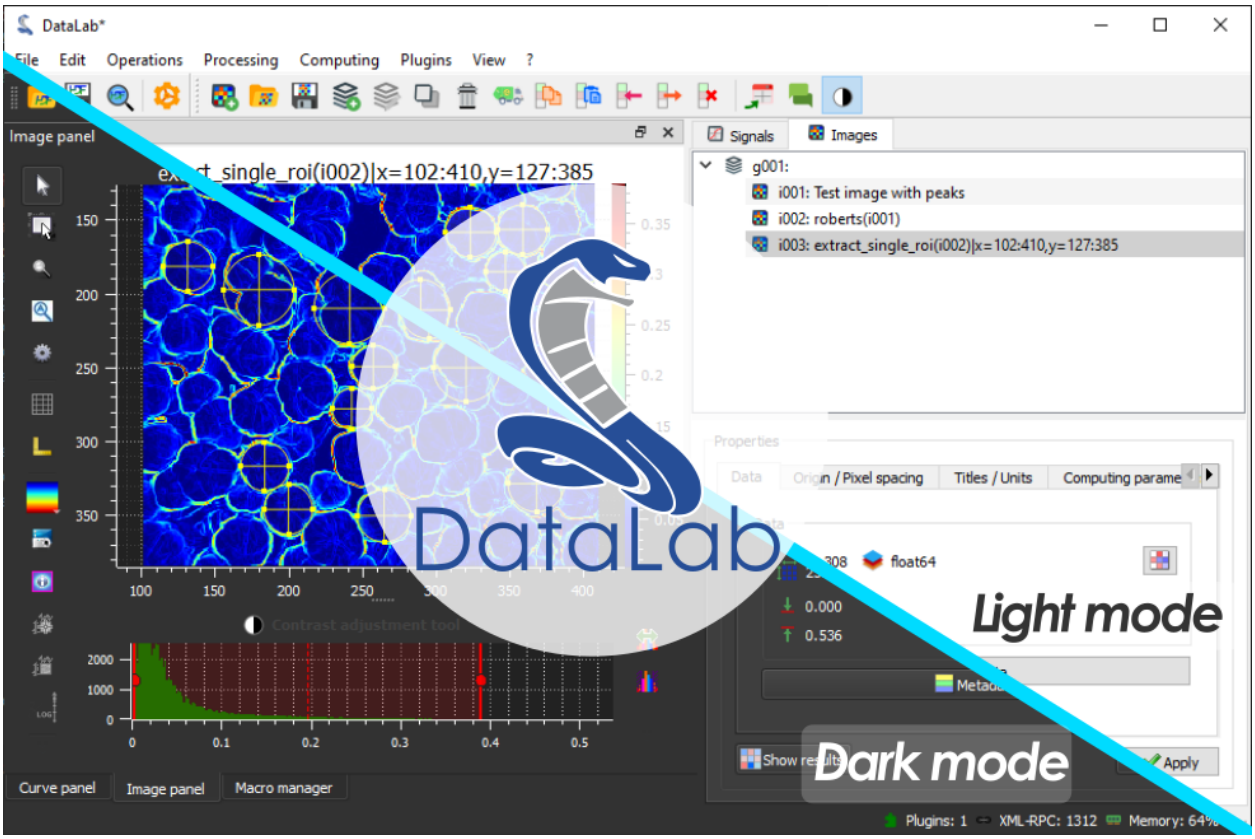


Fig. 1: DataLab supports dark and light mode depending on your OS settings

Fig. 2: Example of a multi-gaussian curve fit

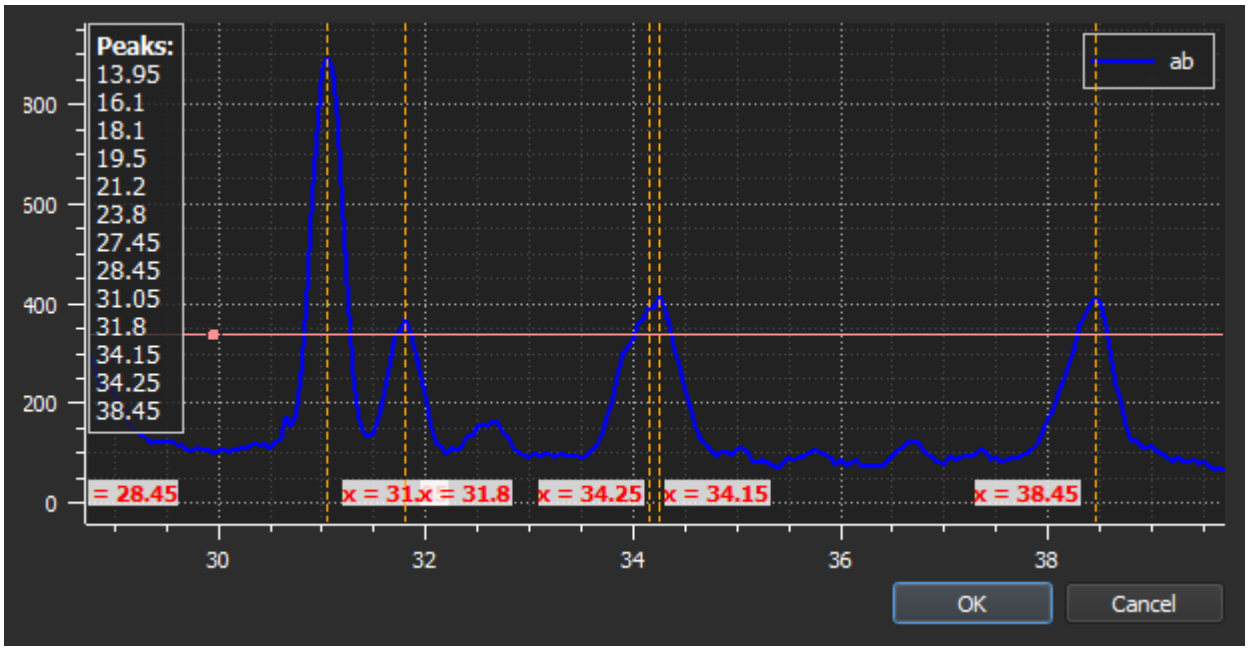


Fig. 3: Semi-automatic peak detection

Data processing key features

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, …
✓	✓	ROI extraction, Swap X/Y axes
✓		Semi-automatic multi-peak detection
	✓	Rotation (flip, rotate), resize, …
	✓	Flat-field correction
✓		Normalize, derivative, integral
✓	✓	Linear calibration
	✓	Thresholding, clipping
✓	✓	Gaussian filter, Wiener filter
✓	✓	Moving average, moving median
✓	✓	FFT, inverse FFT
✓		Interactive fit: Gauss, Lorenzt, Voigt, polynomial
✓		Interactive multigaussian fit
✓	✓	Computing on custom ROI
✓		FWHM, FW @ $1/e^2$
	✓	Centroid (robust method w/r noise)
	✓	Minimum enclosing circle center

CHAPTER

TWO

GENERAL FEATURES

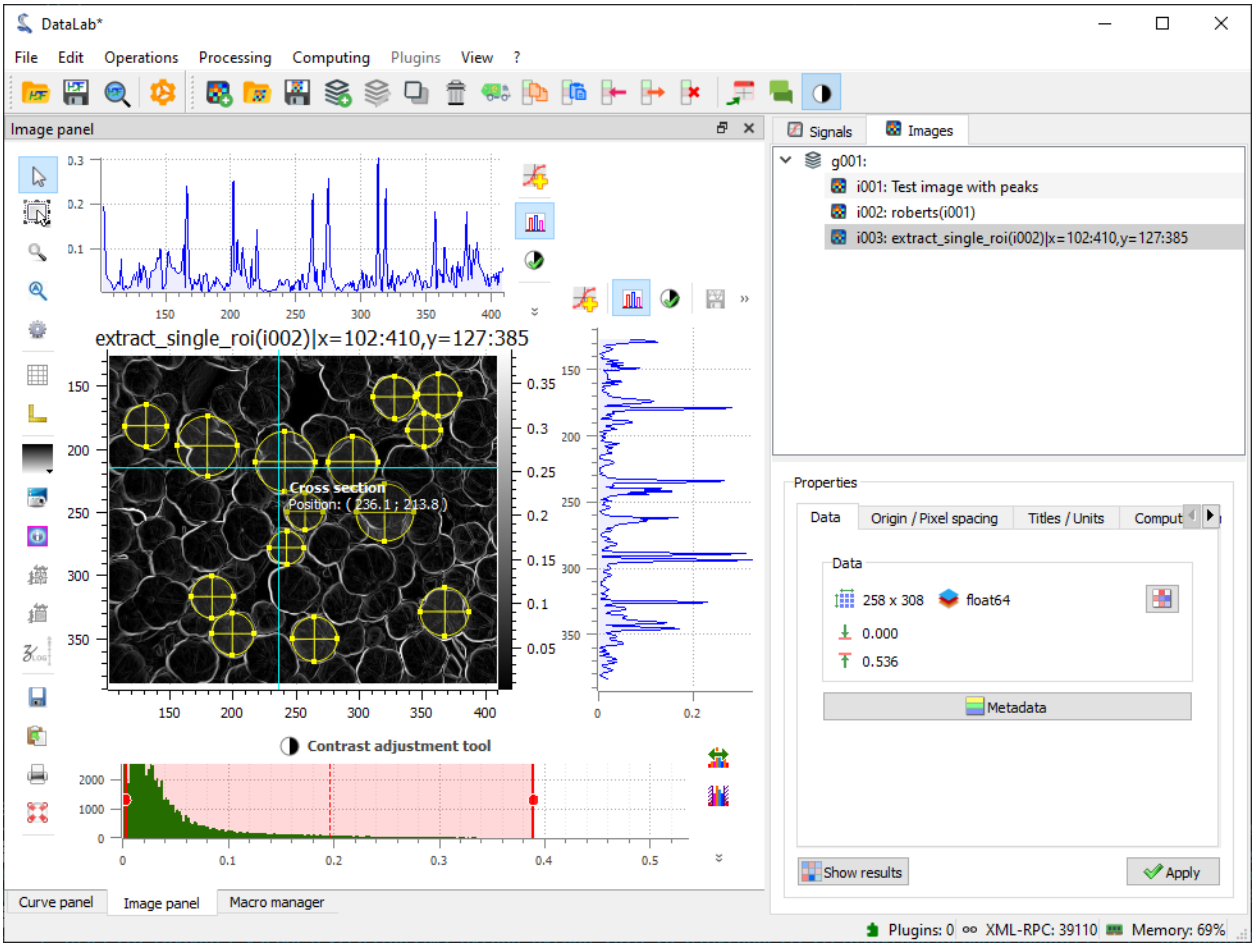


Fig. 1: DataLab main window

2.1 Command line features

2.1.1 Run DataLab

To run DataLab from the command line, type the following:

```
$ cdl
```

To show help on command line usage, simply run:

```
$ cdl --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 ',')

optional arguments:
  -h, --help            show this help message and exit
  -b path, --h5browser path
                        path to open with HDF5 browser
  -v, --version         show DataLab version
  --unattended          non-interactive mode
  --screenshot          automatic screenshots
  --delay DELAY         delay (seconds) before quitting application in unattended mode
  --xmlrpcport XMLRPCPORT
                        XML-RPC port number
  --verbose {quiet,minimal,normal}
                        verbosity level: for debugging/testing purpose
```

2.1.2 Open HDF5 file at startup

To open HDF5 files, or even import only a specified HDF5 dataset, use the following:

```
$ cdl /path/to/file1.h5
$ cdl /path/to/file1.h5,/path/to/dataset1
$ cdl /path/to/file1.h5,/path/to/dataset1;/path/to/file2.h5,/path/to/dataset2
```

2.1.3 Open HDF5 browser at startup

To open the HDF5 browser at startup, use one of the following commands:

```
$ cdl -b /path/to/file1.h5
$ cdl --h5browser /path/to/file1.h5
```

2.1.4 Run DataLab demo

To execute DataLab demo, run the following:

```
$ cdl-demo
```

2.1.5 Run unit tests

Note: This test suite is based on *guidata.guitest* discovery mechanism. It is not compatible with *pytest* because most of the high level tests have to be executed in a separate process (e.g. scenario tests will fail if executed in the same process as other tests).

To execute all DataLab unit tests, simply run:

```
$ cdl-alltests

================================================================================
 DataLab v0.9.0 automatic unit tests
================================================================================

 DataLab characteristics/environment:
Configuration version: 1.0.0
Path: C:\Dev\Projets\DataLab\cdl
Frozen: False
Debug: False

 DataLab configuration:
Process isolation: enabled
RPC server: enabled
Console: enabled
Available memory threshold: 500 MB
Ignored dependencies: disabled
Processing:
    Extract all ROIs in a single signal or image
    FFT shift: enabled

 Test parameters:
 Selected 51 tests (51 total available)
 Test data path:
    C:\Dev\Projets\DataLab\cdl\data\tests
 Environment:
    CDL_DATA=C:\Dev\Projets\DataLab_data\
    PYTHONPATH=.
```

(continues on next page)

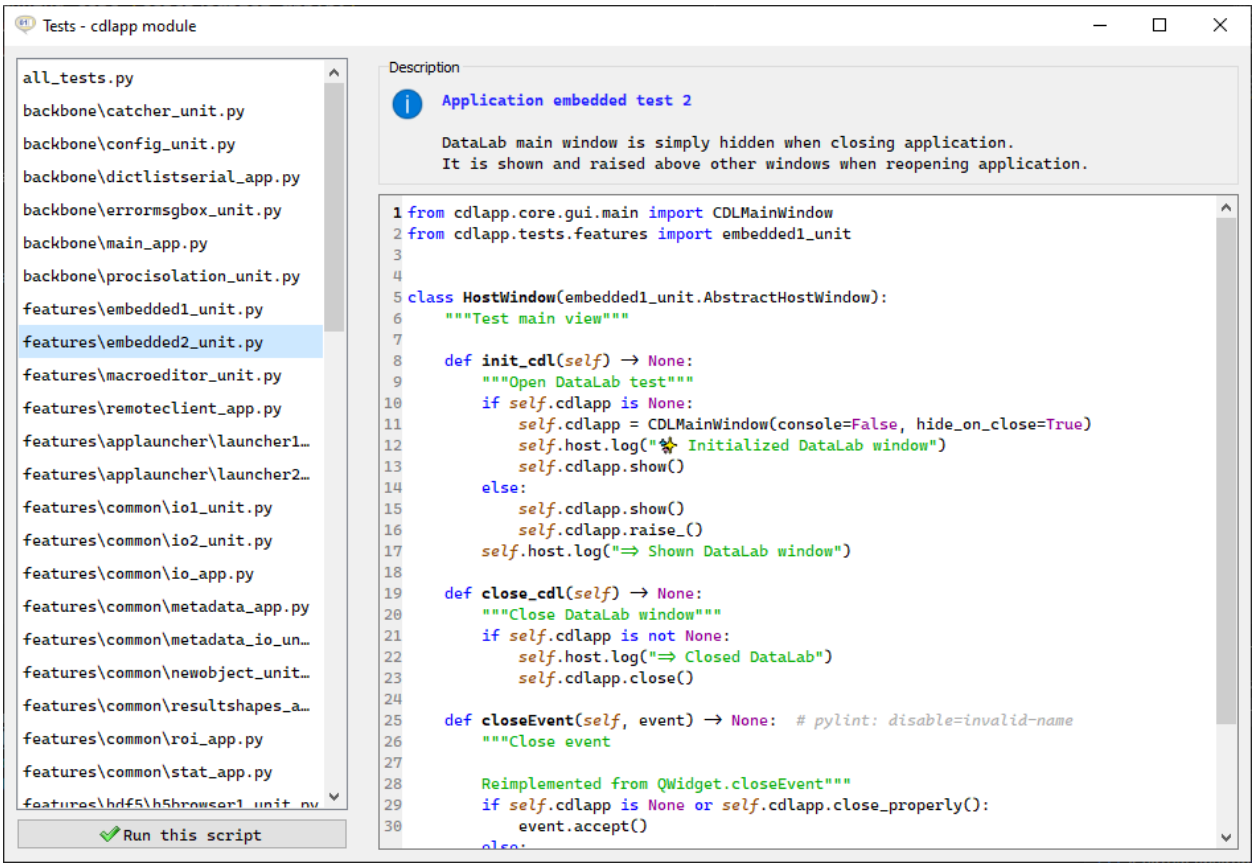
(continued from previous page)

```
    DEBUG=

Please wait while test scripts are executed (a few minutes).
Only error messages will be printed out (no message = test OK).

===[01/51]===  Running test [tests\annotations_app.py]
===[02/51]===  Running test [tests\annotations_unit.py]
===[03/51]===  Running test [tests\auto_app.py]
===[04/51]===  Running test [tests\basic1_app.py]
===[05/51]===  Running test [tests\basic2_app.py]
===[06/51]===  Running test [tests\basic3_app.py]
```

2.1.6 Run interactive tests

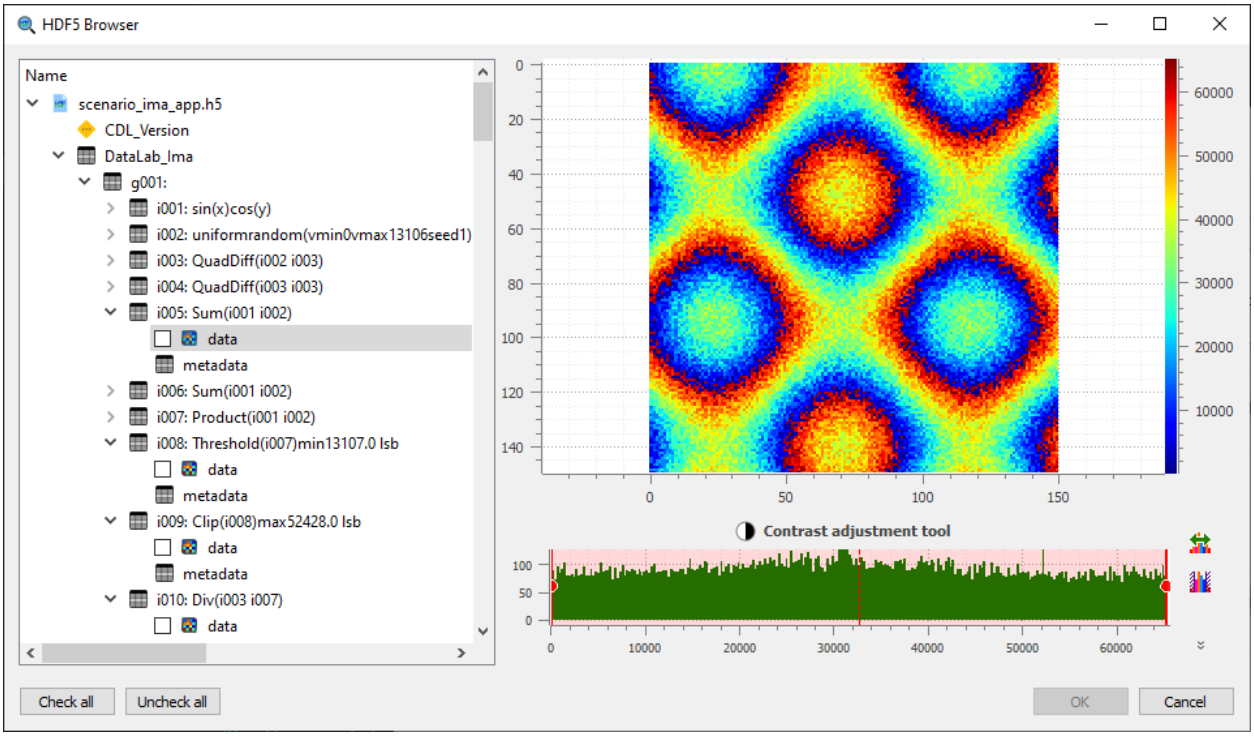
To execute DataLab interactive tests, run the following:

```
$ cdl-tests
```



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 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”

2.3 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 lighweight 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*).

2.3.1 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`.

2.3.2 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.

2.3.3 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`

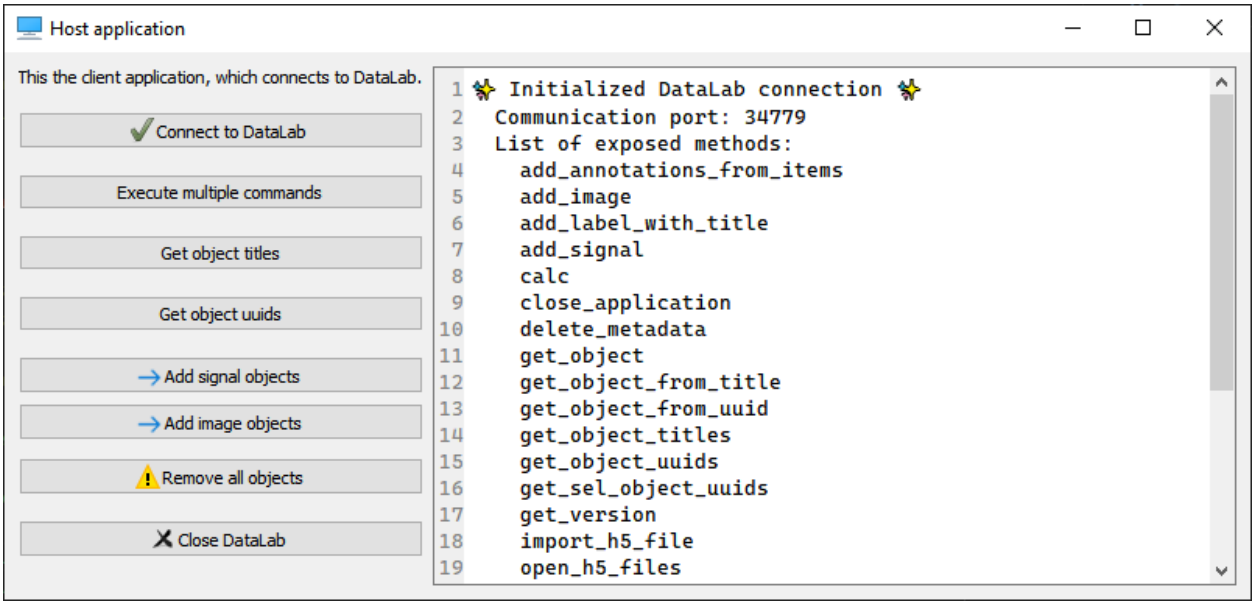


Fig. 2: Screenshot of remote client application test (`cdl.tests.remoteclient_app`)

2.3.4 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):

```
# -*- coding: utf-8 -*-
"""
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 cdl.proxy import RemoteProxy

# %% Connecting to DataLab current session

proxy = RemoteProxy()

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

(continues on next page)

(continued from previous page)

```
z = np.random.rand(20, 20)
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 reimplementation of this class:

```
# -*- coding: utf-8 -*-
#
# Licensed under the terms of the BSD 3-Clause
# (see cdl/LICENSE for details)

"""
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"""
```

(continues on next page)

(continued from previous page)

```
    dbytes = io.BytesIO()
    np.save(dbytes, data, allow_pickle=False)
    return Binary(dbytes.getvalue())


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 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()
```

(continues on next page)

(continued from previous page)

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

    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 open_object(self, filename):
        """Open object from file in current panel (signal/image)"""
        self.serverproxy.open_object(filename)

    def add_signal(
        self, title, xdata, ydata, xunit=None, yunit=None, xlabel=None, ylabel=None
    ):
        """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)

    def add_image(
        self,
        title,
        data,
        xunit=None,
        yunit=None,
        zunit=None,
        xlabel=None,
        ylabel=None,
        zlabel=None,
```

(continues on next page)

(continued from previous page)

```
    ):
        """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)

    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):
        """Get object (signal/image) list for current panel"""
        return self.serverproxy.get_object_uuids(panel)


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()
```

2.3.5 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.

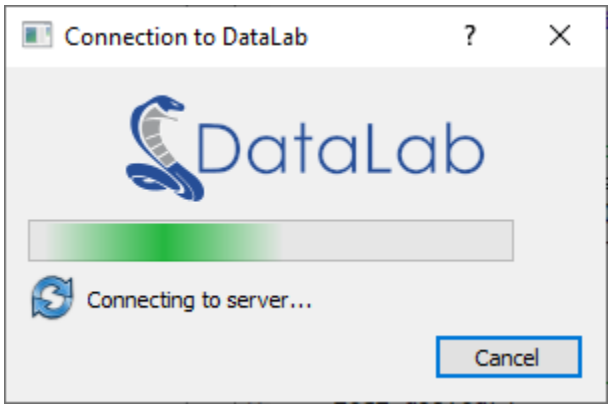


Fig. 3: Screenshot of connection dialog (`cdl.widgets.connection.ConnectionDialog`)

Example of use:

```
# -*- coding: utf-8 -*-
#
# Licensed under the terms of the BSD 3-Clause
# (see cdlclient/LICENSE for details)

"""
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()
```

2.3.6 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.remotecontrol 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
```

(continues on next page)

(continued from previous page)

```
>>> 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** (*str | None*) – XML-RPC port to connect to. If not specified, the port is automatically retrieved from DataLab configuration.
- **timeout** (*float | None*) – Timeout in seconds. Defaults to 5.0.
- **retries** (*int | None*) – 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*) → bool

Add signal data to DataLab.

Parameters

- **title** (*str*) – Signal title
- **xdata** (*numpy.ndarray*) – X data
- **ydata** (*numpy.ndarray*) – Y data
- **xunit** (*str | None*) – X unit. Defaults to None.
- **yunit** (*str | None*) – Y unit. Defaults to None.
- **xlabel** (*str | None*) – X label. Defaults to None.
- **ylabel** (*str | None*) – Y label. Defaults to None.

Returns

True if signal was added successfully, False otherwise

Return type

bool

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*) → bool

Add image data to DataLab.

Parameters

- **title** (*str*) – Image title
- **data** (*numpy.ndarray*) – Image data
- **xunit** (*str | None*) – X unit. Defaults to None.
- **yunit** (*str | None*) – Y unit. Defaults to None.
- **zunit** (*str | None*) – Z unit. Defaults to None.
- **xlabel** (*str | None*) – X label. Defaults to None.
- **ylabel** (*str | None*) – Y label. Defaults to None.
- **zlabel** (*str | None*) – Z label. Defaults to None.

Returns
True if image was added successfully, False otherwise

Return type
bool

Raises
ValueError – Invalid data dtype

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

Call compute function `name` in current panel's processor.

Parameters

- **name** (*str*) – Compute function name
- **param** (*guidata.dataset.DataSet | None*) – Compute function parameter. Defaults to None.

Returns
Compute function result

Return type
guidata.dataset.DataSet

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** (*list*) – annotation plot items
- **refresh_plot** (*bool | None*) – refresh plot. Defaults to True.
- **panel** (*str | None*) – panel name (valid values: "signal", "image"). If None, current panel is used.

add_object(*obj: SignalObj | ImageObj*) → None

Add object to DataLab.

Parameters
obj (SignalObj | ImageObj) – Signal or image object

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

close_application() → None

Close DataLab application

context_no_refresh() → Callable

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*) → None

Delete metadata of selected objects

Parameters
refresh_plot (*bool | None*) – Refresh plot. Defaults to True.

get_current_panel() → str

Return current panel name.

Returns
Panel name (valid values: “signal”, “image”, “macro”))

Return type
str

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 (*str | None*) – panel name (valid values: “signal”, “image”). If None, current panel is used.

Returns
list of object titles

Return type
list[str]

Raises
ValueError – if panel not found

get_object_uuids(*panel: 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 (*str | None*) – panel name (valid values: “signal”, “image”). If None, current panel is used.

Returns
list of object uuids

Return type
list[str]

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

Return type
list[str]

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 version.

Returns
DataLab version

Return type
str

import_h5_file(*filename: str*, *reset_all: bool | None = None*) → None

Open DataLab HDF5 browser to Import HDF5 file.

Parameters

- **filename** (*str*) – HDF5 file name
- **reset_all** (*bool | None*) – Reset all application data. Defaults to None.

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[str] | None*) – List of HDF5 files to open. Defaults to None.
- **import_all** (*bool | None*) – Import all objects from HDF5 files. Defaults to None.
- **reset_all** (*bool | None*) – Reset all application data. Defaults to None.

open_object(*filename: str*) → None

Open object from file in current panel (signal/image).

Parameters
filename (*str*) – File name

raise_window() → None

Raise DataLab window

reset_all() → None

Reset all application data

save_to_h5_file(*filename: str*) → None

Save to a DataLab HDF5 file.

Parameters

filename (*str*) – 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** (*str | None*) – 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 (*str*) – Panel name (valid values: "signal", "image", "macro"))

toggle_auto_refresh(*state: bool*) → None

Toggle auto refresh state.

Parameters

state (*bool*) – Auto refresh state

toggle_show_titles(*state: bool*) → None

Toggle show titles state.

Parameters

state (*bool*) – Show titles state

2.3.7 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 "processor" objects of DataLab.

Signal Processor

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

Object handling signal processing: operations, processing, computing

compute_sum() → None

Compute sum

compute_average() → None

Compute average

compute_product() → None

Compute product

compute_roi_extraction(*param: ROIDataParam | None = None*) → None

Extract Region Of Interest (ROI) from data

compute_swap_axes() → None

Swap data axes

compute_abs() → None

Compute absolute value

compute_re() → None

Compute real part

compute_im() → None

Compute imaginary part

compute_astype(*param: DataTypeSParam | None = None*) → None

Convert data type

compute_log10() → None

Compute Log10

compute_difference(*obj2:* SignalObj *| None = None*) → None

Compute difference between two signals

compute_quadratic_difference(*obj2:* SignalObj *| None = None*) → None

Compute quadratic difference between two signals

compute_division(*obj2:* SignalObj *| None = None*) → None

Compute division between two signals

compute_peak_detection(*param: PeakDetectionParam | None = None*) → None

Detect peaks from data

compute_normalize(*param: NormalizeYParam | None = None*) → None

Normalize data

compute_derivative() → None

Compute derivative

compute_integral() → None

Compute integral

compute_calibration(*param: XYCalibrateParam | None = None*) → None
Compute data linear calibration

compute_threshold(*param: ThresholdParam | None = None*) → None
Compute threshold clipping

compute_clip(*param: ClipParam | None = None*) → None
Compute maximum data clipping

compute_gaussian_filter(*param: GaussianParam | None = None*) → None
Compute gaussian filter

compute_moving_average(*param: MovingAverageParam | None = None*) → None
Compute moving average

compute_moving_median(*param: MovingMedianParam | None = None*) → None
Compute moving median

compute_wiener() → None
Compute Wiener filter

compute_fft(*param: FFTParam | None = None*) → None
Compute iFFT

compute_ifft(*param: FFTParam | None = None*) → None
Compute FFT

compute_fit(*name*, *fitdlgfunc*)
Compute fitting curve

compute_polyfit(*param: PolynomialFitParam | None = None*) → None
Compute polynomial fitting curve

compute_multigaussianfit() → None
Compute multi-Gaussian fitting curve

compute_fwhm(*param: FWHMParam | None = None*) → None
Compute FWHM

compute_fw1e2() → None
Compute FW at $1/e^2$

Image Processor

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

Object handling image processing: operations, processing, computing

compute_sum() → None
Compute sum

compute_average() → None
Compute average

compute_product() → None
Compute product

compute_logp1(*param: LogP1Param | None = None*) → None
Compute base 10 logarithm

compute_rotate(*param: RotateParam | None = None*) → None
Rotate data arbitrarily

compute_rotate90() → None
Rotate data 90°

compute_rotate270() → None
Rotate data 270°

compute_fliph() → None
Flip data horizontally

compute_flipv() → None
Flip data vertically

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

compute_binning(*param: BinningParam | None = None*) → None
Binning image

compute_roi_extraction(*param: ROIDataParam | None = None*) → None
Extract Region Of Interest (ROI) from data

compute_profile(*param: ProfileParam | None = None*) → None
Compute profile

compute_average_profile(*param: AverageProfileParam | None = None*) → None
Compute average profile

compute_swap_axes() → None
Swap data axes

compute_abs() → None
Compute absolute value

compute_re() → None
Compute real part

compute_im() → None
Compute imaginary part

compute_astype(*param: DataTypeIParam | None = None*) → None
Convert data type

compute_log10() → None
Compute Log10

compute_difference(*obj2:* ImageObj | *None* = *None*) → None
Compute difference between two images

compute_quadratic_difference(*obj2:* ImageObj | *None* = *None*) → None
Compute quadratic difference between two images

compute_division(*obj2:* ImageObj | *None* = *None*) → None
Compute division between two images

compute_flatfield(*obj2:* ImageObj | *None* = *None*, *param: FlatFieldParam* | *None* = *None*) → None
Compute flat field correction

compute_calibration(*param: ZCalibrateParam* | *None* = *None*) → None
Compute data linear calibration

compute_threshold(*param: ThresholdParam* | *None* = *None*) → None
Compute threshold clipping

compute_clip(*param: ClipParam* | *None* = *None*) → None
Compute maximum data clipping

compute_gaussian_filter(*param: GaussianParam* | *None* = *None*) → None
Compute gaussian filter

compute_moving_average(*param: MovingAverageParam* | *None* = *None*) → None
Compute moving average

compute_moving_median(*param: MovingMedianParam* | *None* = *None*) → None
Compute moving median

compute_wiener() → None
Compute Wiener filter

compute_fft(*param: FFTParam* | *None* = *None*) → None
Compute FFT

compute_ifft(*param: FFTParam* | *None* = *None*) → None
Compute iFFT

compute_butterworth(*param: ButterworthParam* | *None* = *None*) → None
Compute Butterworth filter

compute_adjust_gamma(*param: AdjustGammaParam* | *None* = *None*) → None
Compute gamma correction

compute_adjust_log(*param: AdjustLogParam* | *None* = *None*) → None
Compute log correction

compute_adjust_sigmoid(*param: AdjustSigmoidParam* | *None* = *None*) → None
Compute sigmoid correction

compute_rescale_intensity(*param: RescaleIntensityParam* | *None* = *None*) → None
Rescale image intensity levels

compute_equalize_hist(*param: EqualizeHistParam* | *None* = *None*) → None
Histogram equalization

compute_equalize_adapthist(*param: EqualizeAdaptHistParam | None = None*) → None
Adaptive histogram equalization

compute_denoise_tv(*param: DenoiseTVParam | None = None*) → None
Compute Total Variation denoising

compute_denoise_bilateral(*param: DenoiseBilateralParam | None = None*) → None
Compute bilateral filter denoising

compute_denoise_wavelet(*param: DenoiseWaveletParam | None = None*) → None
Compute Wavelet denoising

compute_denoise_tophat(*param: MorphologyParam | None = None*) → None
Denoise using White Top-Hat

compute_all_denoise(*params: list | None = None*) → None
Compute all denoising filters

compute_white_tophat(*param: MorphologyParam | None = None*) → None
Compute White Top-Hat

compute_black_tophat(*param: MorphologyParam | None = None*) → None
Compute Black Top-Hat

compute_erosion(*param: MorphologyParam | None = None*) → None
Compute Erosion

compute_dilation(*param: MorphologyParam | None = None*) → None
Compute Dilation

compute_opening(*param: MorphologyParam | None = None*) → None
Compute morphological opening

compute_closing(*param: MorphologyParam | None = None*) → None
Compute morphological closing

compute_all_morphology(*param: MorphologyParam | None = None*) → None
Compute all morphology filters

compute_canny(*param: CannyParam | None = None*) → None
Compute Canny filter

compute_roberts() → None
Compute Roberts filter

compute_prewitt() → None
Compute Prewitt filter

compute_prewitt_h() → None
Compute Prewitt filter (horizontal)

compute_prewitt_v() → None
Compute Prewitt filter (vertical)

compute_sobel() → None
Compute Sobel filter

compute_sobel_h() → None
Compute Sobel filter (horizontal)

compute_sobel_v() → None
Compute Sobel filter (vertical)

compute_scharr() → None
Compute Scharr filter

compute_scharr_h() → None
Compute Scharr filter (horizontal)

compute_scharr_v() → None
Compute Scharr filter (vertical)

compute_farid() → None
Compute Farid filter

compute_farid_h() → None
Compute Farid filter (horizontal)

compute_farid_v() → None
Compute Farid filter (vertical)

compute_laplace() → None
Compute Laplace filter

compute_all_edges() → None
Compute all edges

compute_centroid() → None
Compute image centroid

compute_enclosing_circle() → None
Compute minimum enclosing circle

compute_peak_detection(*param: Peak2DDetectionParam | None = None*) → None
Compute 2D peak detection

compute_contour_shape(*param: ContourShapeParam | None = None*) → None
Compute contour shape fit

compute_hough_circle_peaks(*param: HoughCircleParam | None = None*) → None
Compute peak detection based on a circle Hough transform

compute_blob_dog(*param: BlobDOGParam | None = None*) → None
Compute blob detection using Difference of Gaussian method

compute_blob_doh(*param: BlobDOHParam | None = None*) → None
Compute blob detection using Determinant of Hessian method

compute_blob_log(*param: BlobLOGParam | None = None*) → None
Compute blob detection using Laplacian of Gaussian method

compute_blob_opencv(*param: BlobOpenCVParam | None = None*) → None
Compute blob detection using OpenCV

2.4 Internal data model

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

- *cdl.core.model.signal.SignalObj*, which represents a signal object, and
- *cdl.core.model.image.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.

2.4.1 Public API

The public API is the following:

DataLab Public API's object model module

This modules 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`

This module is thus a convenient way to import all the objects at once.

DataLab Base Computation parameters module

This modules aims at providing all the dataset parameters that are used by the `cdl.core.gui.processor` module.

Those datasets are defined other modules:

- `cdl.core.computation.base`
- `cdl.core.computation.image`
- `cdl.core.computation.signal`

This module is thus a convenient way to import all the parameters at once.

Signal object and related classes

class cdl.core.model.signal.**CurveStyles**

Bases: `object`

Object to manage curve styles

style_generator()

Cycling through curve styles

classmethod apply_style(*param: CurveParam*)

Apply style to curve

class cdl.core.model.signal.**ROIParam**(*title: str | None = None*, *comment: str | None = None*, *icon: str = ''*)

Bases: DataSet

Signal ROI parameters

class cdl.core.model.signal.**SignalObj**(*title=None*, *comment=None*, *icon=''*)

Bases: DataSet, BaseObj

Signal object

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** (*str*) – title
- **dtype** (*numpy.dtype*) – data type

Returns
copied object

Return type
SignalObj

set_data_type(*dtype: dtype*) → None

Change data type.

Parameters
dtype (*numpy.dtype*) – data type

set_xydata(*x: ndarray | list*, *y: ndarray | list*, *dx: ndarray | list | None = None*, *dy: ndarray | list | None = None*) → None

Set xy data

Parameters

- **x** (*numpy.ndarray*) – x data
- **y** (*numpy.ndarray*) – y data
- **dx** (*numpy.ndarray*) – dx data (optional: error bars)
- **dy** (*numpy.ndarray*) – 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*) → 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 (*int*) – ROI index

Returns
data

Return type
numpy.ndarray

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 overriden 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*) → CurveItem

Make plot item from data.

Parameters
update_from (*CurveItem*) – plot item to update from

Returns
plot item

Return type
CurveItem

update_item(*item: CurveItem*, *data_changed: bool = True*) → None

Update plot item from data.

Parameters

- **item** (*CurveItem*) – plot item
- **data_changed** (*bool*) – if True, data has changed

roi_coords_to_indexes(*coords: list*) → ndarray

Convert ROI coordinates to indexes.

Parameters
coords (*list*) – coordinates

Returns
indexes

Return type
numpy.ndarray

get_roi_param(*title: str*, **defaults*) → DataSet

Return ROI parameters dataset.

Parameters

- **title** (*str*) – title
- ***defaults** – default values

static params_to_roidata(*params: DataSetGroup*) → ndarray

Convert ROI dataset group to ROI array data.

Parameters
params (*DataSetGroup*) – ROI dataset group

Returns
ROI array data

Return type
numpy.ndarray

new_roi_item(*fmt: str*, *lbl: bool*, *editable: bool*)

Return a new ROI item from scratch

Parameters

- **fmt** (*str*) – format string
- **lbl** (*bool*) – if True, add label
- **editable** (*bool*) – if True, ROI is editable

iterate_roi_items(*fmt: str*, *lbl: bool*, *editable: bool = True*)

Make plot item representing a Region of Interest.

Parameters

- **fmt** (*str*) – format string
- **lbl** (*bool*) – if True, add label
- **editable** (*bool*) – if True, ROI is editable

Yields
PlotItem – plot item

add_label_with_title(*title: str | None = None*) → None

Add label with title annotation

Parameters
title (*str*) – title (if None, use signal title)

cdl.core.model.signal.**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 | None = None*, *labels: tuple | None = None*) → *SignalObj*

Create a new Signal object.

Parameters

- **title** (*str*) – signal title
- **x** (*numpy.ndarray*) – X data
- **y** (*numpy.ndarray*) – Y data
- **dx** (*numpy.ndarray*) – dX data (optional: error bars)
- **dy** (*numpy.ndarray*) – dY data (optional: error bars)
- **metadata** (*dict*) – signal metadata
- **units** (*tuple*) – X, Y units (tuple of strings)
- **labels** (*tuple*) – X, Y labels (tuple of strings)

Returns
signal object

Return type
SignalObj

class cdl.core.model.signal.**SignalTypes**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: Choices

Signal types

class cdl.core.model.signal.**GaussLorentzVoigtParam**(*title: str | None = None*, *comment: str | None = None*, *icon: str = ''*)

Bases: DataSet

Parameters for Gaussian and Lorentzian functions

class cdl.core.model.signal.**FreqUnits**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: Choices

Frequency units

classmethod convert_in_hz(*value*, *unit*)

Convert value in Hz

class cdl.core.model.signal.**PeriodicParam**(*title: str | None = None*, *comment: str | None = None*, *icon: str = ''*)

Bases: DataSet

Parameters for periodic functions

get_frequency_in_hz()

Return frequency in Hz

class cdl.core.model.signal.**StepParam**(*title: str | None = None*, *comment: str | None = None*, *icon: str = ''*)

Bases: DataSet

Parameters for step function

class cdl.core.model.signal.**NewSignalParam**(*title: str | None = None*, *comment: str | None = None*, *icon: str = ''*)

Bases: DataSet

New signal dataset

cdl.core.model.signal.**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** (*str*) – dataset title (default: None, uses default title)
- **stype** (*str*) – signal type (default: None, uses default type)
- **xmin** (*float*) – X min (default: None, uses default value)
- **xmax** (*float*) – X max (default: None, uses default value)
- **size** (*int*) – signal size (default: None, uses default value)

Returns
new signal dataset instance

Return type
NewSignalParam

cdl.core.model.signal.**triangle_func**(*xarr: ndarray*) → ndarray

Triangle function

Parameters
xarr (*numpy.ndarray*) – x data

cdl.core.model.signal.**create_signal_from_param**(*newparam: NewSignalParam*, *addparam: gds.DataSet | None = None*, *edit: bool = False*, *parent: QW.QWidget | None = None*) → *SignalObj* | None

Create a new Signal object from a dialog box.

Parameters

- **newparam** (NewSignalParam) – new signal parameters
- **addparam** (*guidata.dataset.DataSet*) – additional parameters
- **edit** (*bool*) – Open a dialog box to edit parameters (default: False)
- **parent** (*QWidget*) – parent widget

Returns
signal object or None if canceled

Return type
SignalObj

Image object and related classes

cdl.core.model.image.**make_roi_rectangle**(*x0: int*, *y0: int*, *x1: int*, *y1: int*, *title: str*) → AnnotatedRectangle

Make and return the annnotated rectangle associated to ROI

Parameters

- **x0** (*int*) – top left corner X coordinate
- **y0** (*int*) – top left corner Y coordinate
- **x1** (*int*) – bottom right corner X coordinate
- **y1** (*int*) – bottom right corner Y coordinate
- **title** (*str*) – title

cdl.core.model.image.**make_roi_circle**(*x0: int*, *y0: int*, *x1: int*, *y1: int*, *title: str*) → AnnotatedCircle

Make and return the annnotated circle associated to ROI

Parameters

- **x0** (*int*) – top left corner X coordinate
- **y0** (*int*) – top left corner Y coordinate
- **x1** (*int*) – bottom right corner X coordinate
- **y1** (*int*) – bottom right corner Y coordinate
- **title** (*str*) – title

cdl.core.model.image.**to_builtin**(*obj*) → str | int | float | list | dict | ndarray | None

Convert an object implementing a numeric value or collection into the corresponding builtin/NumPy type.

Return None if conversion fails.

class cdl.core.model.image.**RoiDataGeometries**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: Enum

ROI data geometry types

class cdl.core.model.image.**RoiDataItem**(*data: ndarray | list | tuple*)

Bases: object

Object representing an image ROI.

Parameters
data (*numpy.ndarray | list | tuple*) – ROI data

classmethod from_image(*obj*, *geometry:* RoiDataGeometries) → *RoiDataItem*

Construct roi data item from image object: called for making new ROI items

Parameters

- **obj** (ImageObj) – image object
- **geometry** (RoiDataGeometries) – ROI geometry

property geometry: *RoiDataGeometries*

ROI geometry

get_rect() → tuple[int, int, int, int]

Get rectangle coordinates

get_masked_view(*data: ndarray*, *maskdata: ndarray*) → ndarray

Return masked view for data

Parameters

- **data** (*numpy.ndarray*) – data
- **maskdata** (*numpy.ndarray*) – mask data

apply_mask(*data: ndarray*, *yxratio: float*) → ndarray

Apply ROI to data as a mask and return masked array

Parameters

- **data** (*numpy.ndarray*) – data
- **yxratio** (*float*) – Y/X ratio

make_roi_item(*index: int | None*, *fmt: str*, *lbl: bool*, *editable: bool = True*)

Make ROI plot item

Parameters

- **index** (*int | None*) – ROI index
- **fmt** (*str*) – format string
- **lbl** (*bool*) – if True, show label
- **editable** (*bool*) – if True, ROI is editable

cdl.core.model.image.**roi_label**(*name: str*, *index: int*)

Returns name_{index}

class cdl.core.model.image.**RectangleROIParam**(*title: str | None = None*, *comment: str | None = None*, *icon: str = ''*)

Bases: DataSet

ROI parameters

get_suffix()

Get suffix text representation for ROI extraction

get_coords()

Get ROI coordinates

class cdl.core.model.image.**CircularROIParam**(*title: str | None = None*, *comment: str | None = None*, *icon: str = ''*)

Bases: DataSet

ROI parameters

get_single_roi()

Get single circular ROI, i.e. after extracting ROI from image

get_suffix()

Get suffix text representation for ROI extraction

get_coords()

Get ROI coordinates

property x0

Return rectangle top left corner X coordinate

property x1

Return rectangle bottom right corner X coordinate

property y0

Return rectangle top left corner Y coordinate

property y1

Return rectangle bottom right corner Y coordinate

class cdl.core.model.image.**ImageObj**(*title=None*, *comment=None*, *icon=''*)

Bases: DataSet, BaseObj

Image object

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.

property size: tuple[int, int]

Returns (width, height)

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 (*Mapping | dict*) – object

property dicom_template

Get DICOM template

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 (*int*) – ROI index

Returns
masked data

Return type
numpy.ndarray

copy(*title: str | None = None*, *dtype: dtype | None = None*) → *ImageObj*

Copy object.

Parameters

- **title** (*str*) – title

- **dtype** (*numpy.dtype*) – data type

Returns
copied object

Return type
ImageObj

set_data_type(*dtype: dtype*) → None

Change data type.

Parameters
dtype (*numpy.dtype*) – data type

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 overriden 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 (*MaskedImageItem | None*) – update from plot item

Returns
plot item

Return type
MaskedImageItem

update_item(*item: MaskedImageItem*, *data_changed: bool = True*) → None

Update plot item from data.

Parameters

- **item** (*MaskedImageItem*) – plot item
- **data_changed** (*bool*) – if True, data has changed

get_roi_param(*title*, **defaults*) → DataSet

Return ROI parameters dataset.

Parameters

- **title** (*str*) – title
- ***defaults** – default values

static params_to_roidata(*params: DataSetGroup*) → ndarray | None

Convert ROI dataset group to ROI array data.

Parameters
params (*DataSetGroup*) – ROI dataset group

Returns
ROI array data

Return type
numpy.ndarray

new_roi_item(*fmt: str*, *lbl: bool*, *editable: bool*, *geometry:* RoiDataGeometries) → MaskedImageItem

Return a new ROI item from scratch

Parameters

- **fmt** (*str*) – format string
- **lbl** (*bool*) – if True, add label
- **editable** (*bool*) – if True, ROI is editable
- **geometry** (RoiDataGeometries) – ROI geometry

roi_coords_to_indexes(*coords: list*) → ndarray

Convert ROI coordinates to indexes.

Parameters
coords (*list*) – coordinates

Returns
indexes

Return type
numpy.ndarray

iterate_roi_items(*fmt: str*, *lbl: bool*, *editable: bool = True*) → Iterator

Make plot item representing a Region of Interest.

Parameters

- **fmt** (*str*) – format string
- **lbl** (*bool*) – if True, add label
- **editable** (*bool*) – if True, ROI is editable

Yields
PlotItem – plot item

property maskdata: ndarray

Return masked data (areas outside defined regions of interest)

Returns
masked data

Return type
numpy.ndarray

invalidate_maskdata_cache() → None

Invalidate mask data cache: force to rebuild it

add_label_with_title(*title: str | None = None*) → None

Add label with title annotation

Parameters
title (*str*) – title (if None, use image title)

cdl.core.model.image.**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** (*str*) – image title
- **data** (*numpy.ndarray*) – image data
- **metadata** (*dict*) – image metadata
- **units** (*tuple*) – X, Y, Z units (tuple of strings)
- **labels** (*tuple*) – X, Y, Z labels (tuple of strings)

Returns
image object

Return type
ImageObj

class cdl.core.model.image.**ImageDatatypes**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: Choices

Image data types

classmethod from_dtype(*dtype*)

Return member from NumPy dtype

classmethod check()

Check if data types are valid

class cdl.core.model.image.**ImageTypes**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: Choices

Image types

class cdl.core.model.image.**NewImageParam**(*title: str | None = None*, *comment: str | None = None*, *icon: str = ''*)

Bases: DataSet

New image dataset

cdl.core.model.image.**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** (*str*) – dataset title (default: None, uses default title)
- **itype** (ImageTypes) – image type (default: None, uses default type)

- **height** (*int*) – image height (default: None, uses default height)
- **width** (*int*) – image width (default: None, uses default width)
- **dtype** (ImageDatatypes) – image data type (default: None, uses default data type)

Returns
new image dataset instance

Return type
NewImageParam

class cdl.core.model.image.**Gauss2DParam**(*title: str | None = None*, *comment: str | None = None*, *icon: str = ''*)

Bases: DataSet

2D Gaussian parameters

cdl.core.model.image.**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** (NewImageParam) – new image parameters
- **addparam** (*guidata.dataset.DataSet*) – additional parameters
- **edit** (*bool*) – Open a dialog box to edit parameters (default: False)
- **parent** (*QWidget*) – parent widget

Returns
new image object or None if user cancelled

Return type
ImageObj

2.5 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.

2.5.1 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 that contains a class derived from the *PluginBase* class. The name of the class is not important, as long as it is derived from *PluginBase*. The class must have a *PLUGIN_INFO* attribute that is an instance of the *PluginInfo* class. The *PLUGIN_INFO* attribute is used by DataLab to retrieve information about the plugin.

2.5.2 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).

2.5.3 Example: processing plugin

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

```
# -*- coding: utf-8 -*-
#
# Licensed under the terms of the BSD 3-Clause
# (see cdl/LICENSE for details)

"""
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.config import _
from cdl.core.computation import image as cpima
from cdl.core.computation import signal as cpsig
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(
```

(continues on next page)

(continued from previous page)

```
    src: dlo.SignalObj, p: test_data.GaussianNoiseParam
) -> dlo.SignalObj:
    """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.signalpanel):
                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(
```

(continues on next page)

(continued from previous page)

```
            add_noise_to_image,
            paramclass=dlo.NormalRandomParam,
            title=_("Add noise"),
        )

    def create_peak2d_image(self) -> None:
        """Create 2D peak image"""
        obj = self.imagepanel.new_object(add_to_panel=False)
        param = test_data.PeakDataParam.create(size=max(obj.data.shape))
        self.imagepanel.processor.update_param_defaults(param)
        if param.edit(self.imagepanel):
            obj.data = test_data.get_peak2d_data(param)
            self.proxy.add_object(obj)

    def __get_newimageparam(self):
        """Create new image parameter dataset"""
        newparam = self.imagepanel.get_newparam_from_current()
        newparam.hide_image_type = True
        if newparam.edit(self.imagepanel):
            return newparam
        return None

    def create_sincos_image(self) -> None:
        """Create 2D sin cos image"""
        newparam = self.__get_newimageparam()
        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.__get_newimageparam()
        if newparam is not None:
            obj = test_data.create_noisygauss_image(newparam)
            self.proxy.add_object(obj)

    def create_multigauss_image(self) -> None:
        """Create 2D multi gauss image"""
        newparam = self.__get_newimageparam()
        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.__get_newimageparam()
        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"""
```

(continues on next page)

(continued from previous page)

```
        param = test_data.RingParam(_("Ring"))
        if param.edit(self.imagepanel):
            obj = test_data.create_ring_image(param)
            self.proxy.add_object(obj)

    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,
```

(continues on next page)

(continued from previous page)

```
                select_condition="always",
            )
            iah.new_action(
                _("Create annotated image"),
                triggered=self.create_annotated_image,
                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",
            )
```

2.5.4 Example: input/output plugin

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

```
# -*- coding: utf-8 -*-
#
# Licensed under the terms of the BSD 3-Clause
# (see cdl/LICENSE for details)

"""
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:
```

(continues on next page)

(continued from previous page)

```
        fname (str): path to FXD file
        debug (bool): debug mode
    """

    HEADER = "<llllllffl"

    def __init__(self, fname: str = None, debug: bool = False) -> None:
        self.__debug = debug
        self.file_format = None  # long
        self.nbcols = 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.nbcols: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)
```

(continues on next page)

(continued from previous page)

```
            unpacked_rec = header_s.unpack(record)
            (
                self.file_format,
                self.nbcols,
                self.nbrows,
                self.nbframes,
                self.pixeltype,
                self.quantlevels,
                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.nbcols * size)
        data = np.fromstring(block, dtype=dtype)
        self.data = data.reshape(self.nbrows, self.nbcols)


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
```

(continues on next page)

(continued from previous page)

```python
# ============================================================================
# Dürr NDT XYZ file format
# ============================================================================


class XYZImageFormat(ImageFormatBase):
    """Object representing Dürr NDT XYZ image file type"""

    FORMAT_INFO = FormatInfo(
        name="Dürr NDT",
        extensions="*.xyz",
        readable=True,
        writeable=False,
    )

    @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)
```

2.5.5 Other examples

Other examples of plugins can be found in the *plugins/examples* directory of the DataLab source code (explore here on GitHub).

2.5.6 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[*PluginBase*]

Return plugin classes

classmethod get_plugins() → list[*PluginBase*]

Return plugin instances

classmethod get_plugin(*name_or_class*) → *PluginBase* | None

Return plugin instance

classmethod register_plugin(*plugin:* PluginBase)

Register plugin

classmethod unregister_plugin(*plugin:* PluginBase)

Unregister plugin

classmethod get_plugin_infos() → str

Return plugin infos (names, versions, descriptions) in html format

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_yesno(*message: str*, *title: str | None = None*, *cancelable: bool = False*) → bool

Ask yes/no question

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[*PluginBase*]

Discover plugins using naming convention

2.6 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)

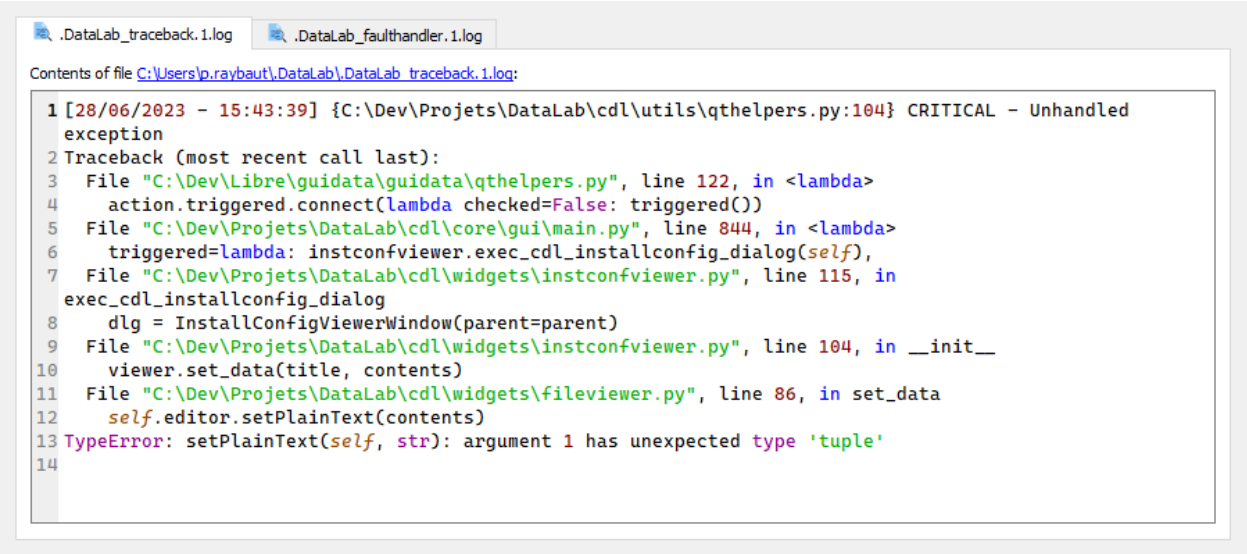


Fig. 4: DataLab log viewer (see "?" menu)

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*).

2.7 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.

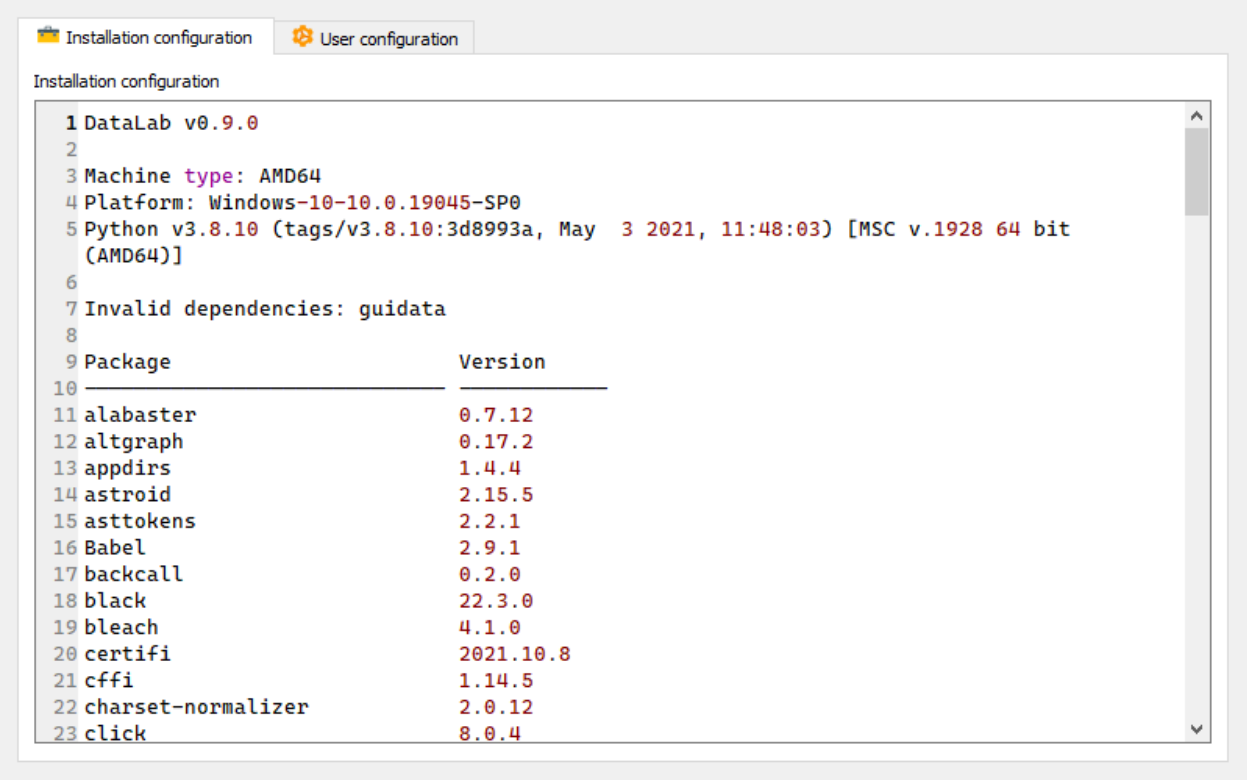


Fig. 5: Installation and configuration (see “?” menu)

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*).

CHAPTER

THREE

SIGNAL PROCESSING

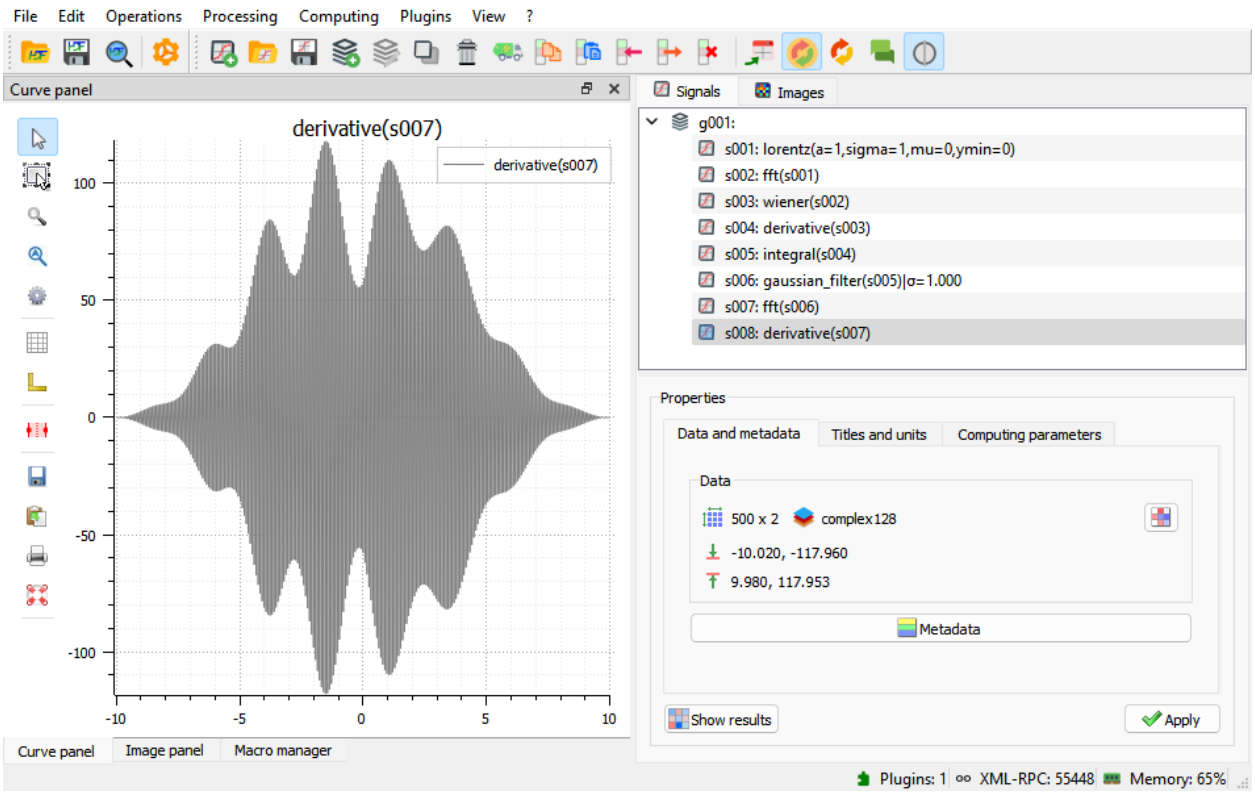
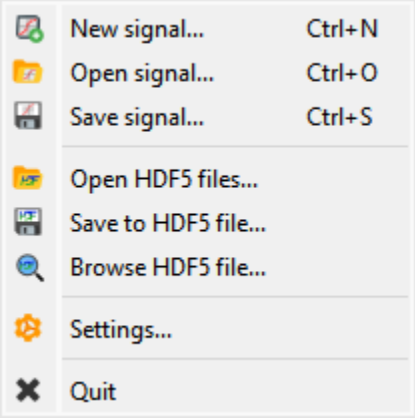


Fig. 1: DataLab main window: Signal processing view

3.1 “File” menu



New signal
Create a new signal from various models:

Model	Equation
Zeros	$y[i] = 0$
Random	$y[i] \in [-0.5, 0.5]$
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	$y = y_0 + A.\frac{Re(exp(-z^2).erfc(-j.z))}{\sqrt{2\pi}.\sigma}$ with $z = \frac{x - x_0 - j.\sigma}{\sqrt{2}.\sigma}$

Open signal
Create a new signal from the following supported filetypes:

File type	Extensions
Text files	.txt, .csv
NumPy arrays	.npy

Save signal
Save current signal to the following supported filetypes:

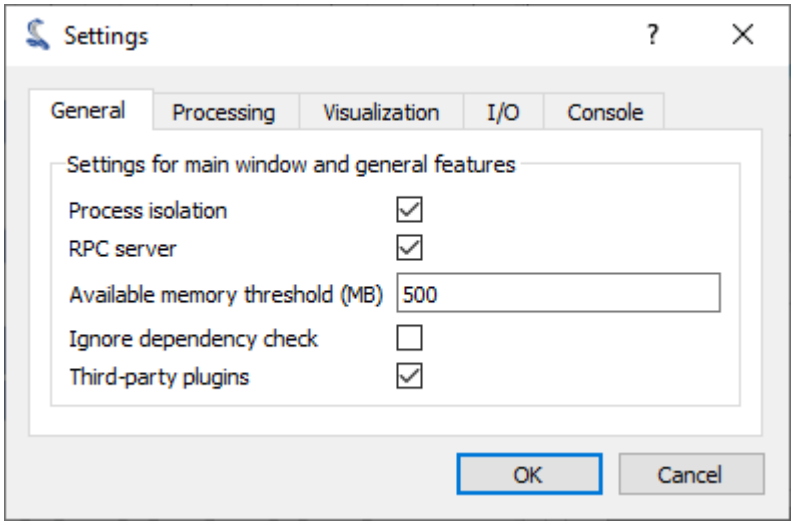
File type	Extensions
Text files	.csv

Open HDF5 file
Import data from a HDF5 file.

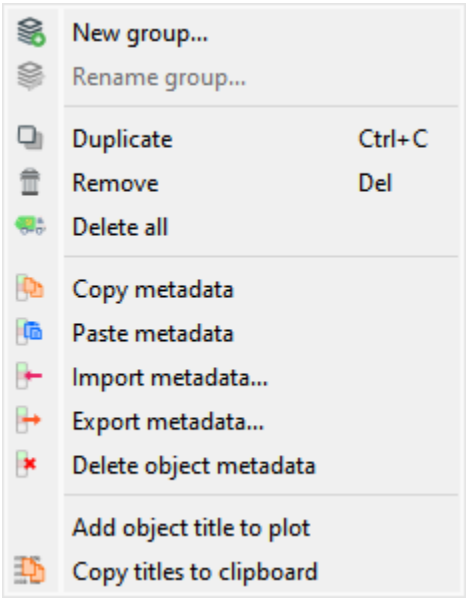
Save to HDF5 file
Export the whole DataLab session (all signals and images) into a HDF5 file.

Browse HDF5 file
Open the *HDF5 Browser* in a new window to browse and import data from HDF5 file.

Settings
Open the the “Settings” dialog box.



3.2 “Edit” menu



Duplicate
Create a new signal which is identical to the currently selected object.

Remove
Remove currently selected signal.

Delete all
Delete all signals.

Copy metadata
Copy metadata from currently selected image into clipboard.

Paste metadata
Paste metadata from clipboard into selected image.

Import metadata into signal
Import metadata from a JSON text file.

Export metadata from signal
Export metadata to a JSON text file.

Delete object metadata
Delete metadata from currently selected signal. Metadata contains additionnal information such as Region of Interest or results of computations

Add object title to plot
Add currently selected signal title to the associated plot.

Copy titles to clipboard
Copy all signal titles to clipboard as a multiline text. This text may be used for reproducing a processing chain, for example.

3.3 "Operation" menu

Sum
Create a new signal which is the sum of all selected signals:

$$y_M = \sum_{k=0}^{M-1} y_k$$

Average

Create a new signal which is the average of all selected signals:

$$y_M = \frac{1}{M}\sum_{k=0}^{M-1} y_k$$

Difference

Create a new signal which is the difference of the **two** selected signals:

$$y_2 = y_1 - y_0$$

Product

Create a new signal which is the product of all selected signals:

$$y_M = \prod_{k=0}^{M-1} y_k$$

Division

Create a new signal which is the division of the **two** selected signals:

$$y_2 = \frac{y_1}{y_0}$$

Absolute value

Create a new signal which is the absolute value of each selected signal:

$$y_k = |y_{k-1}|$$

Real part

Create a new signal which is the real part of each selected signal:

$$y_k = \Re(y_{k-1})$$

Imaginary part

Create a new signal which is the imaginary part of each selected signal:

$$y_k = \Im(y_{k-1})$$

Convert data type

Create a new signal which is the result of converting data type of each selected signal.

Note: Data type conversion relies on `numpy.ndarray.astype()` function with the default parameters (*casting='unsafe'*).

Log10(y)

Create a new signal which is the base 10 logarithm of each selected signal:

$$z_k = \log_{10}(z_{k-1})$$

Peak detection

Create a new signal from semi-automatic peak detection of each selected signal.

ROI extraction

Create a new signal from a user-defined Region of Interest (ROI).

Swap X/Y axes

Create a new signal which is the result of swapping X/Y data.

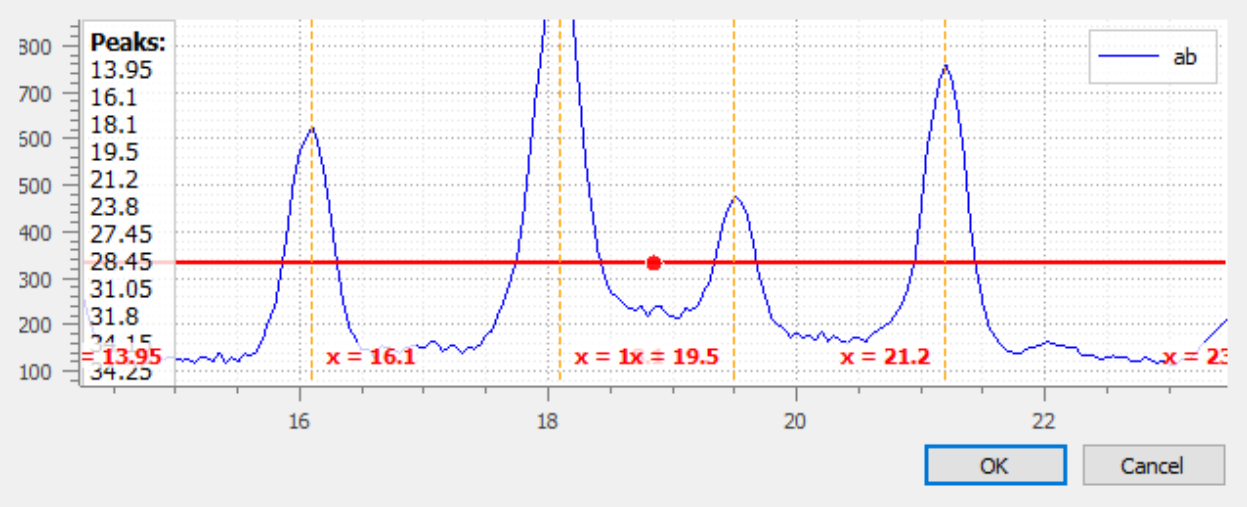


Fig. 2: Peak detection dialog: threshold is adjustable by moving the horizontal marker, peaks are detected automatically (see vertical markers with labels indicating peak position)

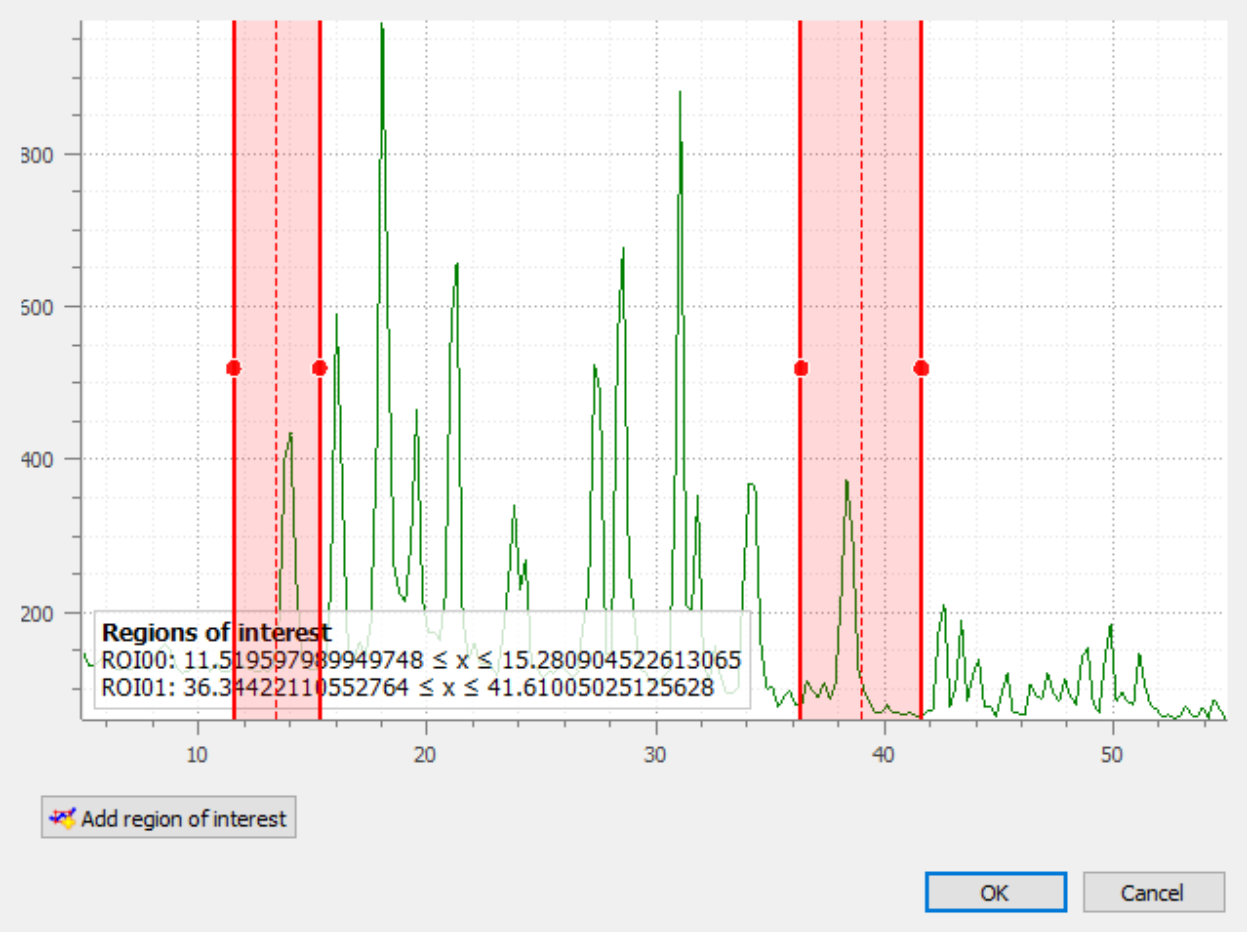


Fig. 3: ROI extraction dialog: the ROI is defined by moving the position and adjusting the width of an horizontal range.

3.4 “Processing” menu

Normalize
Create a new signal which is the normalization of each selected signal by maximum, amplitude, sum or energy:

Parameter	Normalization
Maximum	$y_1 = \frac{y_0}{max(y_0)}$
Amplitude	$y_1 = \frac{y'_0}{max(y'_0)}$ with $y'_0 = y_0 - min(y_0)$
Sum	$y_1 = \frac{y_0}{\sum_{n=0}^{N} y_0[n]}$
Energy	$y_1 = \frac{y_0}{\sum_{n=0}^{N} \lvert y_0[n]\rvert^2}$

Derivative
Create a new signal which is the derivative of each selected signal.

Integral
Create a new signal which is the integral of each selected signal.

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	$y_1 = a.y_0 + b$

Gaussian filter
Compute 1D-Gaussian filter of each selected signal (implementation based on scipy.ndimage.gaussian_filter1d).

Moving average

Compute moving average on M points of each selected signal, without border effect:

$$y_1[i] = \frac{1}{M} \sum_{j=0}^{M-1} y_0[i+j]$$

Moving median
Compute moving median of each selected signal (implementation based on scipy.signal.medfilt).

Wiener filter
Compute Wiener filter of each selected signal (implementation based on scipy.signal.wiener).

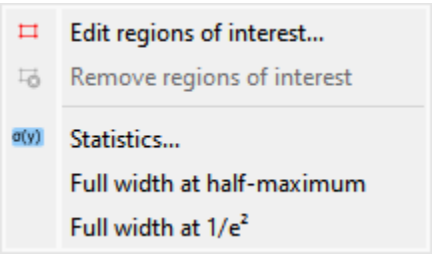
FFT
Create a new signal which is the Fast Fourier Transform (FFT) of each selected signal.

Inverse FFT
Create a new signal which is the inverse FFT of each selected signal.

Lorentzian, Voigt, Polynomial and Multi-Gaussian fit
Open an interactive curve fitting tool in a modal dialog box.

Model	Equation
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	$y = y_0 + A.\frac{Re(exp(-z^2).erfc(-j.z))}{\sqrt{2\pi}.\sigma}$ with $z = \frac{x-x_0-j.\sigma}{\sqrt{2}.\sigma}$
Multi-Gaussian	$y = y_0 + \sum_{i=0}^{K} \frac{A_i}{\sqrt{2\pi}.\sigma_i}.exp(-\frac{1}{2}.(\frac{x-x_{0,i}}{\sigma_i})^2)$

3.5 “Computing” menu



Edit regions of interest
Open a dialog box to setup multiple Region Of Interests (ROI). ROI are stored as metadata, and thus attached to signal.

ROI definition dialog is exactly the same as ROI extraction (see above): the ROI is defined by moving the position and adjusting the width of an horizontal range.

Remove regions of interest
Remove all defined ROI for selected object(s).

Statistics
Compute statistics on selected signal and show a summary table.

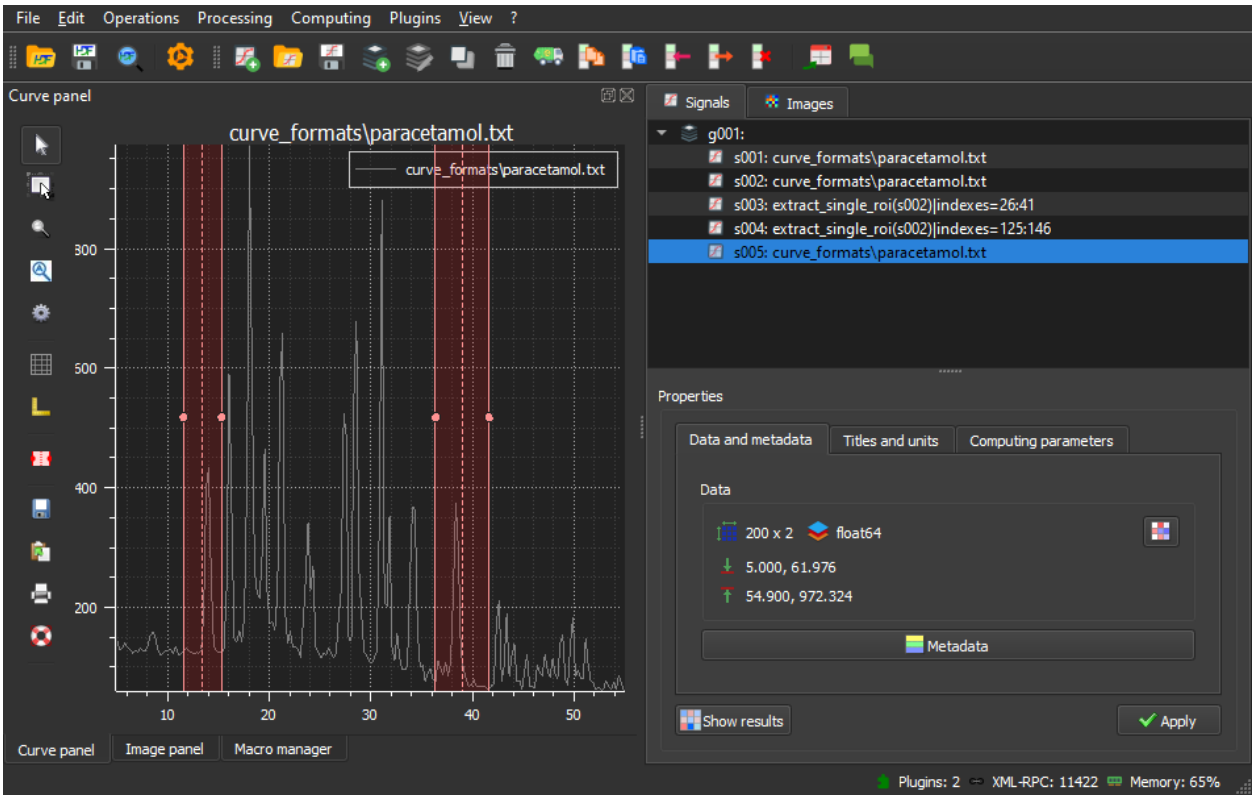


Fig. 4: A signal with an ROI.

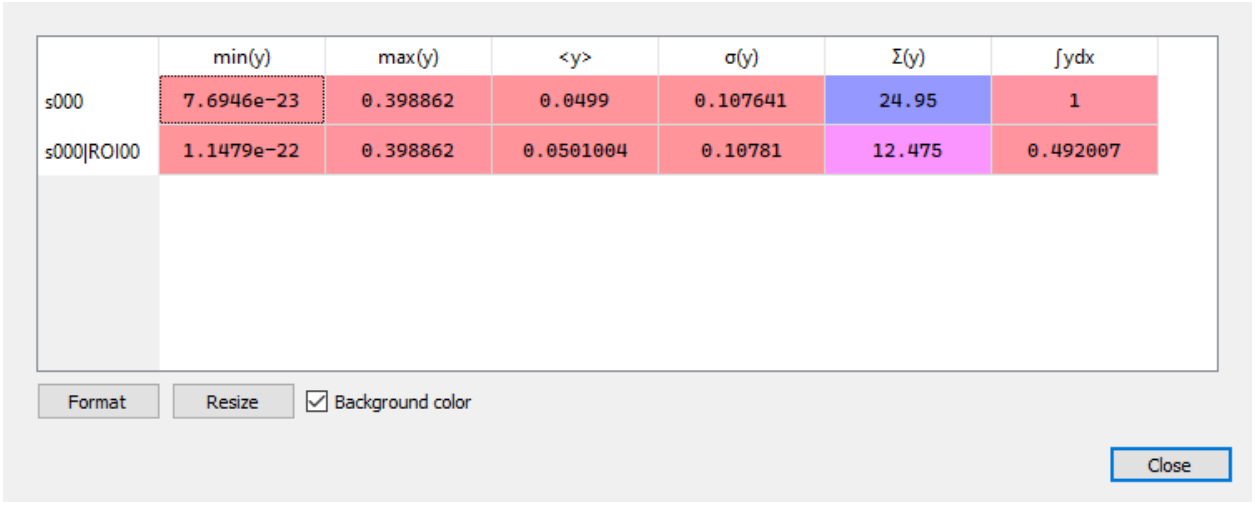


Fig. 5: Example of statistical summary table: each row is associated to an ROI (the first row gives the statistics for the whole data).

Full width at half-maximum

Fit data to a Gaussian, Lorentzian or Voigt model using least-square method. Then, compute the full width at half-maximum value.

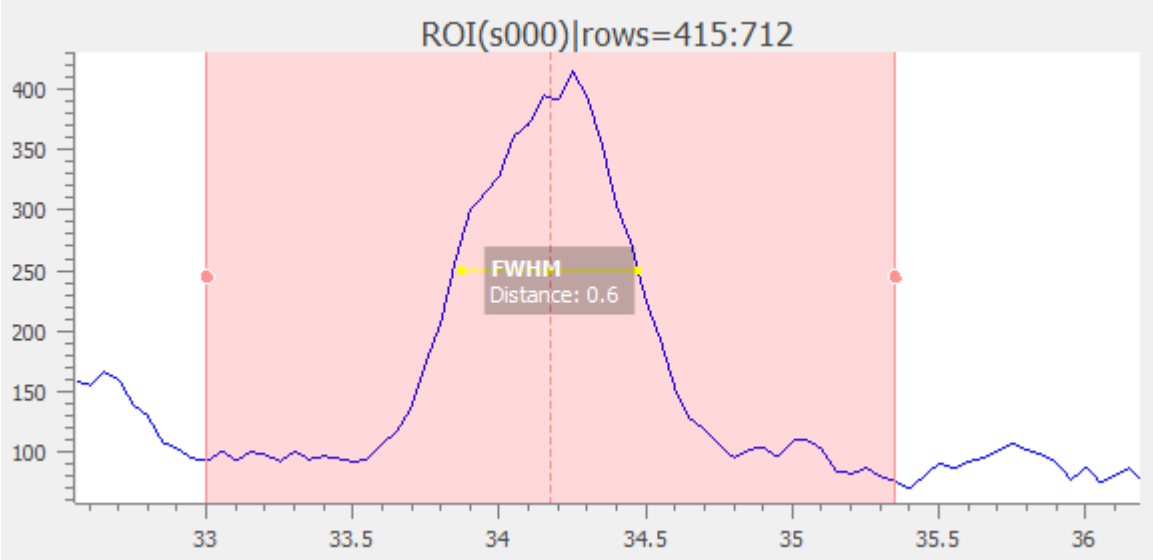


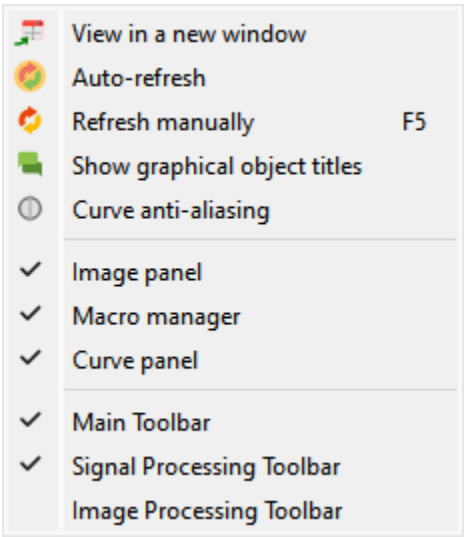
Fig. 6: The computed result is displayed as an annotated segment.

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.

3.6 "View" menu



View in a new window
Open a new window to visualize and the selected signals.

In the separate window, you may visualize your data more comfortably (e.g., by maximizing the window) and you may also annotate the data.

See also:

See *Annotations (Signals)* for more details on annotations.

Show graphical object titles
Show/hide titles of computing results or annotations.

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 clicking the "Refresh manually" button in the toolbar. 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, even if the auto-refresh feature is disabled.

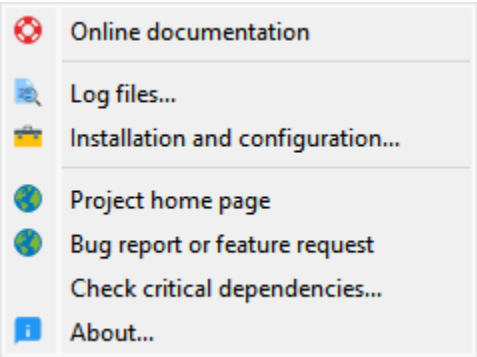
Curve anti-aliasing
Enable/disable anti-aliasing of curves. Anti-aliasing makes the curves look smoother, but it may also make them look less sharp.

Note: Anti-aliasing is enabled by default.

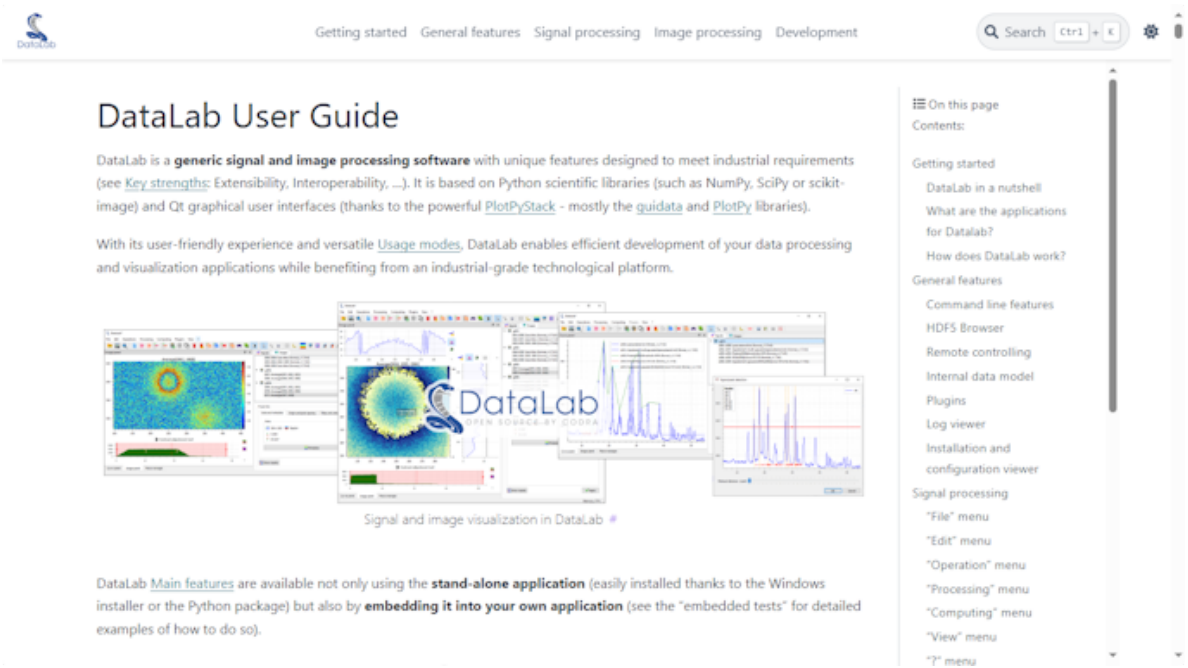
Warning: Anti-aliasing may slow down the visualization, especially when working with large data sets.

Other menu entries
Show/hide panels or toolbars.

3.7 "?" menu



Online or Local documentation
Open the online or local documentation (english only for online version):



Show log files
Open DataLab log viewer

See also:

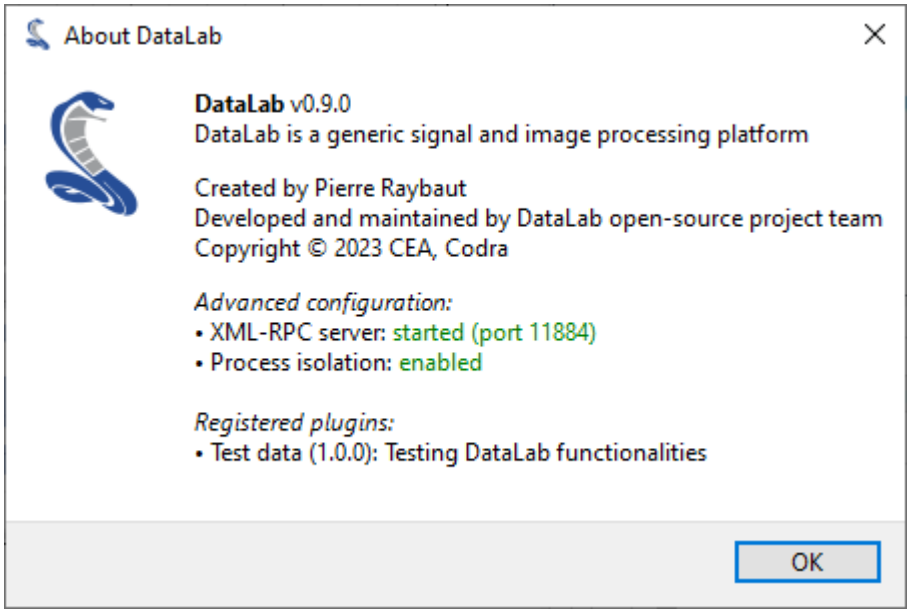
See *Log viewer* for more details on log viewer.

About DataLab installation
Show information regarding your DataLab installation (this is typically needed for submitting a bug report).

See also:

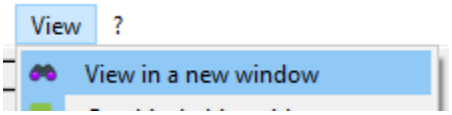
See *Installation and configuration viewer* for more details on this dialog box.

About
Open the "About DataLab" dialog box:



3.8 Annotations (Signals)

DataLab provides an annotation feature for signals (as well as for images).



How to use the feature:

- Create or open a signal in DataLab workspace
- Double-click on the signal or select “View in a new window” in “View” menu
- Add annotations (labels, cursors, rectangles and segments)
- Eventually customize the annotations (right-click, “Parameters”)
- Validate your changes by clicking on “OK” button
- That’s it: your annotations are now attached to the signal and will be saved with your DataLab workspace

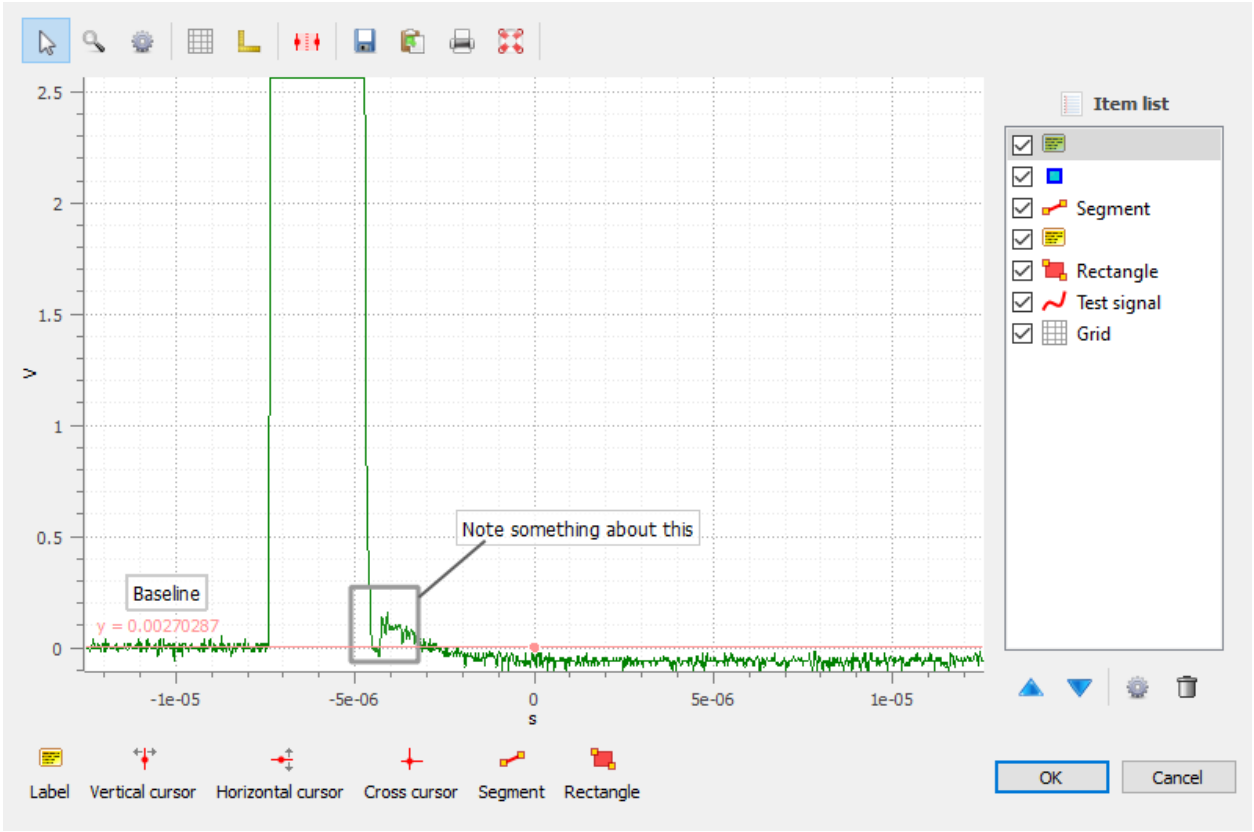
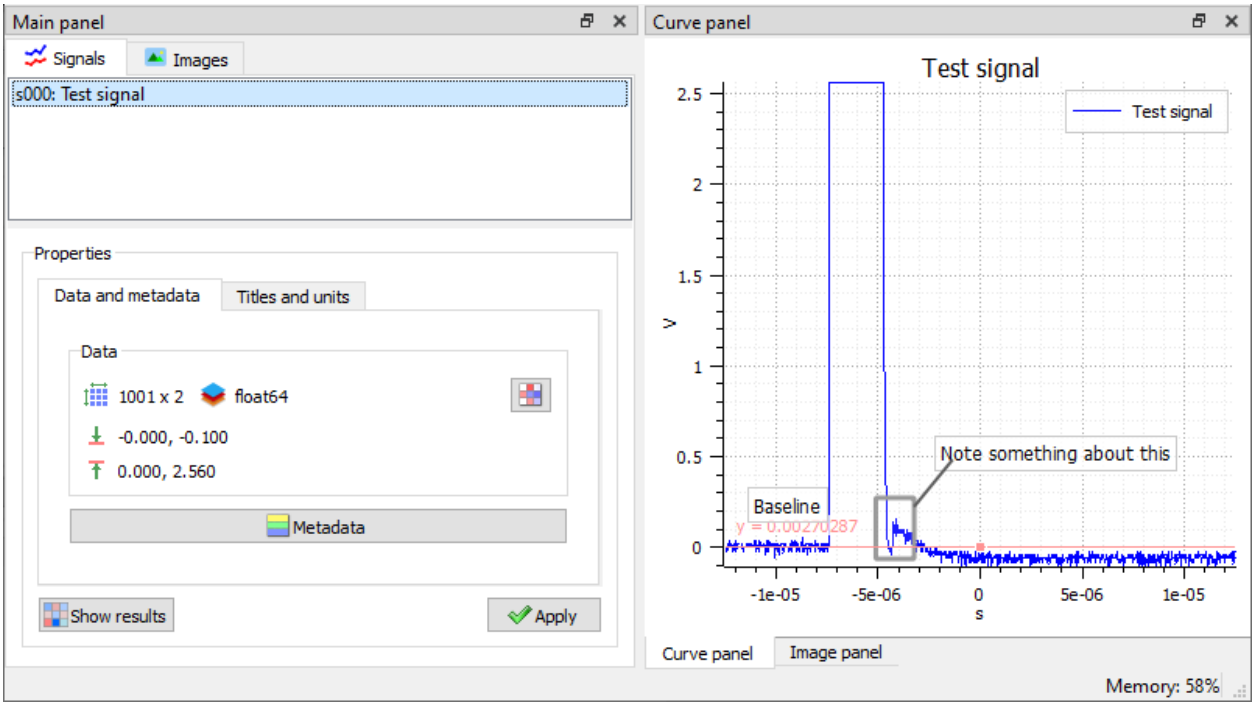


Fig. 7: Example of annotations.

Once the annotations have been added in the separate view (see above), they are part of the object (signal) metadata (see below).

Note: Annotations may be copied from a signal to another by using the “copy/paste metadata” features.


Main panel
Signals
Images
s000: Test signal
Properties
Data and metadata
Titles and units
Data
1001 x 2
float64
-0.000, -0.100
0.000, 2.560
Metadata
Show results
Apply
Curve panel
Test signal
Test signal
2.5
2
1.5
1
0.5
0
V
Note something about this
Baseline
y = 0.00270287
-1e-05
-5e-06
0
5e-06
1e-05
s
Curve panel
Image panel
Memory: 58%

CHAPTER

FOUR

IMAGE PROCESSING

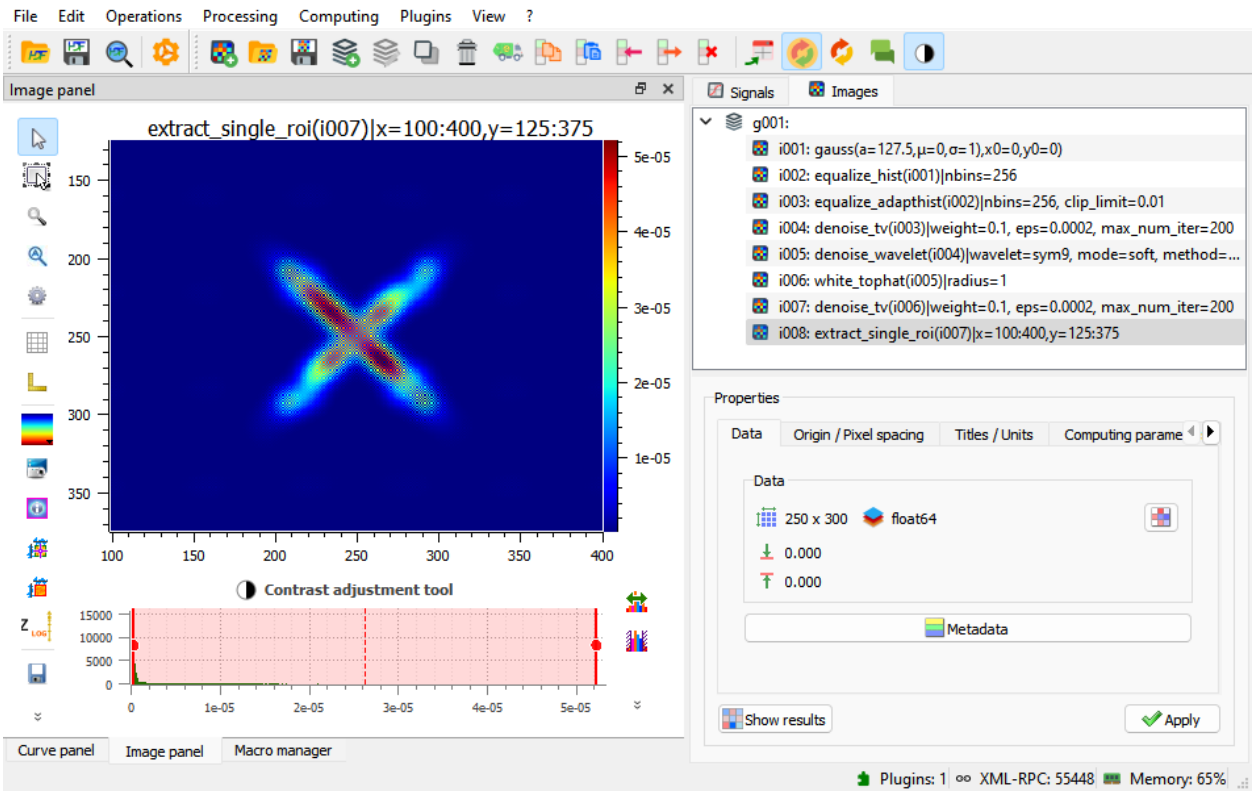
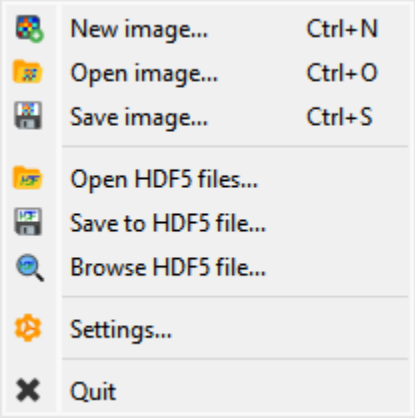


Fig. 1: DataLab main window: Image processing view

4.1 “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	$z = A.exp(-\frac{(\sqrt{(x-x0)^2+(y-y0)^2}-\mu)^2}{2\sigma^2})$

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
Text files	.txt, .csv, .asc
Andor SIF files	.sif
SPIRICON files	.scor-data
FXD files	.fxd
Bitmap images	.bmp

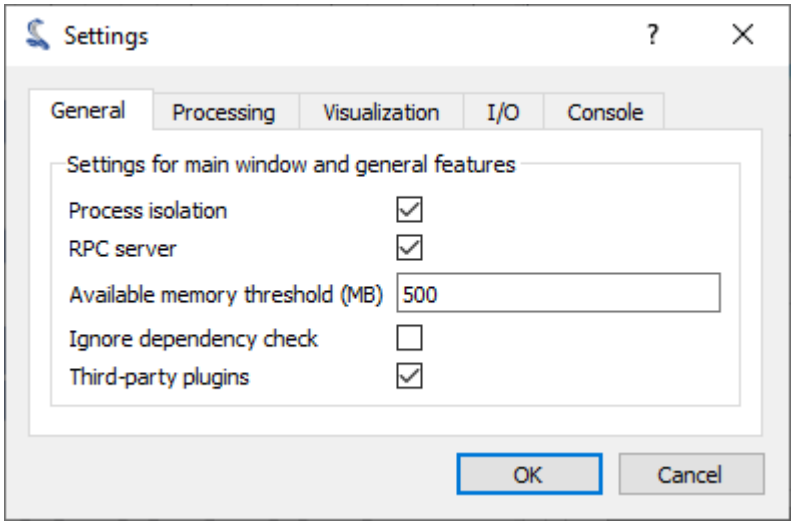
Save image
Save current image (see “Open image” supported filetypes).

Open HDF5 file
Import data from a HDF5 file.

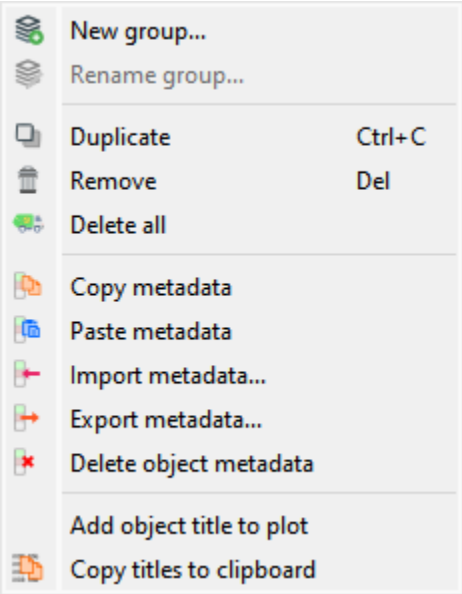
Save to HDF5 file
Export the whole DataLab session (all signals and images) into a HDF5 file.

Browse HDF5 file
Open the *HDF5 Browser* in a new window to browse and import data from HDF5 file.

Settings
Open the the “Settings” dialog box.



4.2 “Edit” menu



Duplicate
Create a new image which is identical to the currently selected object.

Remove
Remove currently selected image.

Delete all
Delete all images.

Copy metadata
Copy metadata from currently selected image into clipboard.

Paste metadata
Paste metadata from clipboard into selected image.

Import metadata into image
Import metadata from a JSON text file.

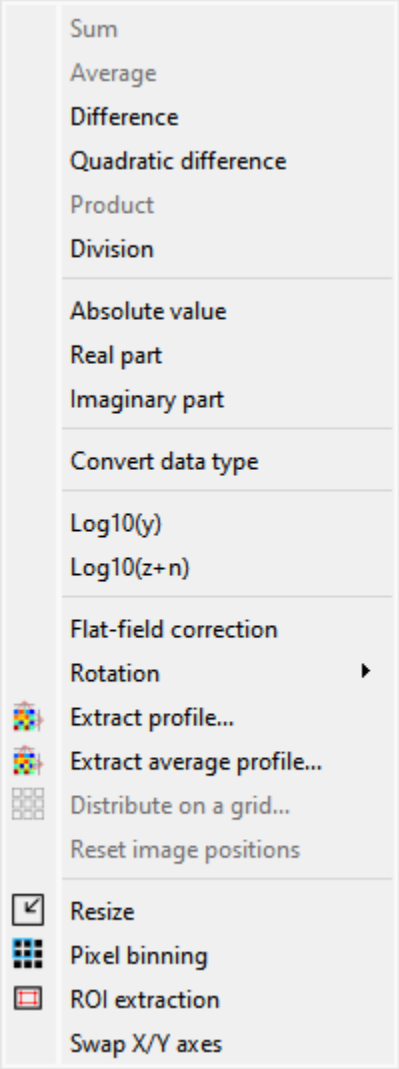
Export metadata from image
Export metadata to a JSON text file.

Delete object metadata
Delete metadata from currently selected image. Metadata contains additionnal information such as Region of Interest or results of computations

Add object title to plot
Add currently selected image title to the associated plot.

Copy titles to clipboard
Copy all image titles to clipboard as a multiline text. This text may be used for reproducing a processing chain, for example.

4.3 “Operation” menu



Sum
: Create a new image which is the sum of all selected images:

$$z_M = \sum_{k=0}^{M-1} z_k$$

Average
: Create a new image which is the average of all selected images:

$$z_M = \frac{1}{M} \sum_{k=0}^{M-1} z_k$$

Difference
: Create a new image which is the difference of the **two** selected images:

$$z_2 = z_1 - z_0$$

Quadratic difference
Create a new image which is the quadratic difference of the **two** selected images:

$$z_2 = \frac{z_1 - z_0}{\sqrt{2}}$$

Product
Create a new image which is the product of all selected images:

$$z_M = \prod_{k=0}^{M-1} z_k$$

Division
Create a new image which is the division of the **two** selected images:

$$z_2 = \frac{z_1}{z_0}$$

Absolute value
Create a new image which is the absolute value of each selected image:

$$z_k = |z_{k-1}|$$

Real part
Create a new image which is the real part of each selected image:

$$z_k = \Re(z_{k-1})$$

Imaginary part
Create a new image which is the imaginary part of each selected image:

$$z_k = \Im(z_{k-1})$$

Convert data type
Create a new image which is the result of converting data type of each selected image.

Note: Data type conversion relies on `numpy.ndarray.astype()` function with the default parameters (*casting='unsafe'*).

Log10(z)
Create a new image which is the base 10 logarithm of each selected image:

$$z_k = \log_{10}(z_{k-1})$$

Log10(z+n)
Create a new image which is the Log10(z+n) of each selected image (avoid Log10(0) on image background):

$$z_k = \log_{10}(z_{k-1} + n)$$

Flat-field correction

Create a new image which is flat-field correction of the **two** selected images:

$$z_1 = \begin{cases} \frac{z_0}{z_f}.\overline{z_f} & \text{if } z_0 > z_{threshold} \\ z_0 & \text{otherwise} \end{cases},$$

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:

$$\overline{z_f} = \frac{1}{N_{row}.N_{col}}. \sum_{i=0}^{N_{row}} \sum_{j=0}^{N_{col}} z_f(i,j)$$

Note: Raw image and flat field image are supposedly already corrected by performing a dark frame subtraction.

Rotation

Create a new image which is the result of rotating (90°, 270° or arbitrary angle) or flipping (horizontally or vertically) data.

Extract profile

Extract an horizontal or vertical profile from each selected image, and create new signals from these profiles.

Extract average profile

Extract an horizontal or vertical profile averaged over a rectangular area, from each selected image, and create new signals from these profiles.

Distribute on a grid

Distribute selected images on a regular grid.

Reset image positions

Reset selected image positions to first image (x0, y0) coordinates.

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.

Swap X/Y axes

Create a new image which is the result of swapping X/Y data.

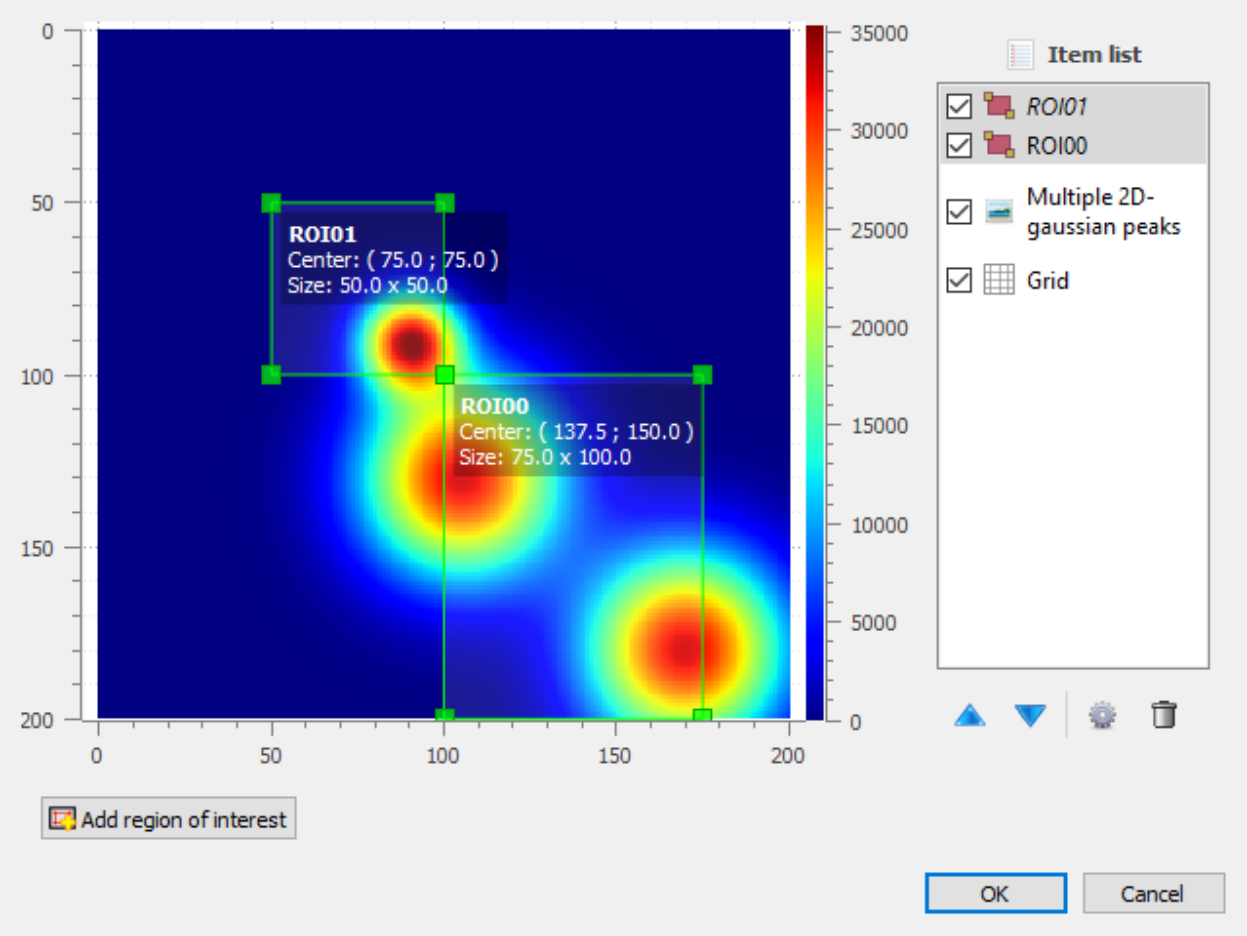


Fig. 2: ROI extraction dialog: the ROI is defined by moving the position and adjusting the size of a rectangle shape.

4.4 “Processing” menu

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	$z_1 = a.z_0 + b$

Thresholding
Apply the thresholding to each selected image.

Clipping
Apply the clipping to each selected image.

Moving average
Compute moving average of each selected image (implementation based on scipy.ndimage.uniform_filter).

Moving median
Compute moving median of each selected image (implementation based on scipy.signal.medfilt).

Wiener filter
Compute Wiener filter of each selected image (implementation based on scipy.signal.wiener).

FFT
Create a new image which is the Fast Fourier Transform (FFT) of each selected image.

Inverse FFT
Create a new image which is the inverse FFT of each selected image.

Butterworth filter
Perform Butterworth filter on an image (implementation based on skimage.filters.butterworth)

Exposure

Gamma correction
Apply gamma correction to each selected image (implementation based on skimage.exposure.adjust_gamma)

Logarithmic correction
Apply logarithmic correction to each selected image (implementation based on skimage.exposure.adjust_log)

Sigmoid correction
Apply sigmoid correction to each selected image (implementation based on skimage.exposure.adjust_sigmoid)

Histogram equalization
Equalize image histogram levels (implementation based on skimage.exposure.equalize_hist)

Adaptive histogram equalization
Equalize image histogram levels using Contrast Limited Adaptive Histogram Equalization (CLAHE) algorithm (implementation based on skimage.exposure.equalize_adapthist)

Intensity rescaling
Stretch or shrink image intensity levels (implementation based on skimage.exposure.rescale_intensity)

Restoration

Total variation denoising
Denoise image using Total Variation algorithm (implementation based on skimage.restoration.denoise_tv_chambolle)

Bilateral filter denoising
Denoise image using bilateral filter (implementation based on skimage.restoration.denoise_bilateral)

Wavelet denoising
Perform wavelet denoising on image (implementation based on skimage.restoration.denoise_wavelet)

White Top-Hat denoising
Denoise image by subtracting its white top hat transform (using a disk footprint)

All denoising methods
Perform all denoising methods on image. Combined with the "distribute on a grid" option, this allows to compare the different denoising methods on the same image.

Morphology

White Top-Hat (disk)
Perform white top hat transform of an image, using a disk footprint (implementation based on skimage.morphology.white_tophat)

Black Top-Hat (disk)
Perform black top hat transform of an image, using a disk footprint (implementation based on skimage.morphology.black_tophat)

Erosion (disk)
Perform morphological erosion on an image, using a disk footprint (implementation based on skimage.morphology.erosion)

Dilation (disk)
Perform morphological dilation on an image, using a disk footprint (implementation based on skimage.morphology.dilation)

Opening (disk)
Perform morphological opening on an image, using a disk footprint (implementation based on skimage.morphology.opening)

Closing (disk)
Perform morphological closing on an image, using a disk footprint (implementation based on skimage.morphology.closing)

All morphological operations
Perform all morphological operations on an image, using a disk footprint. Combined with the "distribute on a grid" option, this allows to compare the different morphological operations on the same image.

Edges

Roberts filter
Perform edge filtering on an image, using the Roberts algorithm (implementation based on skimage.filters.roberts)

Prewitt filter
Perform edge filtering on an image, using the Prewitt algorithm (implementation based on skimage.filters.prewitt)

Prewitt filter (horizontal)
Find the horizontal edges of an image, using the Prewitt algorithm (implementation based on skimage.filters.prewitt_h)

Prewitt filter (vertical)
Find the vertical edges of an image, using the Prewitt algorithm (implementation based on skimage.filters.prewitt_v)

Sobel filter
Perform edge filtering on an image, using the Sobel algorithm (implementation based on skimage.filters.sobel)

Sobel filter (horizontal)
Find the horizontal edges of an image, using the Sobel algorithm (implementation based on skimage.filters.sobel_h)

Sobel filter (vertical)
Find the vertical edges of an image, using the Sobel algorithm (implementation based on skimage.filters.sobel_v)

Scharr filter
Perform edge filtering on an image, using the Scharr algorithm (implementation based on skimage.filters.scharr)

Scharr filter (horizontal)
Find the horizontal edges of an image, using the Scharr algorithm (implementation based on skimage.filters.scharr_h)

Scharr filter (vertical)
Find the vertical edges of an image, using the Scharr algorithm (implementation based on skimage.filters.scharr_v)

Farid filter
Perform edge filtering on an image, using the Farid algorithm (implementation based on skimage.filters.farid)

Farid filter (horizontal)
Find the horizontal edges of an image, using the Farid algorithm (implementation based on skimage.filters.farid_h)

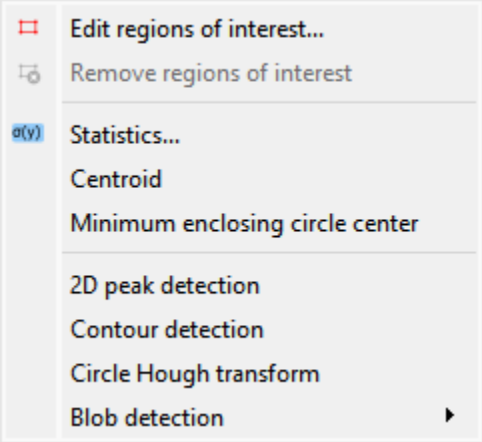
Farid filter (vertical)
Find the vertical edges of an image, using the Farid algorithm (implementation based on skimage.filters.farid_v)

Laplace filter
Perform edge filtering on an image, using the Laplace algorithm (implementation based on skimage.filters.laplace)

All edges filters
Perform all edge filtering algorithms (see above) on an image. Combined with the "distribute on a grid" option, this allows to compare the different edge filters on the same image.

Canny filter
Perform edge filtering on an image, using the Canny algorithm (implementation based on skimage.feature.canny)

4.5 "Computing" menu



Edit regions of interest
Open a dialog box to setup multiple Region Of Interests (ROI). ROI are stored as metadata, and thus attached to image.

ROI definition dialog is exactly the same as ROI extraction (see above).

Remove regions of interest
Remove all defined ROI for selected object(s).

Statistics
Compute statistics on selected image and show a summary table.

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.

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.

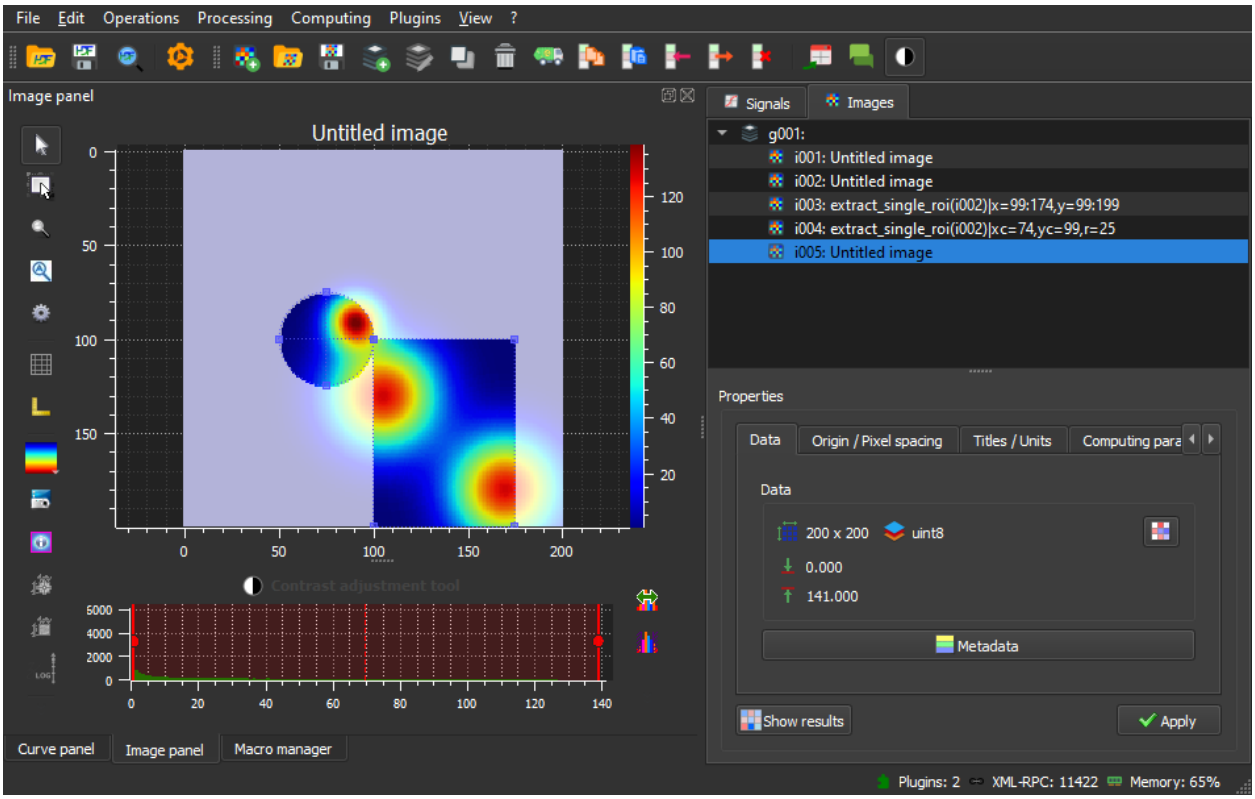


Fig. 3: An image with ROI.

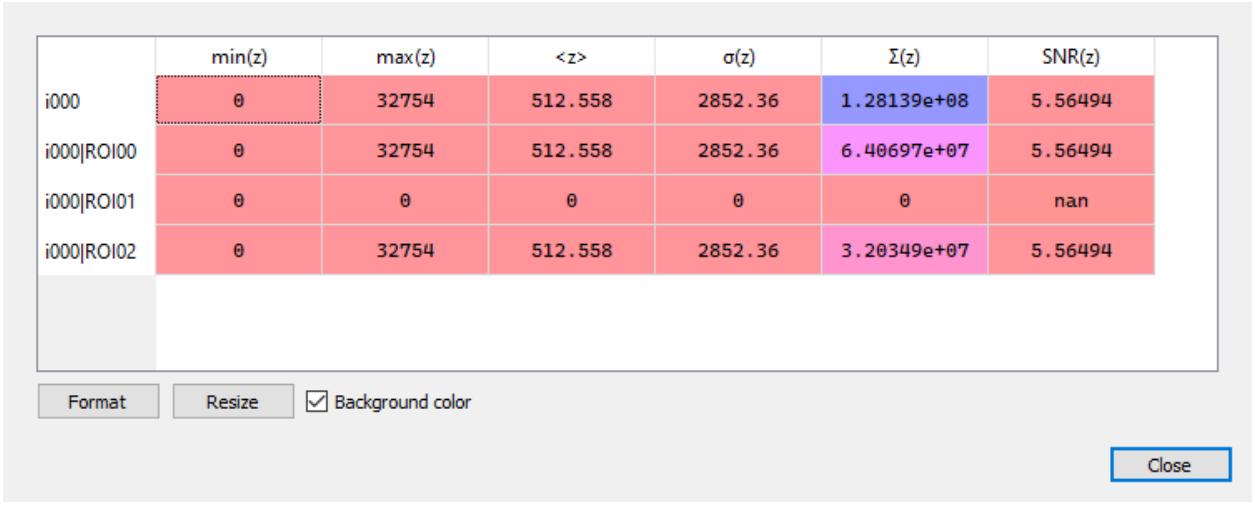


Fig. 4: Example of statistical summary table: each row is associated to an ROI (the first row gives the statistics for the whole data).

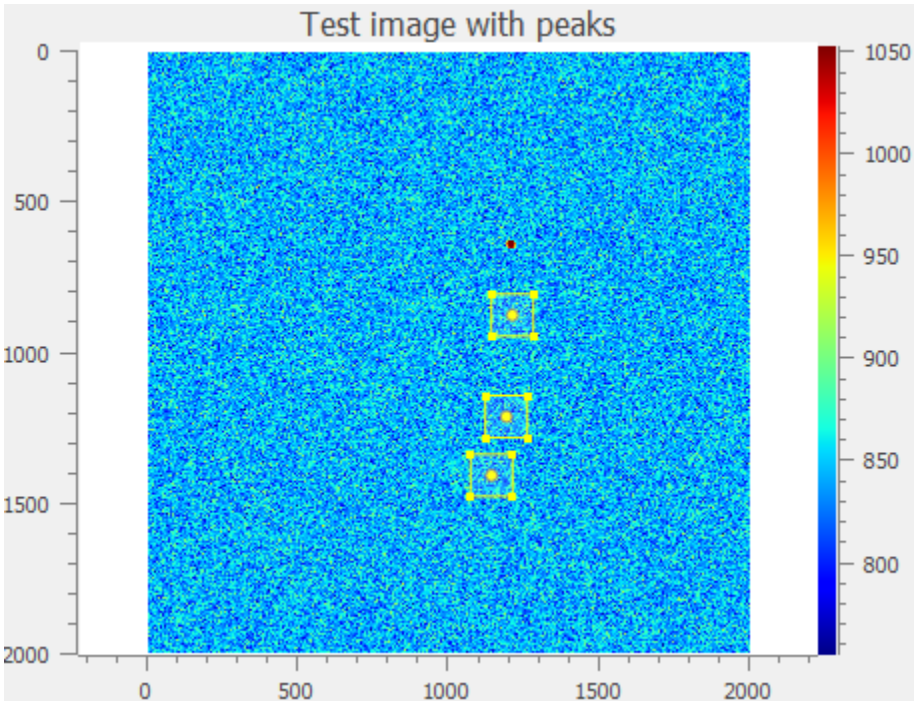


Fig. 5: Example of 2D peak detection.

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.

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.

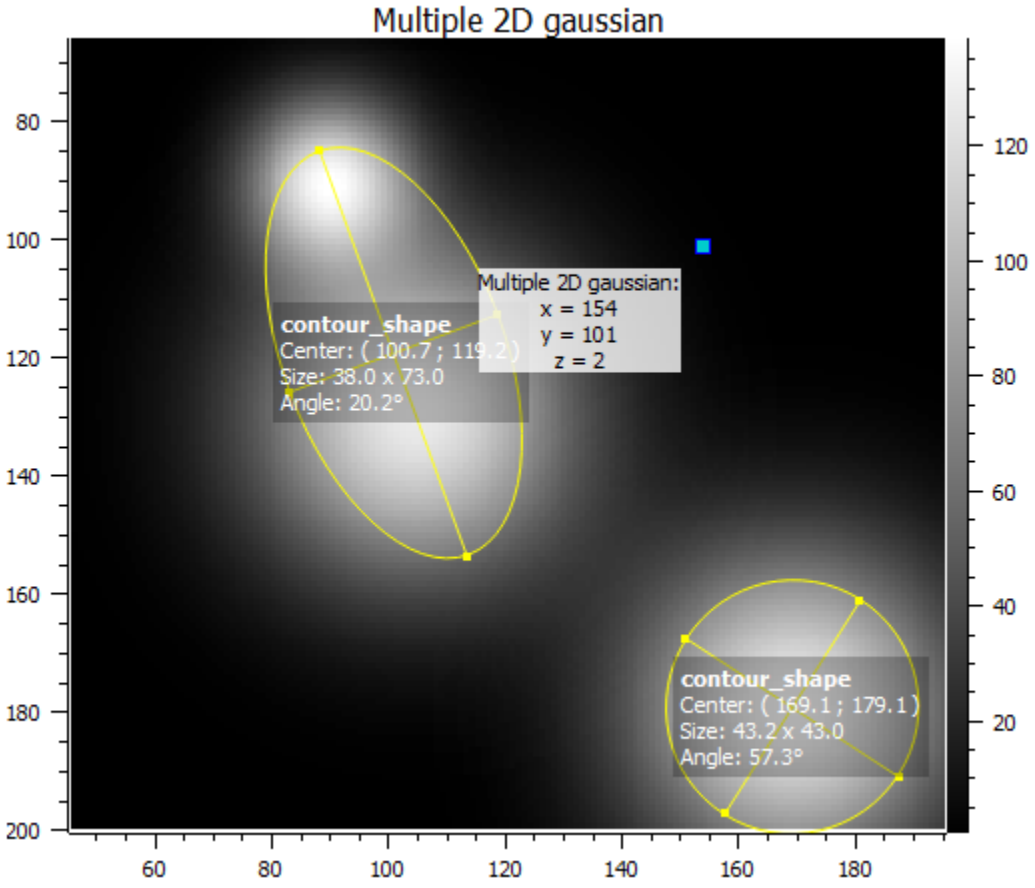
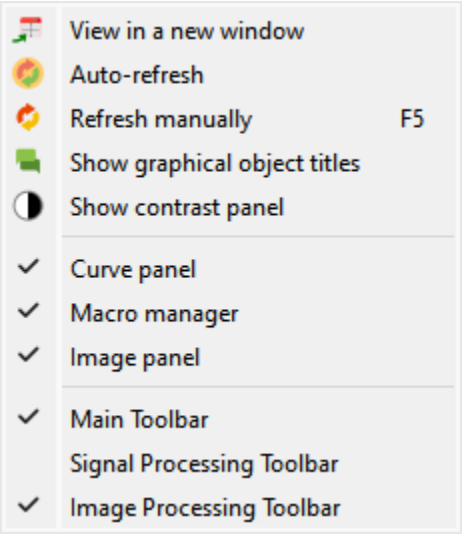


Fig. 6: Example of contour detection.

4.6 “View” menu



View in a new window
Open a new window to visualize and the selected images.

In the separate window, you may visualize your data more comfortably (e.g., by maximizing the window) and you may also annotate the data.

See also:

See *Annotations (Images)* for more details on annotations.

Show graphical object titles
Show/hide titles of computing results or annotations.

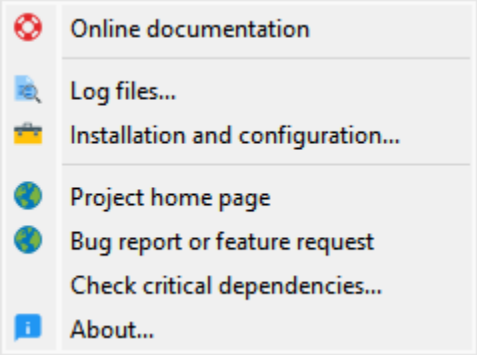
Auto-refresh
Automatically refresh the visualization when the data changes. When enabled (default), the plot view is automatically refreshed when the data changes. When disabled, the plot view is not refreshed until you manually refresh it by clicking the “Refresh manually” button in the toolbar. Even 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, even if the auto-refresh feature is disabled.

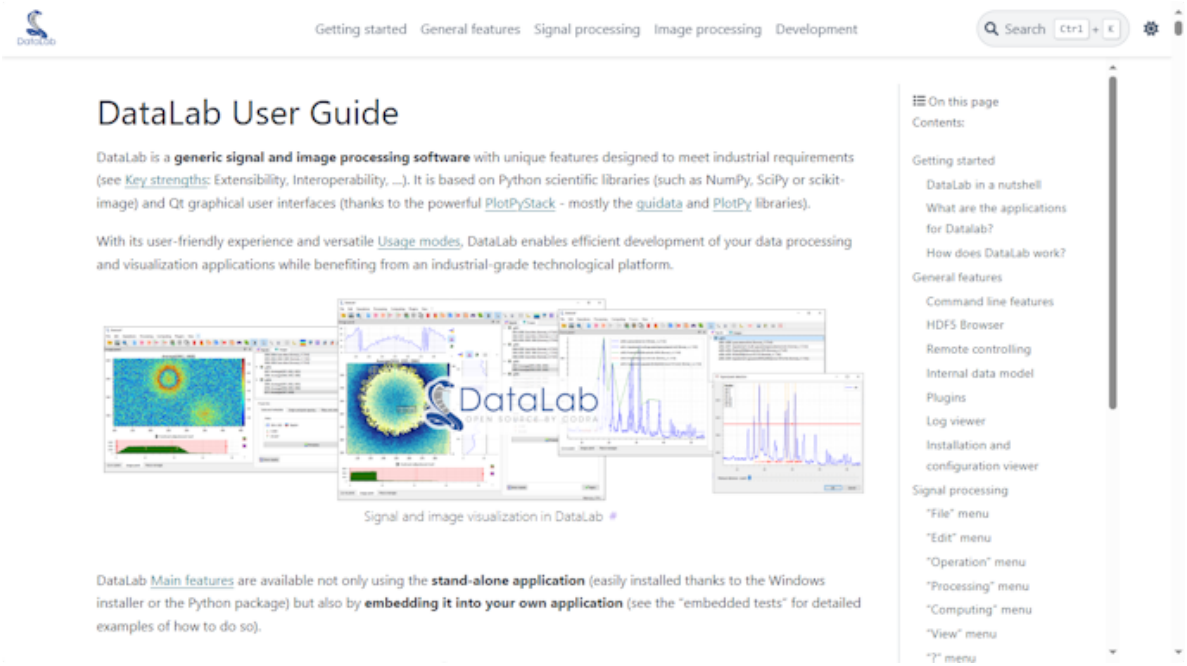
Show contrast panel
Show/hide contrast adjustment panel.

Other menu entries
Show/hide panels or toolbars.

4.7 “?” menu



Online or Local documentation
Open the online or local documentation (english only for online version):



Show log files
Open DataLab log viewer

See also:

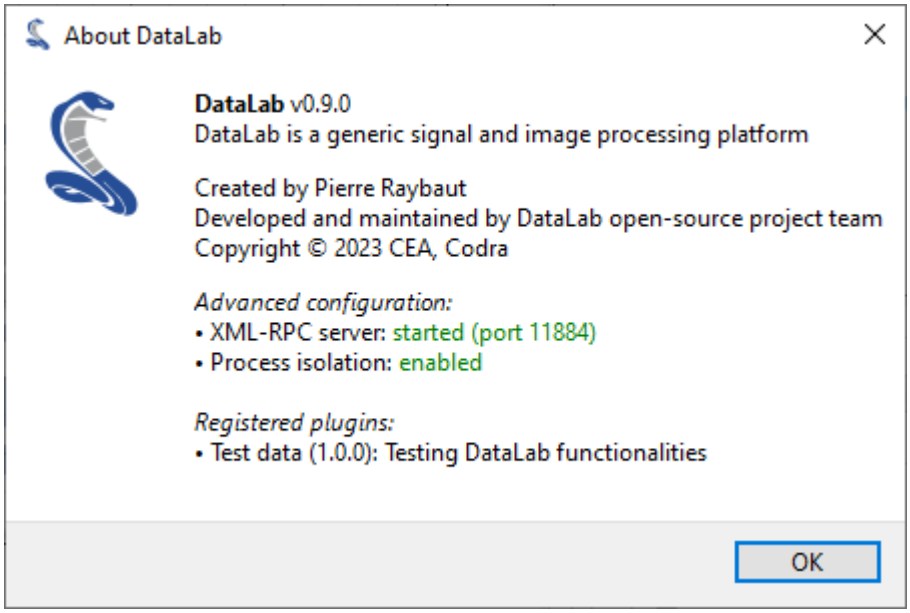
See *Log viewer* for more details on log viewer.

About DataLab installation
Show information regarding your DataLab installation (this is typically needed for submitting a bug report).

See also:

See *Installation and configuration viewer* for more details on this dialog box.

About
Open the “About DataLab” dialog box:



4.8 2D Peak Detection

DataLab provides a "2D Peak Detection" feature which is based on a minimum-maximum filter algorithm.

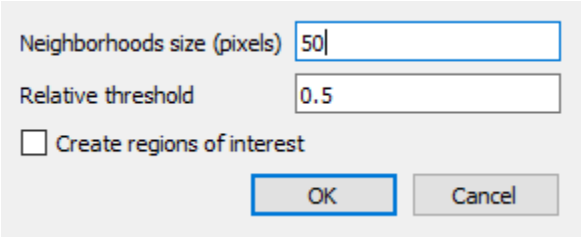


Fig. 7: 2D peak detection parameters.

How to use the feature:

- Create or open an image in DataLab workspace
- Select "2d peak detection" in "Computing" menu
- Enter parameters "Neighborhoods size" and "Relative threhold"
- 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:

Fig. 8: 2d peak detection results (see test "peak2d_app.py")

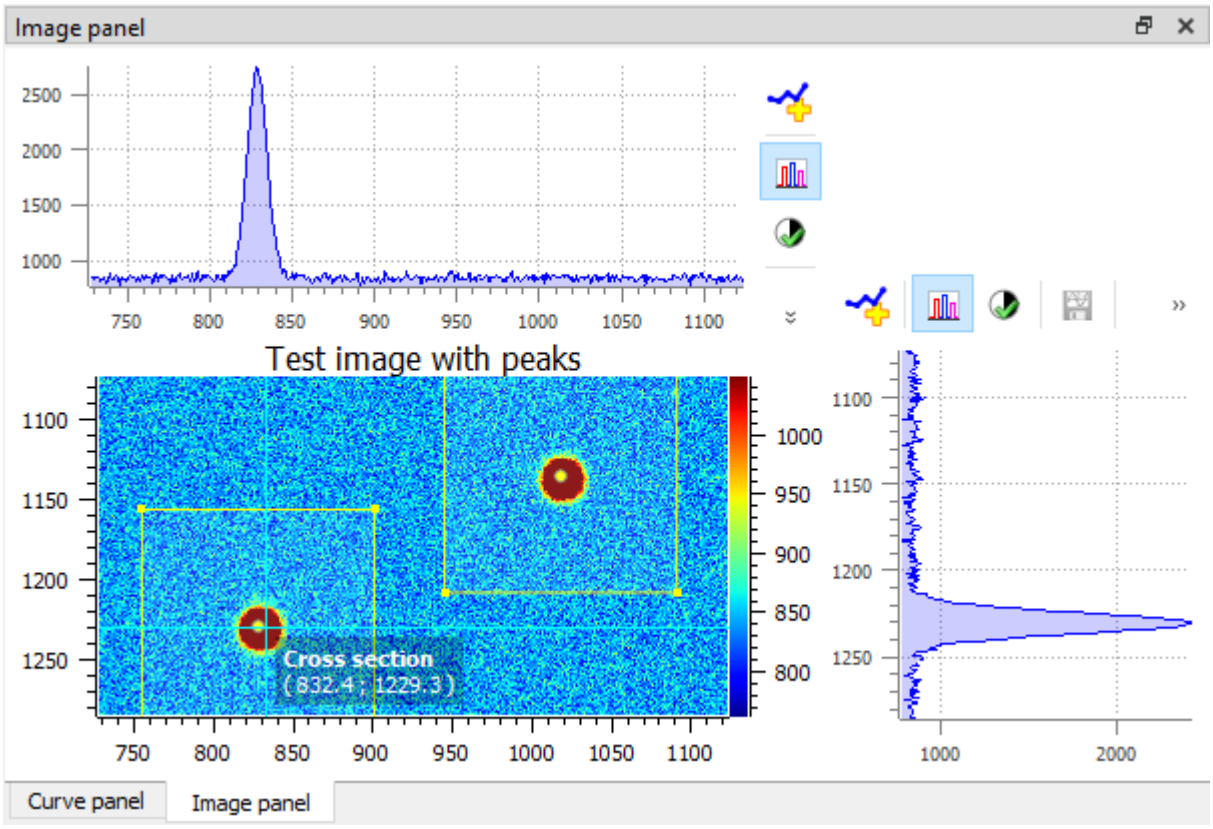


Fig. 9: Example of 2D peak detection.

- 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

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:

```python
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 (numpy.ndarray): Input data
        size (int | None): Neighborhood size (default: None)
        level (float | None): Relative level (default: 0.5)

    Returns:
        np.ndarray: Coordinates of peaks
    """
    if size is None:
        size = max(min(data.shape) // 40, 50)
    data_max = spf.maximum_filter(data, size)
    data_min = spf.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
```

(continues on next page)

(continued from previous page)

```
        dist = distance_matrix(coords)
        for index in reversed(np.unique(np.where((dist < size) & (dist >␣
↪0))[1])):
            coords.pop(index)
    return np.array(coords)
```

4.9 Contour Detection

DataLab provides a "Contour Detection" feature which is based on the marching cubes algorithm.

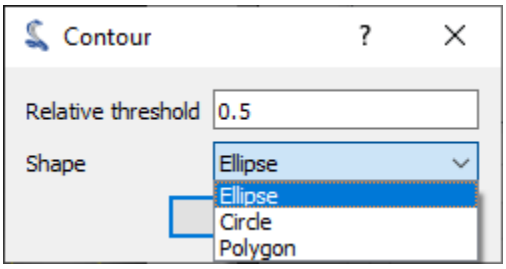


Fig. 10: Contour detection parameters.

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 "Computing" menu
- Enter parameter "Shape" ("Ellipse", "Circle" or "Polygon")

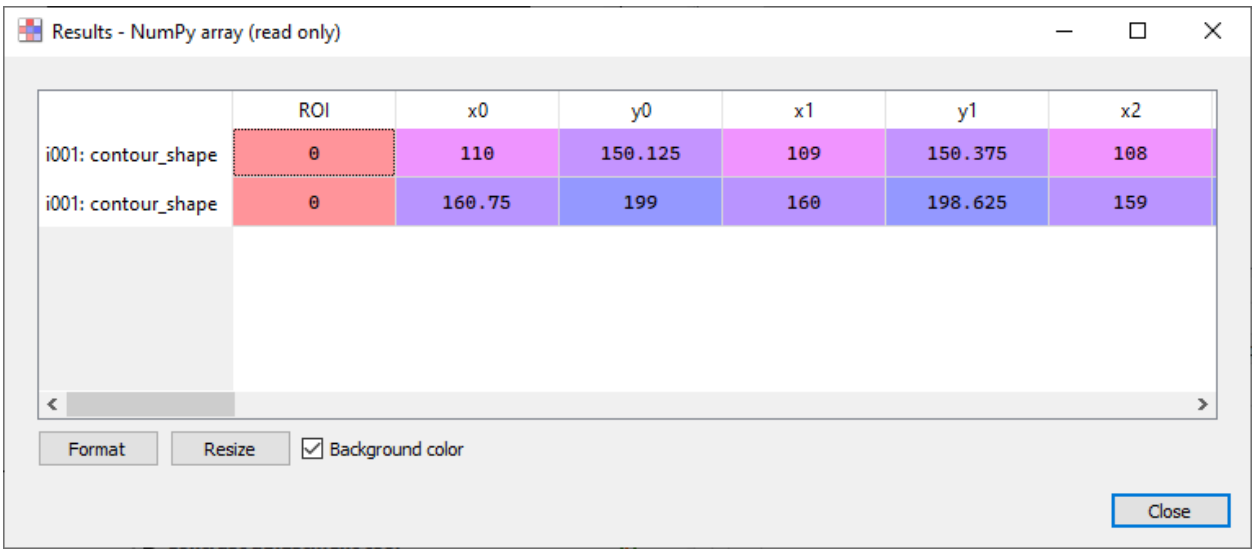


Fig. 11: Contour detection results (see test "contour_app.py")

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)

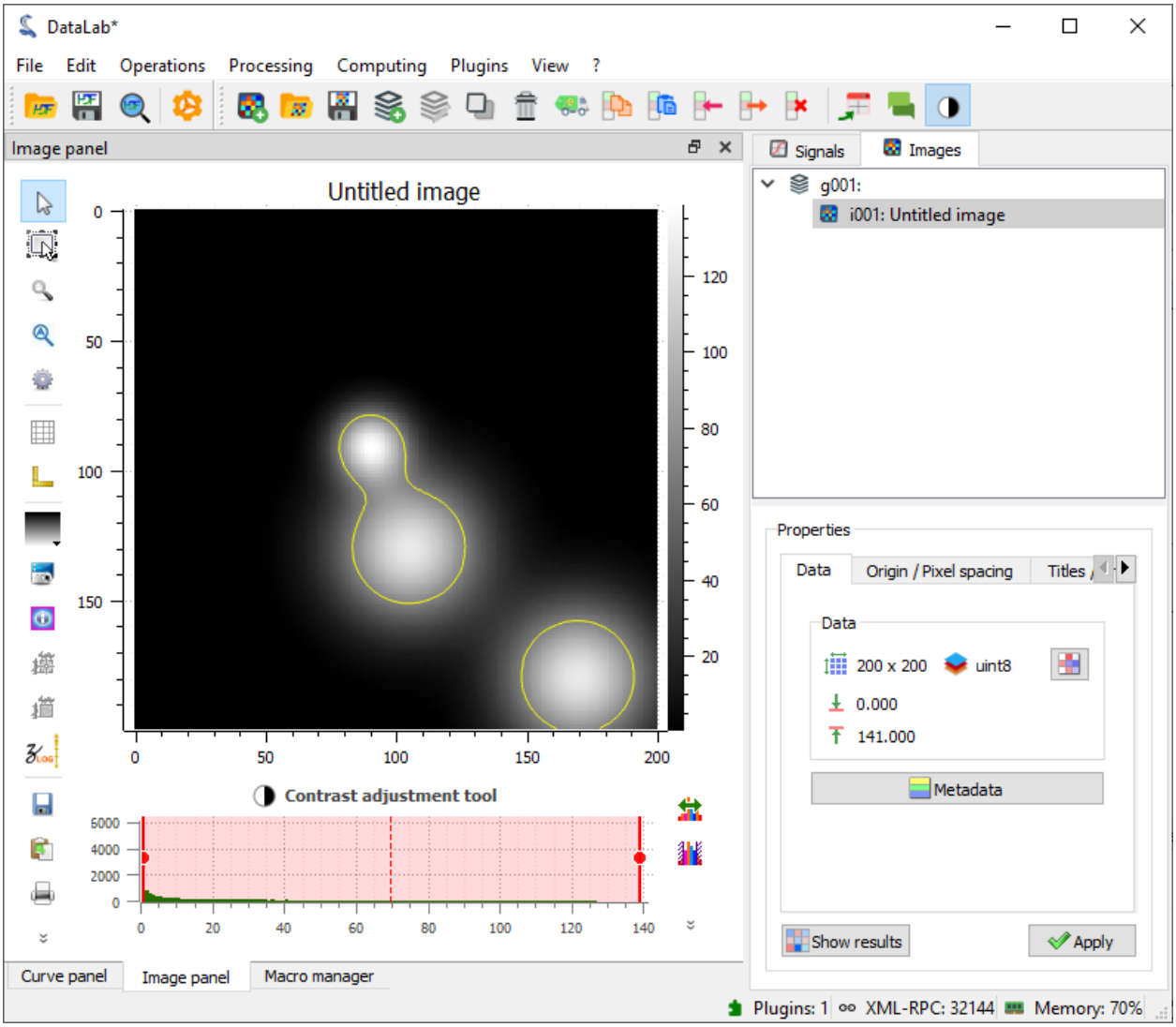


Fig. 12: Example of contour detection.

The contour detection algorithm works in the following way:

- First, iso-valued contours are computed (implementation based on skimage.measure.find_contours.find_contours)
- 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, shape: str = "ellipse", level: float = 0.5
) -> np.ndarray:
```

(continues on next page)

(continued from previous page)

```
    """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. Valid values: 'circle', 'ellipse', 'polygon'.
         (default: '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:
        if shape == "circle":
            model = measure.CircleModel()
            if model.estimate(contour):
                yc, xc, r = model.params
                if r <= 1.0:
                    continue
                coords.append([xc - r, yc, xc + r, yc])
        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
                dxa, dya = a * np.cos(theta), a * np.sin(theta)
                dxb, dyb = b * np.sin(theta), b * np.cos(theta)
                x1, y1, x2, y2 = xc - dxa, yc - dya, xc + dxa, yc + dya
                x3, y3, x4, y4 = xc - dxb, yc - dyb, xc + dxb, yc + dyb
                coords.append([x1, y1, x2, y2, x3, y3, x4, y4])
        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}")
    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):
```

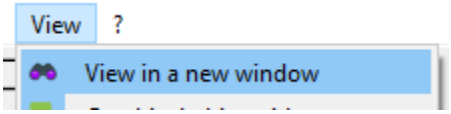
(continues on next page)

(continued from previous page)

```
            arr[i_row, : coord.shape[0]] = coord
        return arr
    return np.array(coords)
```

4.10 Annotations (Images)

DataLab provides an annotation feature for images (as well as for signals).



How to use the feature:

- Create or open an image in DataLab workspace
- Double-click on the image or select “View in a new window” in “View” menu
- Add annotations (labels, rectangles, circles, etc.)
- Eventually customize the annotations (right-click, “Parameters”)
- Validate your changes by clicking on “OK” button
- That’s it: your annotations are now attached to the image and will be saved with your DataLab workspace

Once the annotations have been added in the separate view (see above), they are part of the object (image) metadata (see below).

Note: Annotations may be copied from an image to another by using the “copy/paste metadata” features.

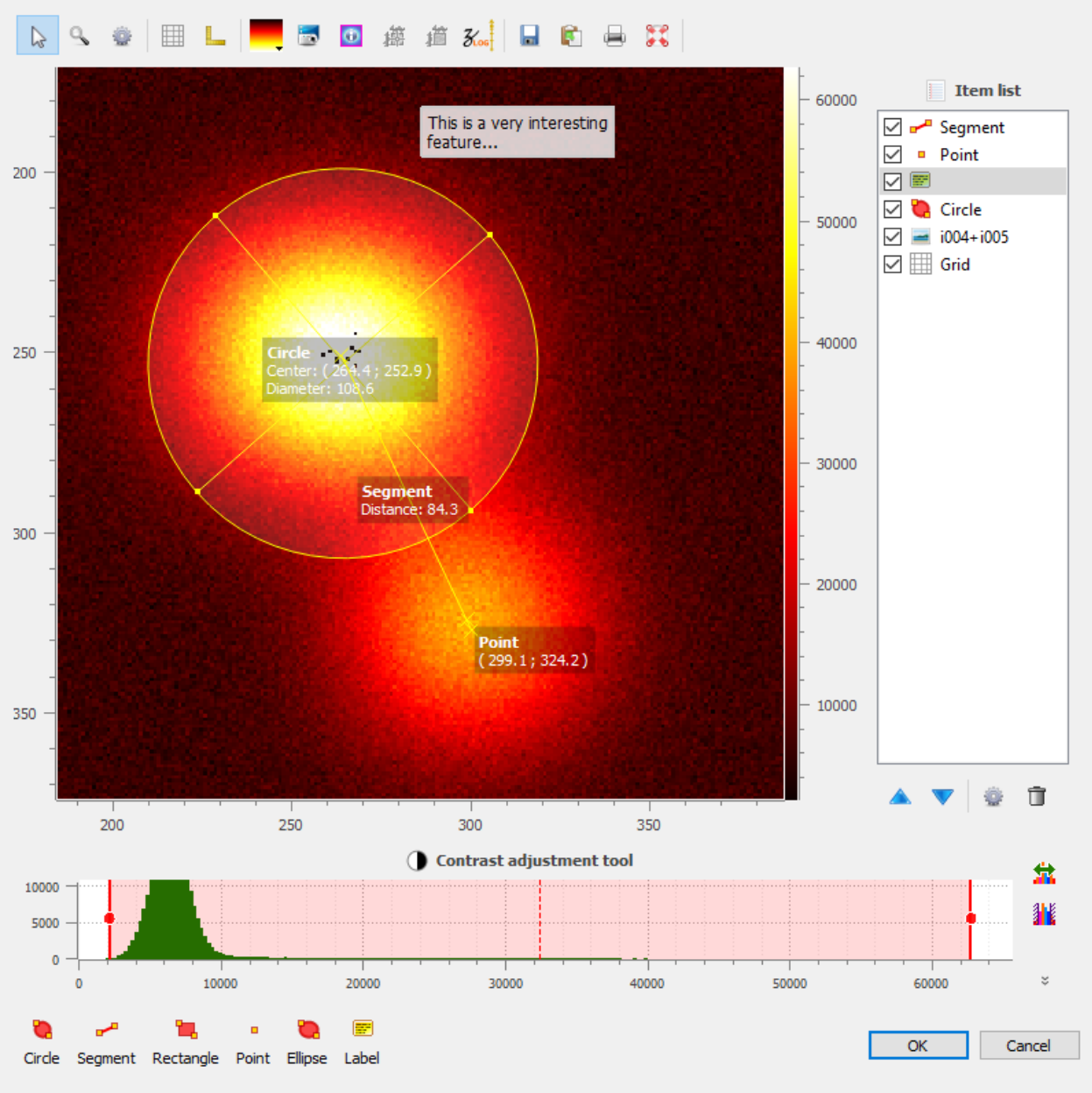
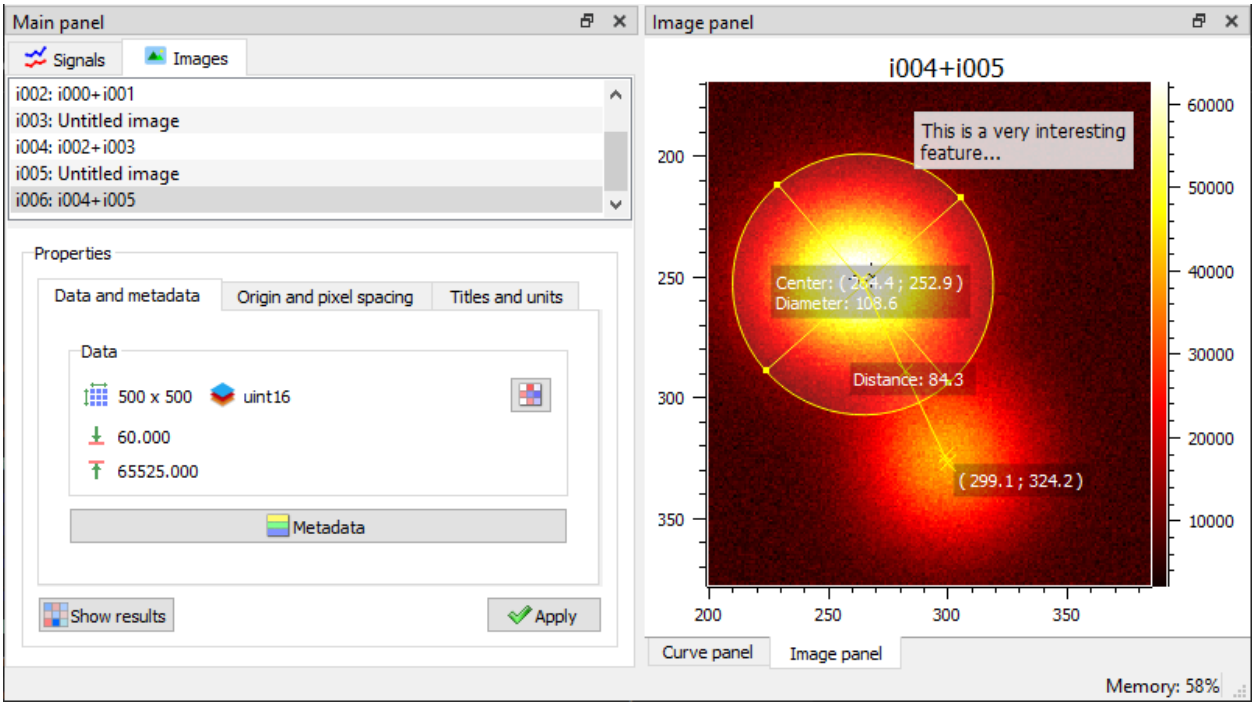


Fig. 13: Example of annotations.


Main panel
Signals
Images
i002: i000+i001
i003: Untitled image
i004: i002+i003
i005: Untitled image
i006: i004+i005
Properties
Data and metadata
Origin and pixel spacing
Titles and units
Data
500 x 500
uint16
60.000
65525.000
Metadata
Show results
Apply
Image panel
i004+i005
This is a very interesting feature...
Center: (264.4 ; 252.9)
Diameter: 108.6
Distance: 84.3
(299.1 ; 324.2)
200
250
300
350
60000
50000
40000
30000
20000
10000
Curve panel
Image panel
Memory: 58%

CHAPTER

FIVE

DEVELOPMENT

5.1 Roadmap

5.1.1 Future milestones

Features

- Add support for multichannel timeseries
- Develop a Jupyter plugin for interactive data analysis connected with DataLab
- Develop a Spyder plugin for interactive data analysis connected with DataLab
- Image computing results (*cdl.model.base.ResultShape*):
  - Add support for "free form" geometrical shapes (this could be used to draw the result of a segmentation algorithm, or the result of an edge detection)
  - Add support for custom geometrical shapes (this could be used to draw the result of a specific algorithm, e.g. a pattern recognition algorithm)

Note: See "TODO" comment just above `cdl.model.base.ResultShape` class definition for more details about how to implement this feature.

Maintenance

- 2025: drop PyQt5 support (end-of-life: mid-2025), and switch to PyQt6 ; this should be straightforward, thanks to the *qtpy* compatibility layer and to the fact that *PlotPyStack* is already compatible with PyQt6)

Other tasks

- Develop a very simple DataLab plugin to demonstrate the plugin system
- Develop a DataLab plugin template
- Make a video tutorial about the plugin system and remote control features

5.1.2 Past milestones

DataLab 0.9

- Python 3.11 is the new reference
- Run computations in a separate process:
  - Execute a "computing server" in background, in another process
  - For each computation, send serialized data and computing function to the server and wait for the result
  - It is then possible to stop any computation at any time by killing the server process and restarting it (eventually after incrementing the communication port number)
- Optimize image displaying performance
- Add preferences dialog box
- Add new image processing features: denoising, ...
- New plugin system: API for third-party extensions
  - Objective #1: a plugin must be manageable using a single Python script, which includes an extension of *ImageProcessor*, *ActionHandler* and new file format support
  - Objective #2: plugins must be simply stored in a folder wich defaults to the user directory (same folder as ".DataLab.ini" configuration file)
- Add a macro-command system:
  - New embedded Python editor
  - Scripts using the same API as high-level applicative test scenarios
  - Support for macro recording
- Add an xmlrpc server to allow DataLab remote control:
  - Controlling DataLab main features (open a signal or an image, open a HDF5 file, etc.) and processing features (run a computation, etc.)
  - Take control of DataLab from a third-party software
  - Run interactive calculations from an IDE (e.g. Spyder or Visual Studio Code)

CodraFT 2.2

- Add default image visualization settings in .INI configuration file

CodraFT 2.1

- "Open in a new window" feature: add support for multiple separate windows, thus allowing to visualize for example two images side by side
- New demo mode
- New command line option features (open/browse HDF5 files at startup)
- ROI features:
  - Add an option to extract multiples ROI on either one signal/image (current behavior) or one signal/image per ROI

  - Images: create ROI using array masks
  - Images: add support for circular ROI

CodraFT 2.0

- New data processing and visualization features (see below)
- Fully automated high-level processing features for internal testing purpose, as well as embedding DataLab in a third-party software
- Extensive test suite (unit tests and application tests) with 90% feature coverage

CodraFT 1.7

- Major redesign
- Python 3.8 is the new reference
- Dropped Python 2 support

CodraFT 1.6

- Last release supporting Python 2

5.2 How to contribute

5.2.1 Fork the project

The first step is to fork the project on GitHub. You can do that by visiting DataLab 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.

5.2.2 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.

See also:

The *Coding guidelines* page presents the coding guidelines that you should follow when contributing to the project.

5.3 Coding guidelines

5.3.1 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:

- black for code formatting.
- isort for import sorting.
- pylint for static code analysis.

black

If you are using Visual Studio Code, the project settings will automatically format your code on save.

Or you may use *black* manually. To format your code, run the following command:

```
black .
```

isort

Again, if you are using Visual Studio Code, the project settings will automatically sort your imports on save.

Or you may use *isort* manually. To sort your imports, run the following command:

```
isort .
```

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.

5.3.2 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.

5.4 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).

5.4.1 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
Black Formatter	Python code formatter
gettext	Gettext syntax highlighting
isort	Python import sorter
Pylance	Python language server
Python	Python extension
reStructuredText Syntax highlighting	reStructuredText syntax highlighting
Ruff	Extremely fast Python linter and code formatter
Todo Tree	Todo tree

5.4.2 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
3.8	OK	3.8.10.0
3.9	OK	3.9.10.0
3.10	OK	3.10.11.1
3.11	OK	3.11.5.0
3.12	OK	3.12.0.1

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.

5.4.3 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).

5.4.4 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\\texmfs\\install\\miktex\\bin\\x64;%PATH%
```

Visual Studio Code configuration used in `launch.json` and `tasks.json` (examples) :

```
@REM Development environment
set CDL_PYTHONEXE=C:\C2OIQ-DevCDL\python-3.8.10.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).

5.4.5 Third-party Software

The following software may be required for maintaining the project:

Software	Description
gettext	Translations
Git	Version control system
ImageMagick	Image manipulation utilities
Inkscape	Vector graphics editor
MikTeX	LaTeX distribution on Windows

CHAPTER

SIX

CHANGELOG

See DataLab roadmap page for future and past milestones.

6.1 DataLab Version 0.10.1

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 “Extract profile” to “Operations” menu, to extract a profile from an image along a row or a column
    - Added “Extract 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 ("Computing" 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 a **PlotPy** issue (https://github.com/PlotPyStack/PlotPy/issues/9), and was 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`

6.2 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 regions of interest 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

6.3 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

6.4 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 "Computing 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 computing 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
    * Sigmoïd 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°, -90°)
    * 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

CHAPTER

SEVEN

COPYRIGHTS AND LICENSING

PYTHON MODULE INDEX

C