



**DataLab**  
***Release 0.9.1***

**Dec 01, 2023**



## CONTENTS:

<b>1</b>	<b>Getting started</b>	<b>3</b>
1.1	DataLab in a nutshell . . . . .	3
1.2	What are the applications for Datalab? . . . . .	3
1.3	How does DataLab work? . . . . .	4
<b>2</b>	<b>General features</b>	<b>11</b>
2.1	Command line features . . . . .	12
2.2	HDF5 Browser . . . . .	15
2.3	Remote controlling . . . . .	15
2.4	Internal data model . . . . .	36
2.5	Plugins . . . . .	48
2.6	Log viewer . . . . .	58
2.7	Installation and configuration viewer . . . . .	58
<b>3</b>	<b>Signal processing</b>	<b>61</b>
3.1	“File” menu . . . . .	62
3.2	“Edit” menu . . . . .	63
3.3	“Operation” menu . . . . .	64
3.4	“Processing” menu . . . . .	67
3.5	“Computing” menu . . . . .	68
3.6	“View” menu . . . . .	70
3.7	“?” menu . . . . .	71
3.8	Annotations (Signals) . . . . .	72
<b>4</b>	<b>Image processing</b>	<b>75</b>
4.1	“File” menu . . . . .	76
4.2	“Edit” menu . . . . .	77
4.3	“Operation” menu . . . . .	78
4.4	“Processing” menu . . . . .	80
4.5	“Computing” menu . . . . .	85
4.6	“View” menu . . . . .	87
4.7	“?” menu . . . . .	89
4.8	2D Peak Detection . . . . .	90
4.9	Contour Detection . . . . .	93
4.10	Annotations (Images) . . . . .	97
<b>5</b>	<b>Development</b>	<b>101</b>
5.1	Roadmap . . . . .	101
5.2	How to contribute . . . . .	103
5.3	Coding guidelines . . . . .	104

5.4	Setting up Development Environment . . . . .	105
<b>6</b>	<b>Changelog</b>	<b>109</b>
6.1	DataLab Version 0.9.1 . . . . .	109
6.2	DataLab Version 0.9.0 . . . . .	110
6.3	Older releases . . . . .	112
<b>7</b>	<b>Copyrights and licensing</b>	<b>125</b>
	<b>Python Module Index</b>	<b>127</b>

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.

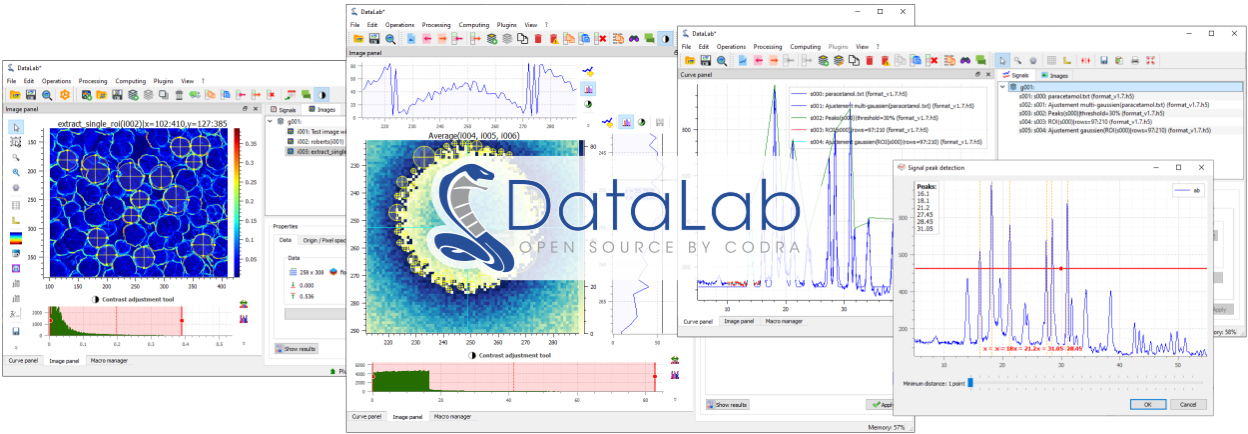


Fig. 1: Signal and image visualization in DataLab

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



## 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	>= 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:

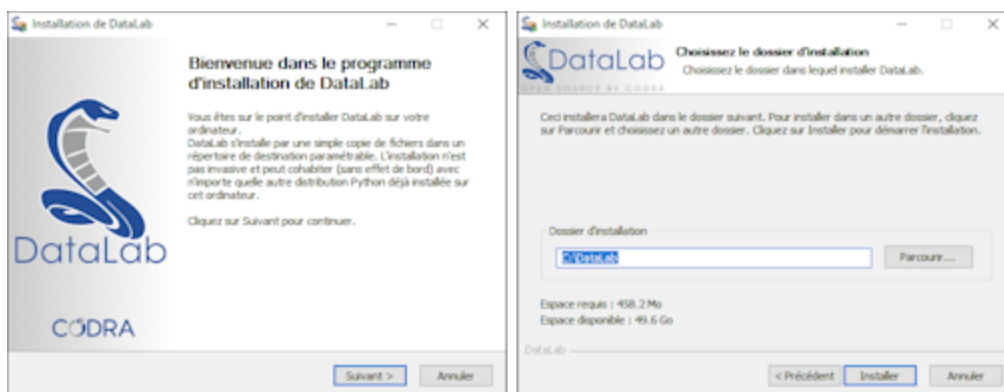
Name	Version	Summary
PyQt5		Python bindings for the Qt cross platform application toolkit
sphinx		Python documentation generator
sphinx_intl		Sphinx utility that make it easy to translate and to apply translation.
myst_parser		An extended [CommonMark]( <a href="https://spec.commonmark.org/">https://spec.commonmark.org/</a> ) 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 running 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 straightforward:

```
$ 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

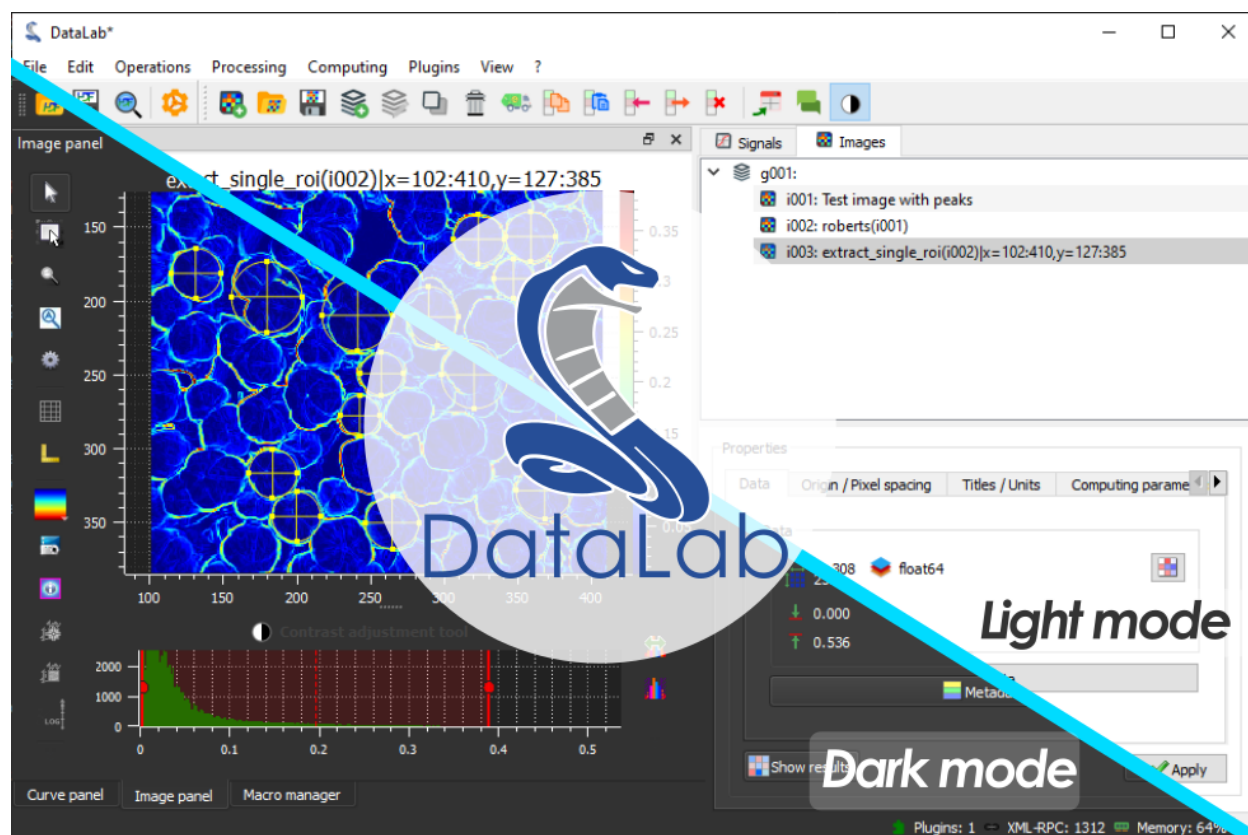


Fig. 1: DataLab supports dark and light mode depending on your OS settings

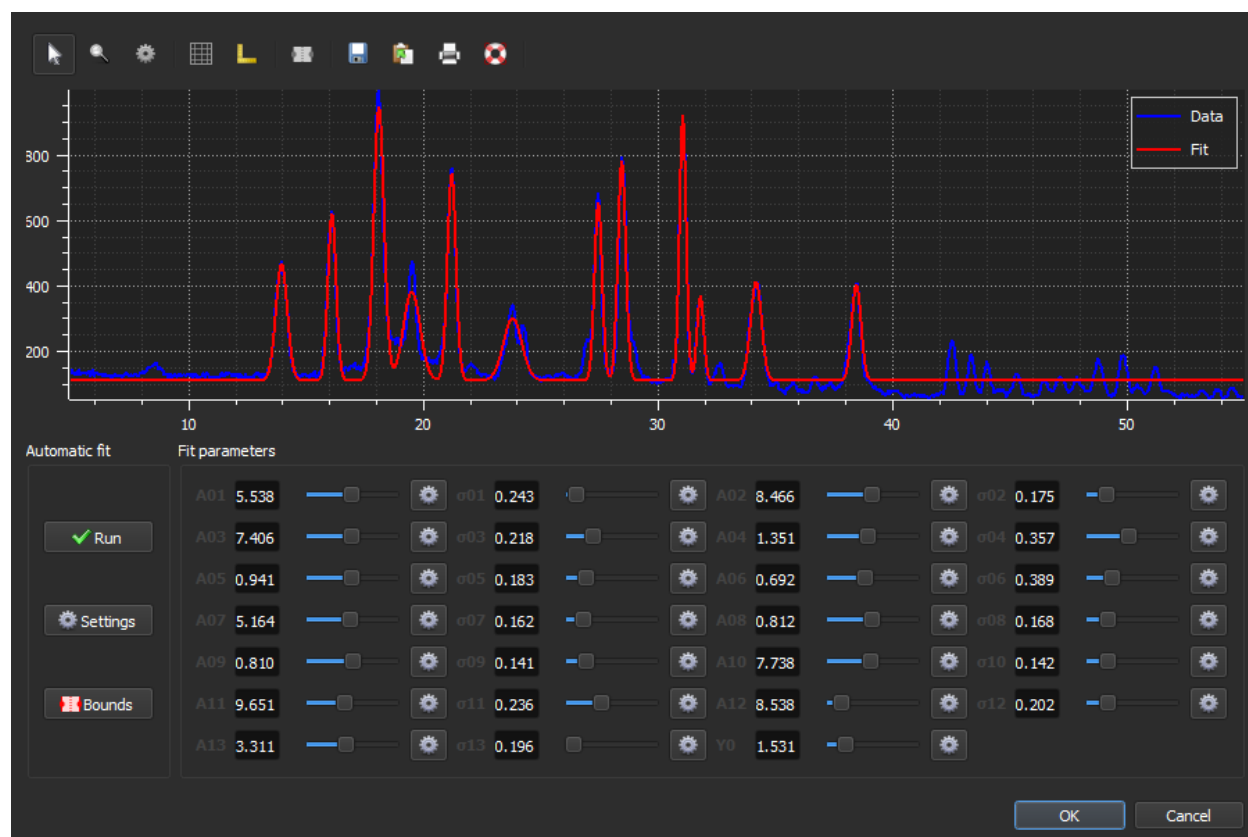


Fig. 2: Example of a multi-gaussian curve fit

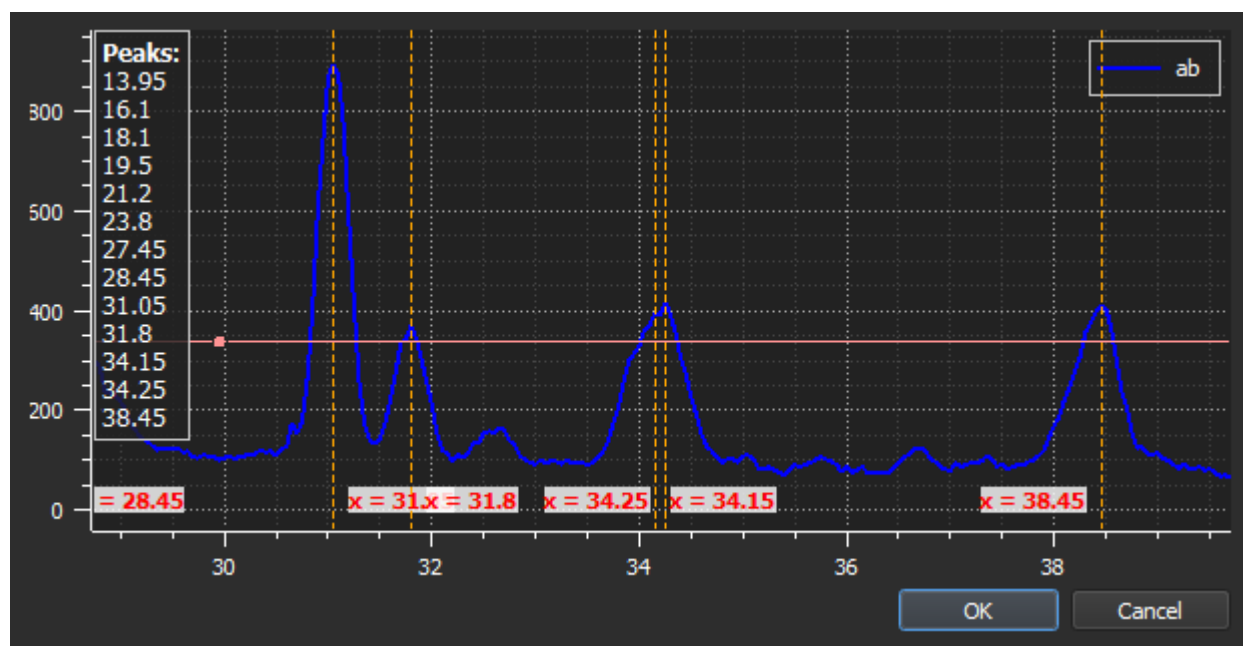


Fig. 3: Semi-automatic peak detection

## 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, Lorentz, Voigt, polynomial
✓		Interactive multigaussian fit
✓	✓	Computing on custom ROI
✓		FWHM, FW @ $1/e^2$
	✓	Centroid (robust method w/r noise)
	✓	Minimum enclosing circle center



## GENERAL FEATURES

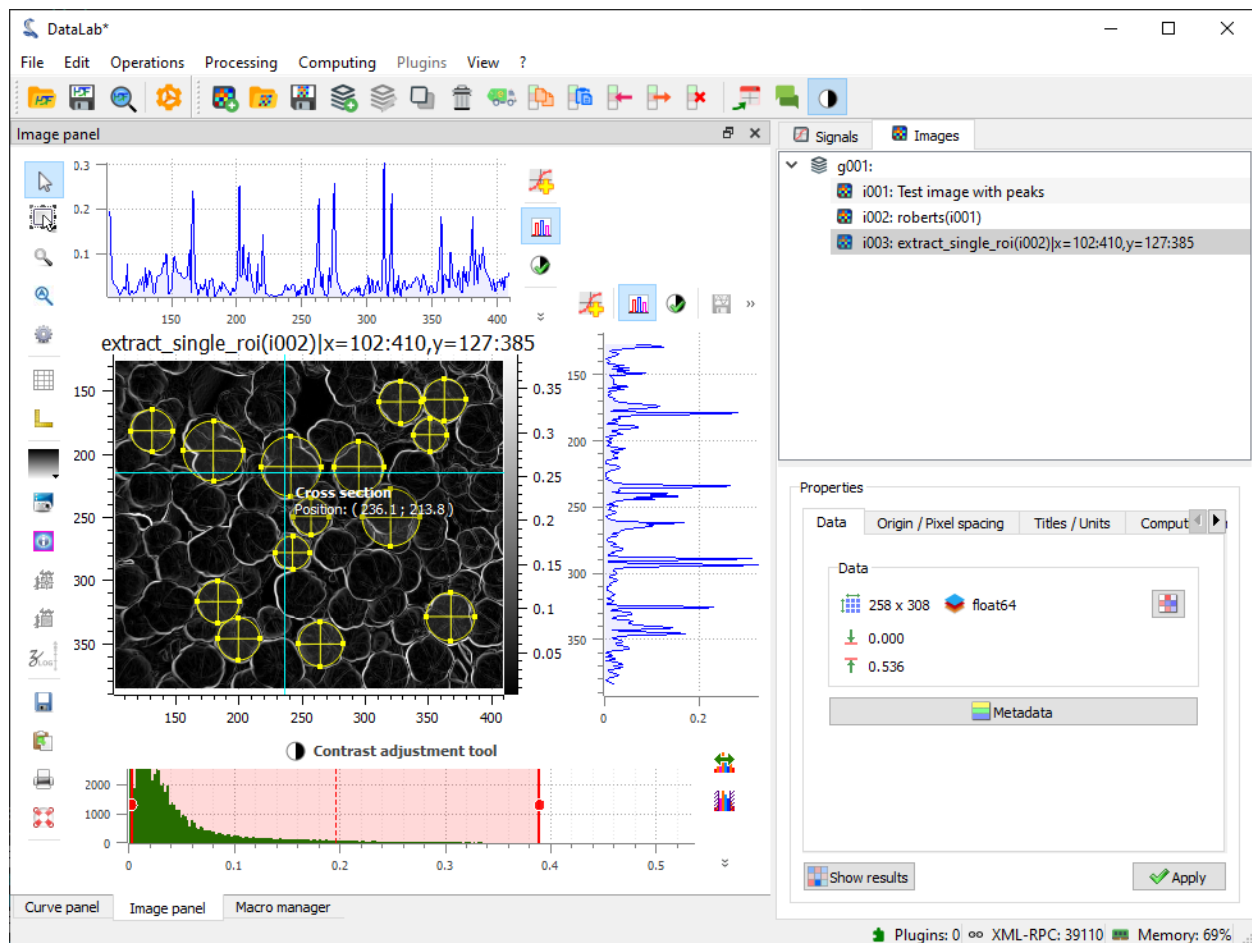


Fig. 1: DataLab main window

## 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.guittest* 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).

```

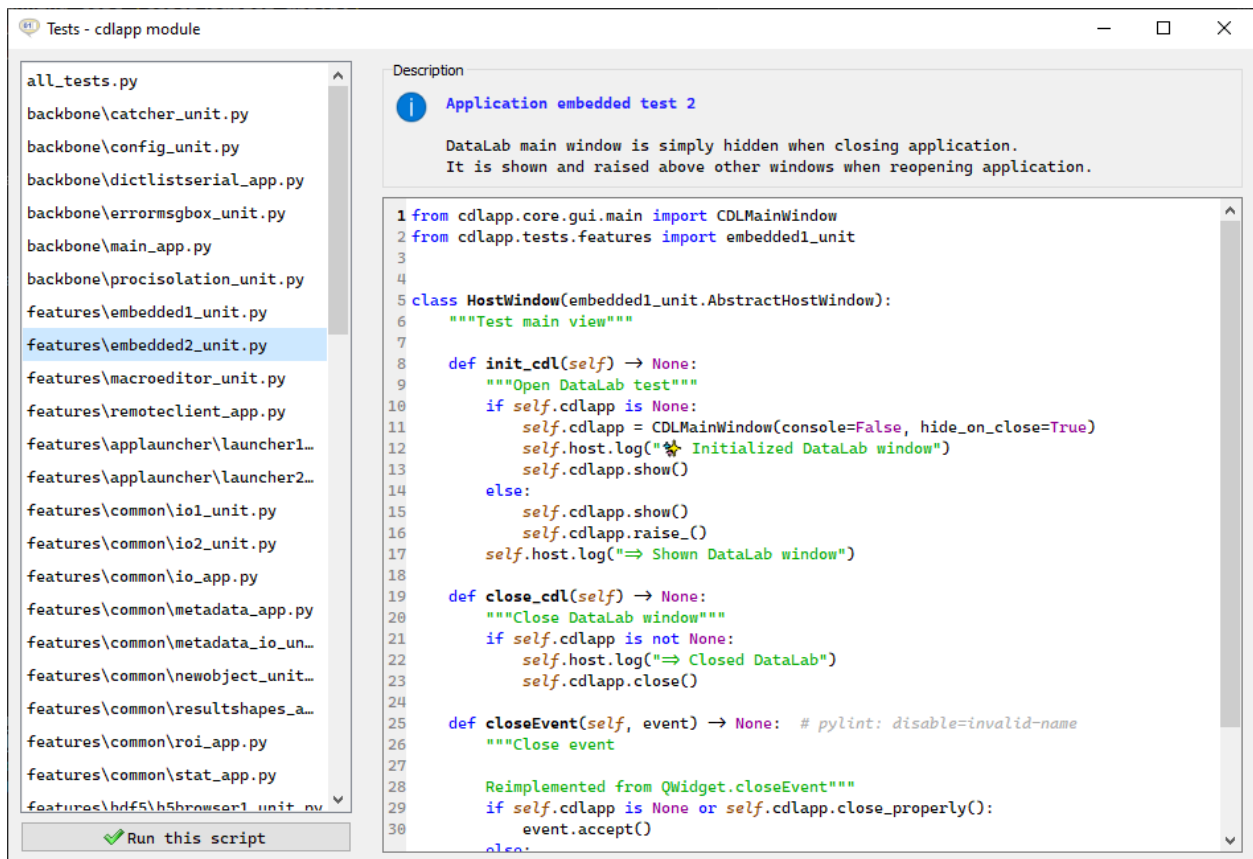
===[01/51]=== Running test [tests\annotations_app.py]
===[02/51]=== Running test [tests\annotations_unit.py]
===[03/51]=== Running test [tests\auto_app.py]
===[04/51]=== Running test [tests\basic1_app.py]
===[05/51]=== Running test [tests\basic2_app.py]
===[06/51]=== Running test [tests\basic3_app.py]

```

## 2.1.6 Run interactive tests

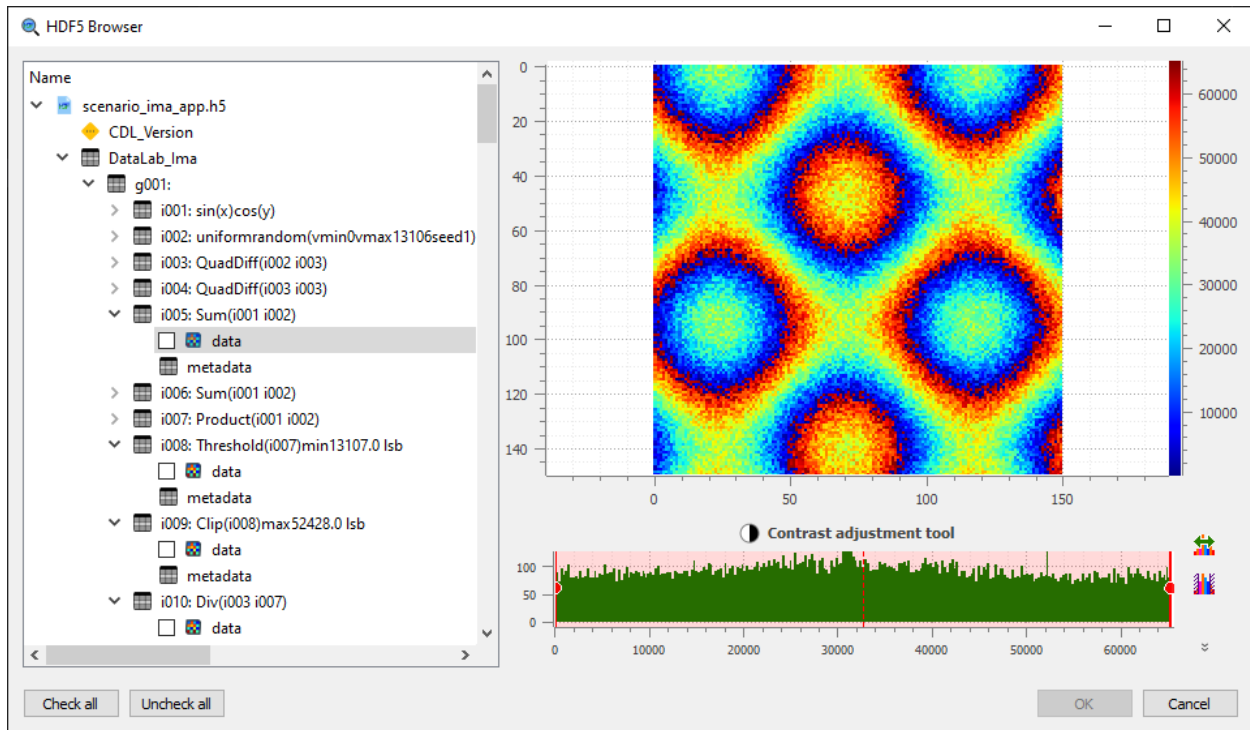
To execute DataLab interactive tests, run the following:

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

### 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`

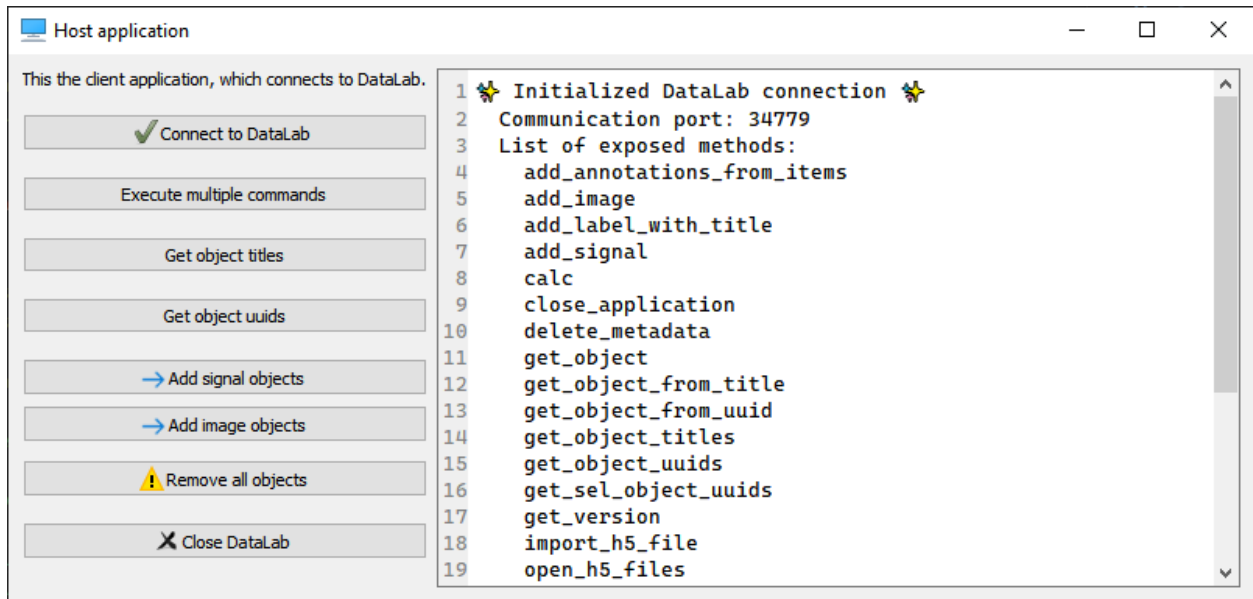


Fig. 2: Screenshot of remote client application test (`cdl.tests.remoteclient_app`)

## 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 (...)

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()

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

proxy.set_current_panel("image")

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

proxy.compute_fft()

```

Here is a Python 2.7 reimplementaion 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"""
    dbytes = io.BytesIO()
    np.save(dbytes, data, allow_pickle=False)
    return Binary(dbytes.getvalue())

def get_cdl_xmlrpc_port():

```

(continues on next page)

(continued from previous page)

```

"""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()

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

    def reset_all(self):

```

(continues on next page)

(continued from previous page)

```

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

    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,
    ):
        """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)

```

(continues on next page)



(continued from previous page)

```

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.

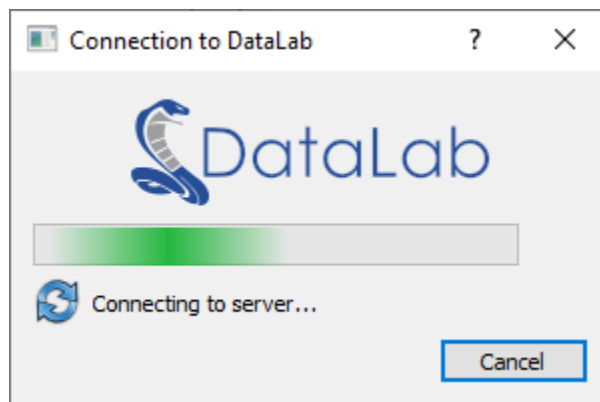


Fig. 3: Screenshot of connection dialog (`cdl.widgets.connection.ConnectionDialog`)

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

```

(continues on next page)

(continued from previous page)

```

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

### DataLab remote control

This module provides utilities to control DataLab from a Python script (e.g. with Spyder) or from a Jupyter notebook. The *RemoteClient* class provides the main interface to DataLab XML-RPC server.

`cdl.core.remote.array_to_rpcbinary(data: ndarray) → Binary`

Convert NumPy array to XML-RPC Binary object, with shape and dtype.

The array is converted to a binary string using NumPy's native binary format.

#### Parameters

**data** – NumPy array to convert

#### Returns

XML-RPC Binary object

`cdl.core.remote.rpcbinary_to_array(binary: Binary) → ndarray`

Convert XML-RPC binary to NumPy array.

#### Parameters

**binary** – XML-RPC Binary object

#### Returns

NumPy array

`cdl.core.remote.dataset_to_json(param: DataSet) → list[str]`

Convert guidata DataSet to JSON data.

The JSON data is a list of three elements:

- The first element is the module name of the DataSet class
- The second element is the class name of the DataSet class

- The third element is the JSON data of the DataSet instance

**Parameters**

**param** – guidata DataSet to convert

**Returns**

JSON data

`cdl.core.remote.json_to_dataset(param_data: list[str]) → DataSet`

Convert JSON data to guidata DataSet.

**Parameters**

**param\_data** – JSON data

**Returns**

guidata DataSet

`cdl.core.remote.remote_call(func: Callable) → object`

Decorator for method calling DataLab main window remotely

**class** `cdl.core.remote.RemoteServer(win: CDLMainWindow)`

XML-RPC server QThread

**serve()** → None

Start server and serve forever

**notify\_port(port: int) → None**

Notify automatically attributed port.

This method is called after the server port has been automatically attributed. It notifies the port number to the main window.

**Parameters**

**port** – Server port number

**classmethod** `check_remote_functions()` → None

Check if all AbstractCDLControl methods are implemented in RemoteServer

**register\_functions(server: SimpleXMLRPCServer) → None**

Register functions

**run()** → None

Thread execution method

**cdl\_is\_ready()** → None

Called when DataLab is ready to process new requests

**static** `get_version()` → str

Return DataLab version

**close\_application()** → None

Close DataLab application

**raise\_window()** → None

Raise DataLab window

**get\_current\_panel()** → str

Return current panel name.

**Returns**

Panel name (valid values: 'signal', 'image', 'macro')

**Return type**

str

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

Switch to panel.

**Parameters**

**panel** (str) – Panel name (valid values: 'signal', 'image', 'macro')

**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 (with extension .h5)

**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) – HDF5 file names. Defaults to None.
- **import\_all** (bool | None) – Import all objects from HDF5 file. Defaults to None.
- **reset\_all** (bool | None) – Reset all application data. Defaults to None.

**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\_object**(filename: str) → None

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

**Parameters**

**filename** (str) – File name

**add\_signal**(title: str, xbinary: Binary, ybinary: Binary, 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
- **xbinary** (Binary) – X data
- **ybinary** (Binary) – 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 successful

**Return type**

*bool*

**add\_image**(*title: str*, *zbinary: Binary*, *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
- **zbinary** (*Binary*) – Z 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 successful

**Return type**

*bool*

**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.

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

Select objects in current panel.

**Parameters**

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

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

Select groups in current panel.

**Parameters**

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

**delete\_metadata**(*refresh\_plot*: *bool* = *True*) → *None*

Delete metadata of selected objects

**Parameters**

**refresh\_plot** (*bool* | *None*) – Refresh plot. Defaults to *True*.

**calc**(*name*: *str*, *param\_data*: *list[str]* | *None* = *None*) → *bool*

Call compute function name in current panel's processor.

**Parameters**

- **name** (*str*) – Compute function name
- **param\_data** (*list[str]* | *None*) – Compute function parameters. Defaults to *None*.

**Returns**

*True* if successful

**Return type**

*bool*

**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. Defaults to *None*.

**Returns**

Object titles

**Return type**

*list[str]*

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

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\_uuids**(*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. Defaults to *None*.

**Returns**

Object uuids

**Return type**

*list*[*str*]

**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\_json*: *str*, *refresh\_plot*: *bool* = *True*, *panel*: *str* | *None* = *None*) → *None*

Add object annotations (annotation plot items).

**Parameters**

- **items\_json** (*str*) – JSON string of annotation 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\_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.

**cdl.core.remote.get\_cdl\_xmlrpc\_port()** → *str*

Return DataLab current XML-RPC port

**Returns**

XML-RPC port

**Raises**

**ConnectionRefusedError** – DataLab has not yet been executed

**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
>>> 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

## 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\_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, fitdglfunc*)  
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\_swap\_axes()** → *None*  
 Swap data axes

**compute\_abs()** → *None*  
 Compute absolute value

**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: cdl.core.computation.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

**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*

**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 annotated 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 annotated 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<sub>index</sub>

**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

**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

**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(
    src: dlo.SignalObj, p: test_data.GaussianNoiseParam
) -> dlo.SignalObj:
    """Add gaussian noise to signal"""
    dst = cpsig.dst_1l(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:
```

(continues on next page)

(continued from previous page)

```

"""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(
            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))

```

(continues on next page)

(continued from previous page)

```

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"""
    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)

```

(continues on next page)

(continued from previous page)

```

# Plugin menu entries -----
def create_actions(self) -> None:
    """Create actions"""
    # Signal panel -----
    sah = self.signalpanel.acthandler
    with sah.new_menu(_("Test data")):
        sah.new_action(_("Add noise to signal"), triggered=self.add_noise_to_signal)
        sah.new_action(
            _("Load spectrum of paracetamol"),
            triggered=self.create_paracetamol_signal,
            select_condition="always",
            separator=True,
        )
        sah.new_action(
            _("Create noisy signal"),
            triggered=self.create_noisy_signal,
            select_condition="always",
        )
    # Image panel -----
    iah = self.imagepanel.acthandler
    with iah.new_menu(_("Test data")):
        iah.new_action(_("Add noise to image"), triggered=self.add_noise_to_image)
        # with iah.new_menu(_("Data samples")):
        iah.new_action(
            _("Create image with peaks"),
            triggered=self.create_peak2d_image,
            select_condition="always",
            separator=True,
        )
        iah.new_action(
            _("Create 2D sin cos image"),
            triggered=self.create_sincos_image,
            select_condition="always",
        )
        iah.new_action(
            _("Create 2D noisy gauss image"),
            triggered=self.create_noisygauss_image,
            select_condition="always",
        )
        iah.new_action(
            _("Create 2D multi gauss image"),
            triggered=self.create_multigauss_image,
            select_condition="always",
        )
        iah.new_action(
            _("Create annotated image"),
            triggered=self.create_annotated_image,
            select_condition="always",
        )
        iah.new_action(
            _("Create 2D step image"),

```

(continues on next page)



(continued from previous page)

```

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

    HEADER = "<111111ffl"

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

```

(continues on next page)

(continued from previous page)

```

self.nbcolls = None # long
self.nbrows = None # long
self.nbframes = None # long
self.pixeltype = None # long
self.quantlevels = None # long
self.maxlevel = None # float
self.minlevel = None # float
self.comment_length = None # long
self.fname = None
self.data = None
if fname is not None:
    self.load(fname)

def __repr__(self) -> str:
    """Return a string representation of the object"""
    info = (
        ("Image width", f"{self.nbcolls:d}"),
        ("Image Height", f"{self.nbrows:d}"),
        ("Frame number", f"{self.nbframes:d}"),
        ("File format", f"{self.file_format:d}"),
        ("Pixel type", f"{self.pixeltype:d}"),
        ("Quantlevels", f"{self.quantlevels:d}"),
        ("Min. level", f"{self.minlevel:f}"),
        ("Max. level", f"{self.maxlevel:f}"),
        ("Comment length", f"{self.comment_length:d}"),
    )
    desc_len = max(len(d) for d in list(zip(*info))[0]) + 3
    res = ""
    for description, value in info:
        res += ("{: " + str(desc_len) + "}}{}\n").format(description + ": ", value)

    res = object.__repr__(self) + "\n" + res
    return res

def load(self, fname: str) -> None:
    """Load header and image pixel data

    Args:
        fname (str): path to FXD file
    """
    with open(fname, "rb") as data_file:
        header_s = struct.Struct(self.HEADER)
        record = data_file.read(9 * 4)
        unpacked_rec = header_s.unpack(record)
        (
            self.file_format,
            self.nbcolls,
            self.nbrows,
            self.nbframes,
            self.pixeltype,
            self.quantlevels,
            self.maxlevel,

```

(continues on next page)

(continued from previous page)

```

        self.minlevel,
        self.comment_length,
    ) = unpacked_rec
    if self.__debug:
        print(unpacked_rec)
        print(self)
    data_file.seek(128 + self.comment_length)
    if self.pixeltype == 0:
        size, dtype = 4, np.float32
    elif self.pixeltype == 1:
        size, dtype = 2, np.uint16
    elif self.pixeltype == 2:
        size, dtype = 1, np.uint8
    else:
        raise NotImplementedError(f"Unsupported pixel type: {self.pixeltype}")
    block = data_file.read(self.nbrows * self.nbcolls * size)
    data = np.fromstring(block, dtype=dtype)
    self.data = data.reshape(self.nbrows, self.nbcolls)

class FXDImageFormat(ImageFormatBase):
    """Object representing Thales Pixium (FXD) image file type"""

    FORMAT_INFO = FormatInfo(
        name="Thales Pixium",
        extensions="*.fxd",
        readable=True,
        writeable=False,
    )

    @staticmethod
    def read_data(filename: str) -> np.ndarray:
        """Read data and return it

        Args:
            filename (str): path to FXD file

        Returns:
            np.ndarray: image data
        """
        fxd_file = FXDFile(filename)
        return fxd_file.data

# =====
# Dürr NDT XYZ file format
# =====

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

```

(continues on next page)

(continued from previous page)

```

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

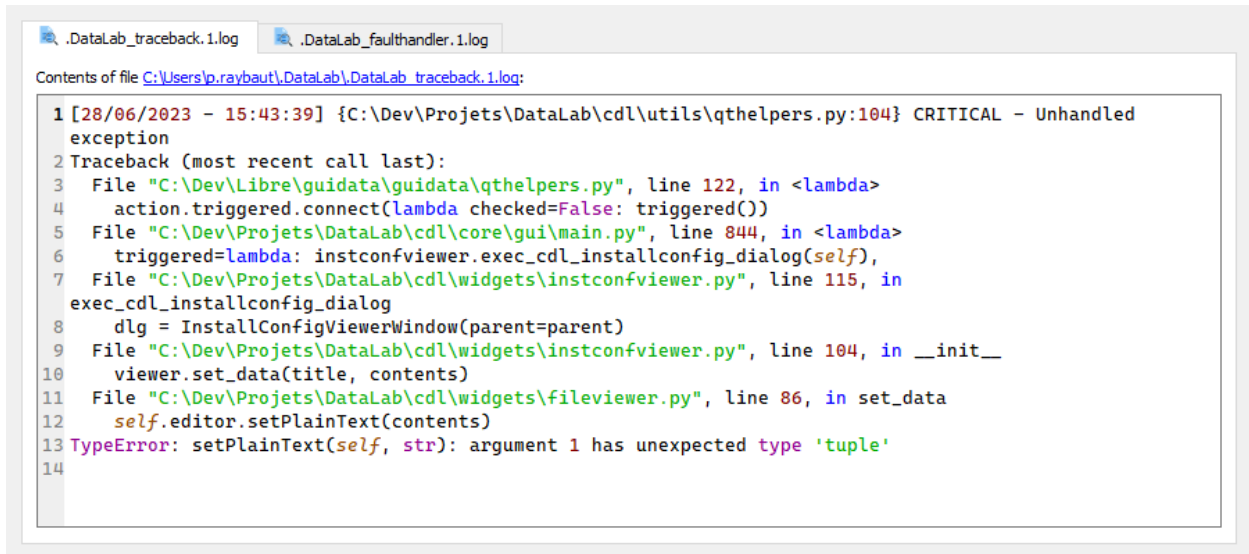


Fig. 4: DataLab log viewer (see “?” menu)

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.

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](#)).

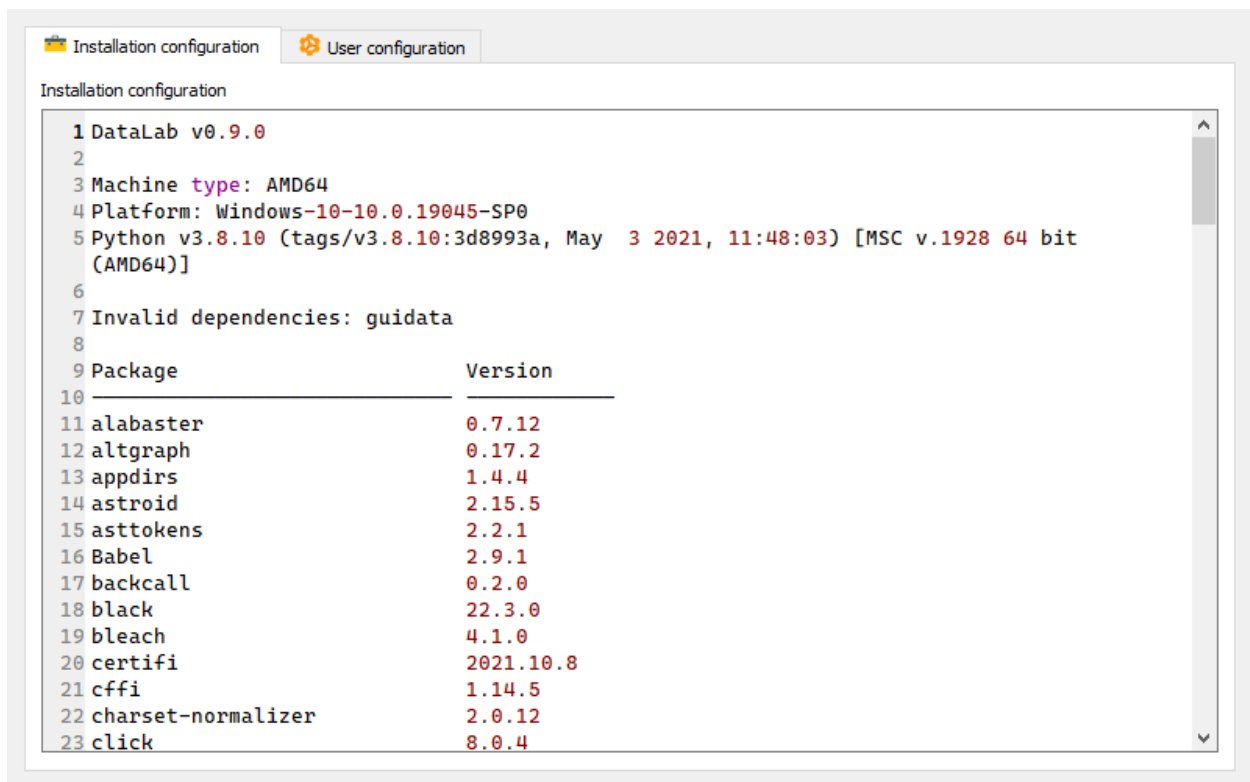


Fig. 5: Installation and configuration (see “?” menu)





## SIGNAL PROCESSING

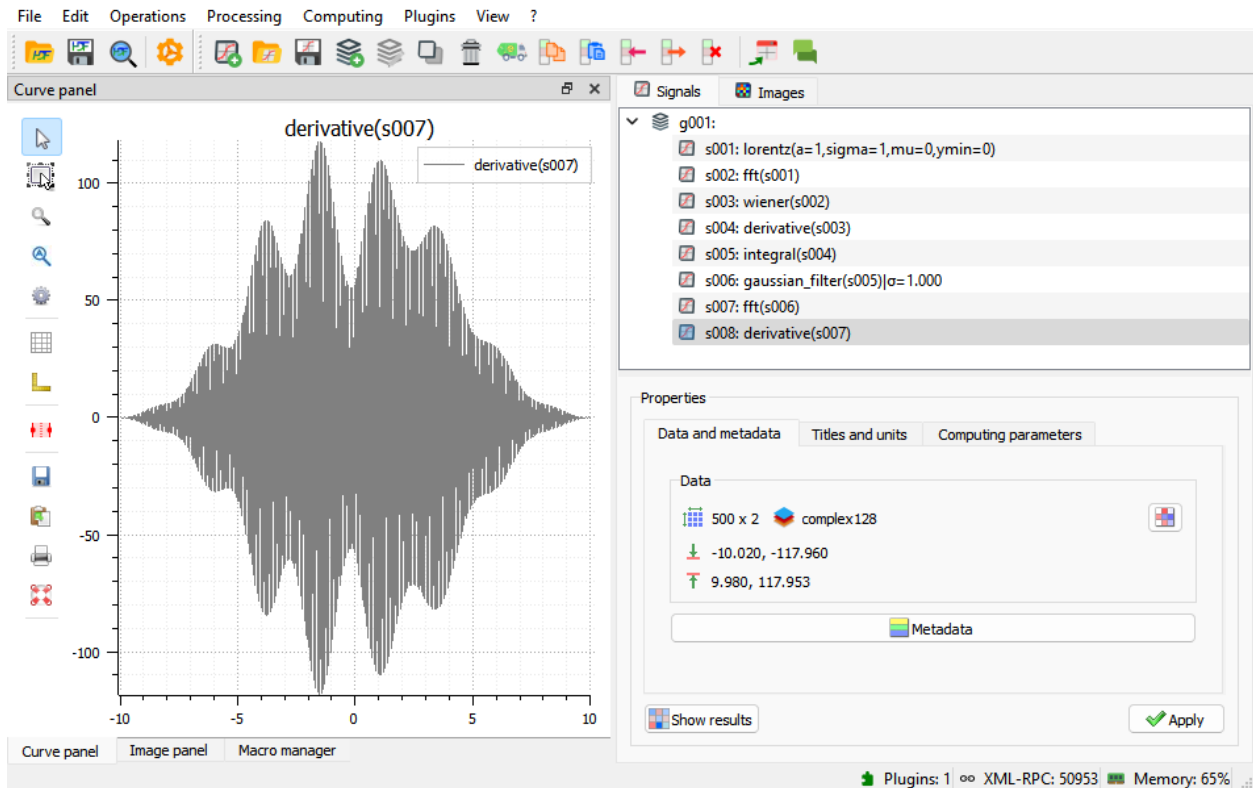
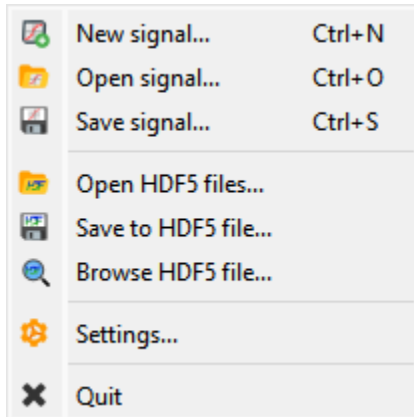


Fig. 1: DataLab main window: Signal processing view

### 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	$y = y_0 + \frac{A}{\sqrt{2\pi} \cdot \sigma} \cdot \exp(-\frac{1}{2} \cdot (\frac{x - x_0}{\sigma})^2)$
Lorentzian	$y = y_0 + \frac{A}{\sigma \cdot \pi} \cdot \frac{1}{1 + (\frac{x - x_0}{\sigma})^2}$
Voigt	$y = y_0 + A \cdot \frac{\text{Re}(\exp(-z^2)) \cdot \text{erfc}(-j \cdot z)}{\sqrt{2\pi} \cdot \sigma}$ with $z = \frac{x - x_0 - j \cdot \sigma}{\sqrt{2} \cdot \sigma}$

#### 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

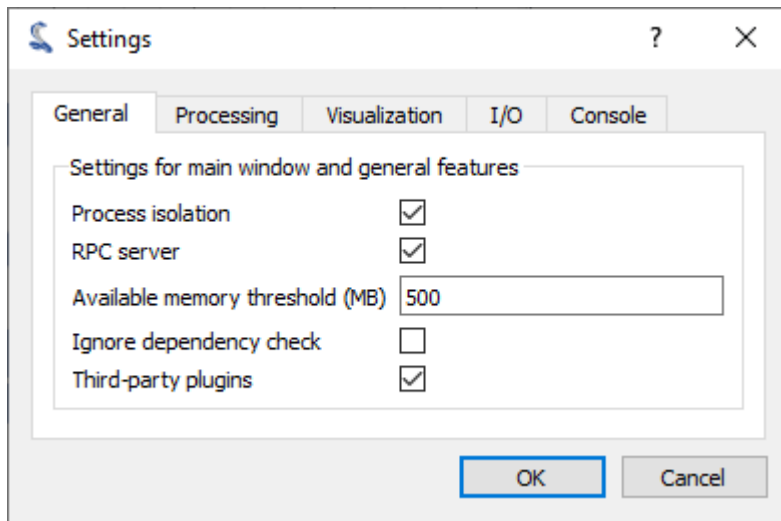
Export the whole DataLab session (all signals and images) into a HDF5 file.

#### Browse HDF5 file

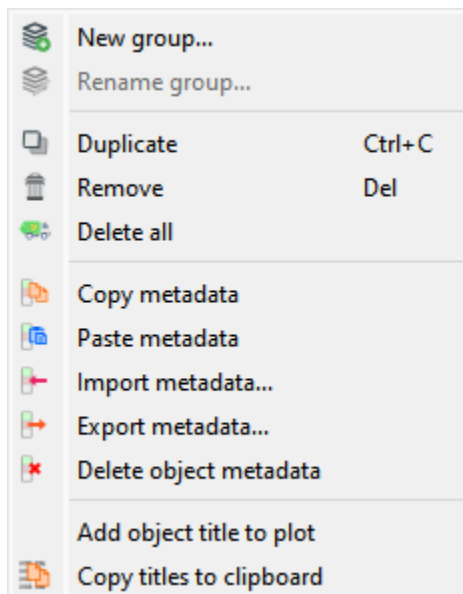
Open the [HDF5 Browser](#) in a new window to browse and import data from HDF5 file.

## 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 additional information such as Region of Interest or results of computations

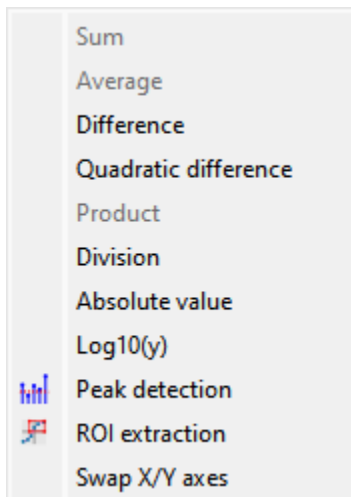
**Add object title to plot**

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:

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

**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.

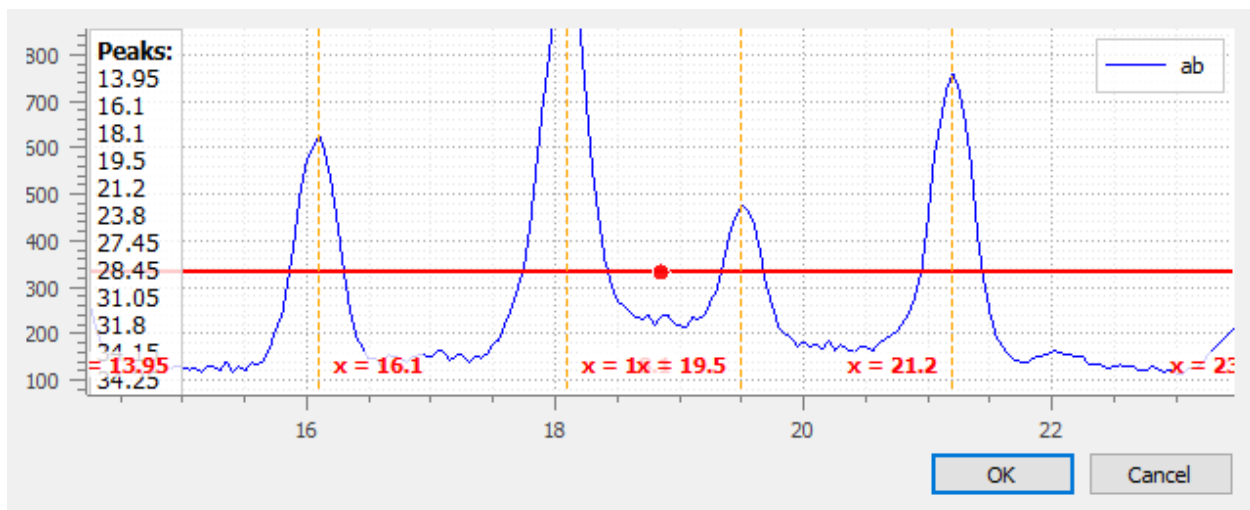


Fig. 2: Peak detection dialog: threshold is adjustable by moving the horizontal marker, peaks are detected automatically (see vertical markers with labels indicating peak position)

**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.

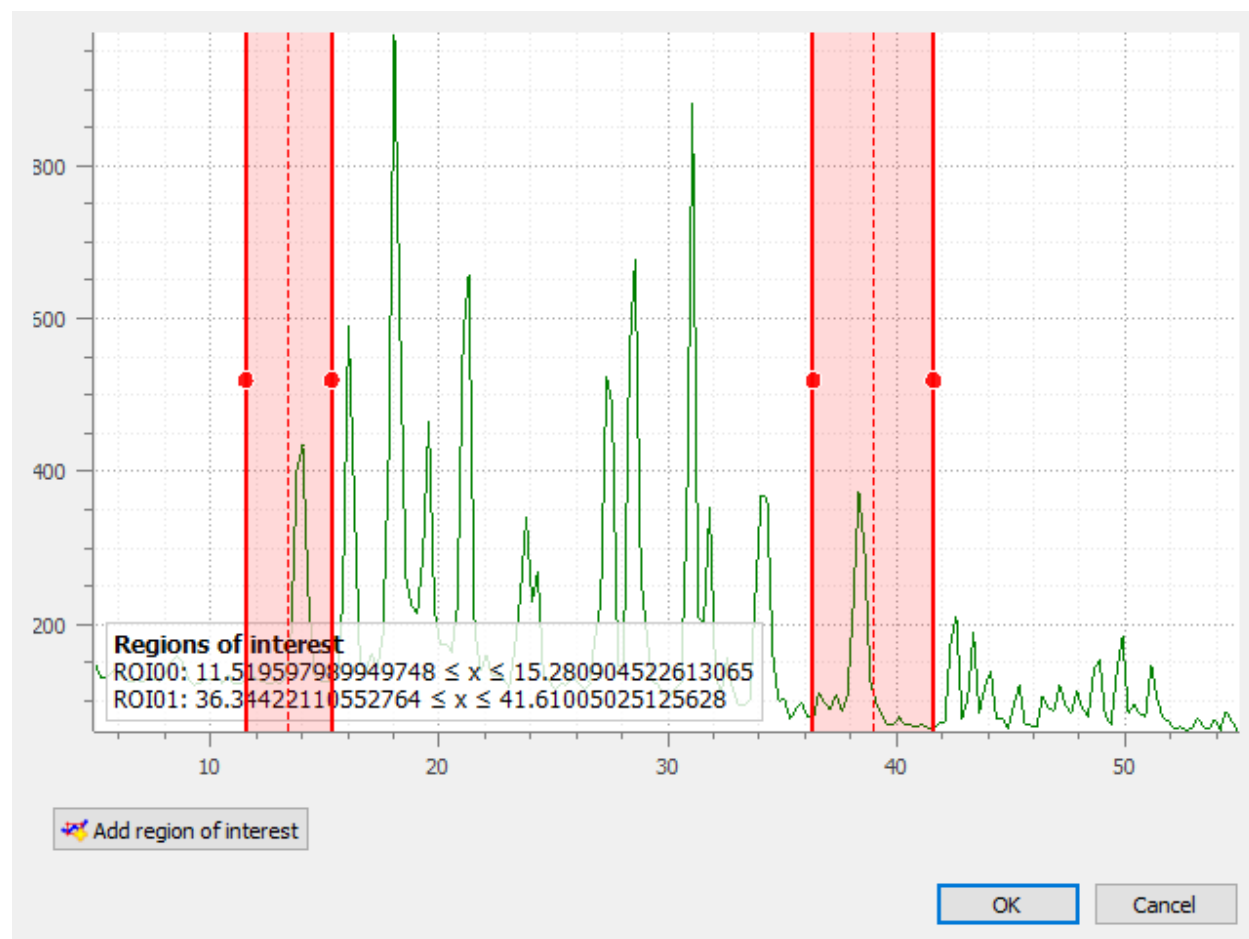
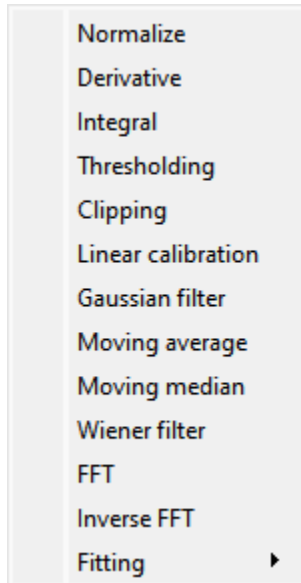


Fig. 3: ROI extraction dialog: the ROI is defined by moving the position and adjusting the width of an horizontal range.

### 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	$y_1 = \frac{y_0}{\sum_{n=0}^N  y_0[n] ^2}$

#### 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

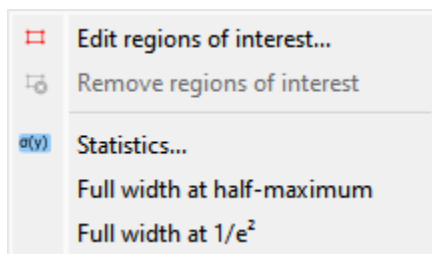
Create a new signal which is the inverse FFT of each selected signal.

#### Lorentzian, Voigt, Polynomial and Multi-Gaussian fit

Open an interactive curve fitting tool in a modal dialog box.

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

## 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.



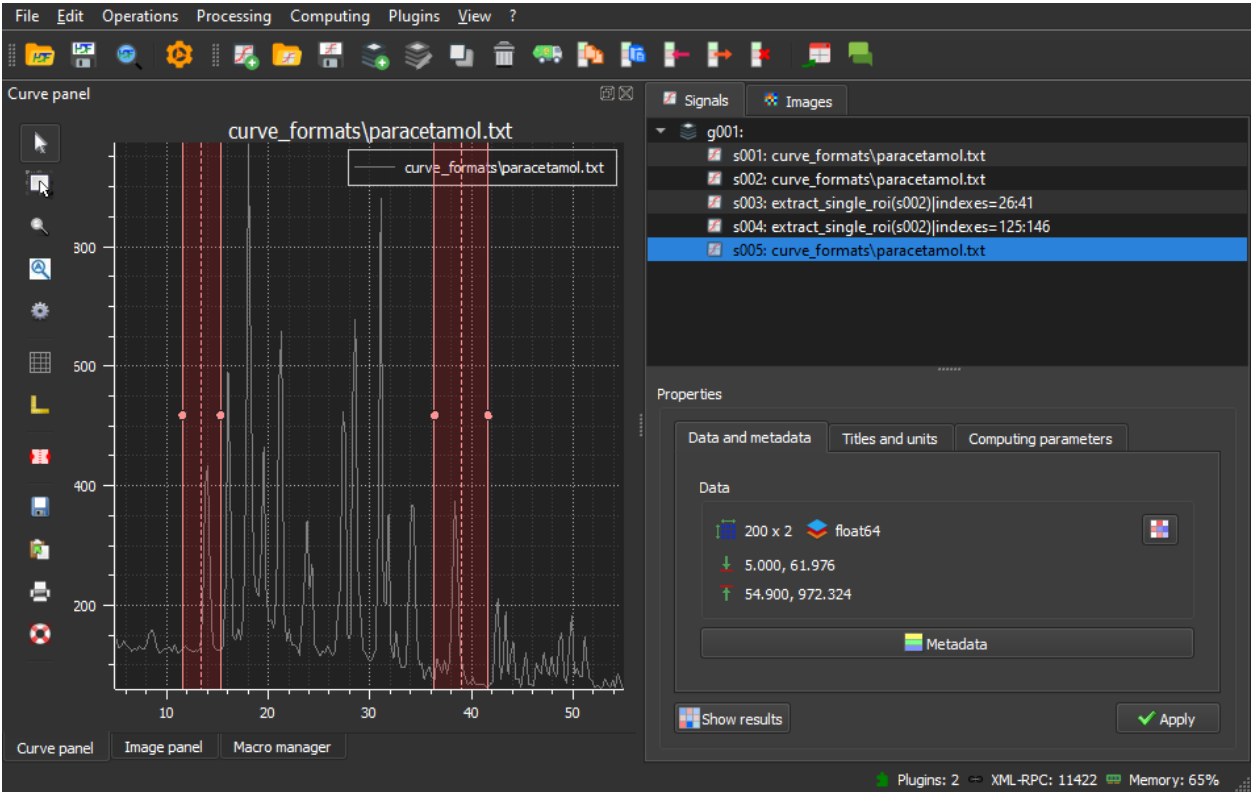


Fig. 4: A signal with an ROI.

	min(y)	max(y)	<y>	$\sigma(y)$	$\Sigma(y)$	$f_{ydx}$
s000	7.6946e-23	0.398862	0.0499	0.107641	24.95	1
s000 ROI00	1.1479e-22	0.398862	0.0501004	0.10781	12.475	0.492007

Format

Resize

☒ Background color

Close

Fig. 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.

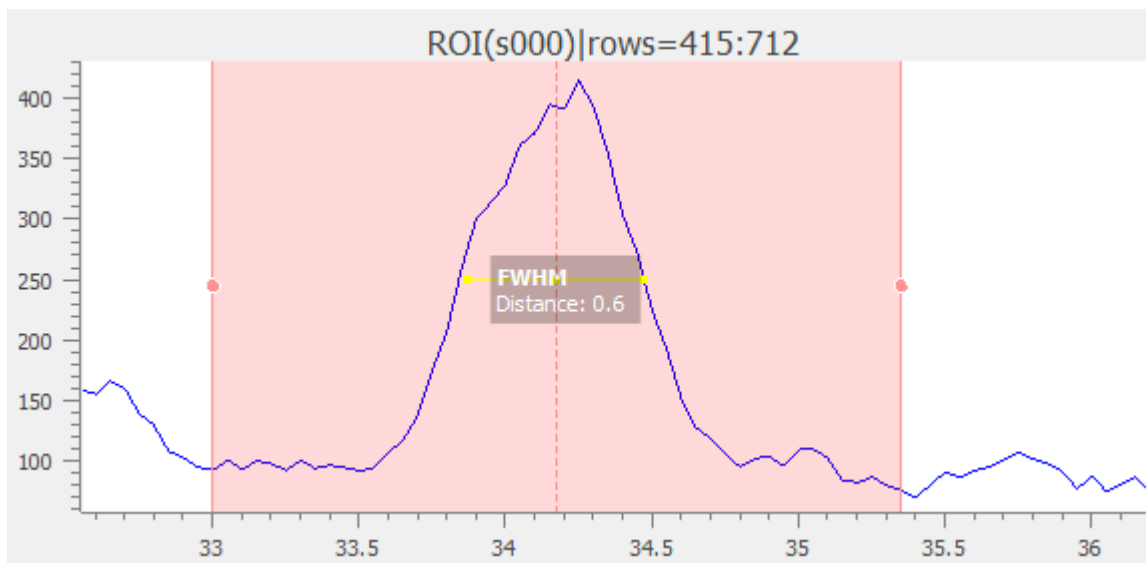


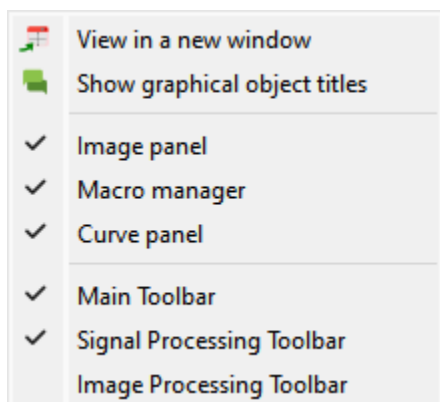
Fig. 6: The computed result is displayed as an annotated segment.

**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.

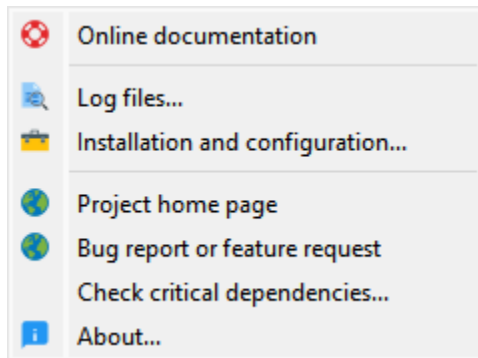
### Show graphical object titles

Show/hide titles of computing results or annotations.

### Other menu entries

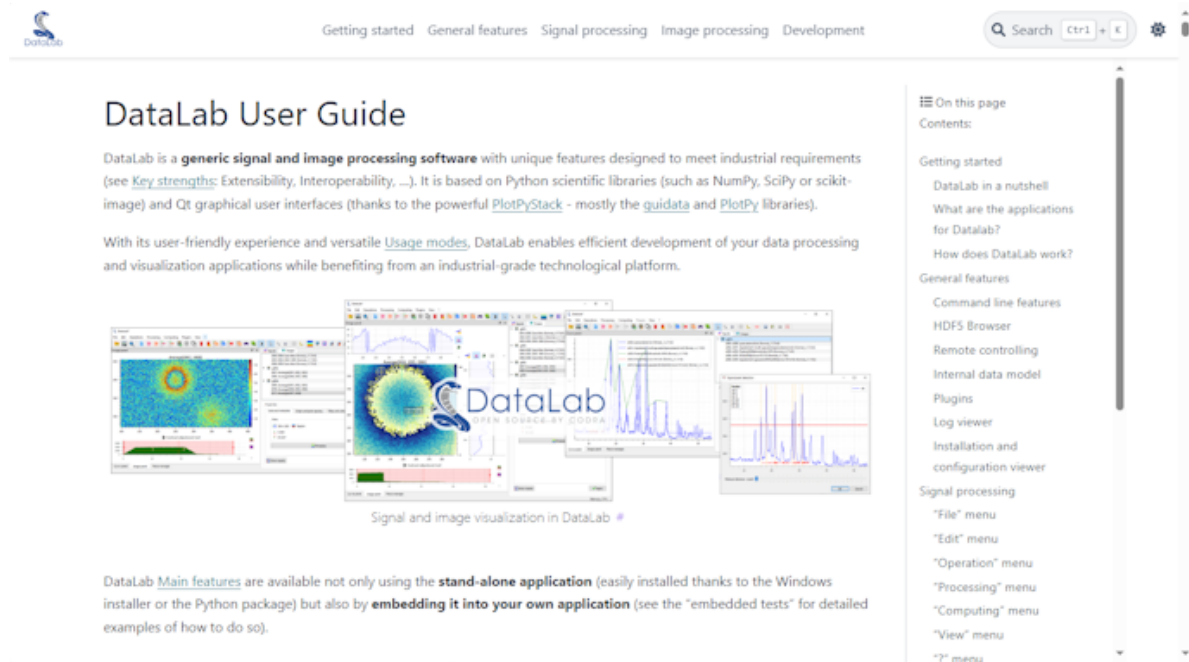
Show/hide panels or toolbars.

## 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

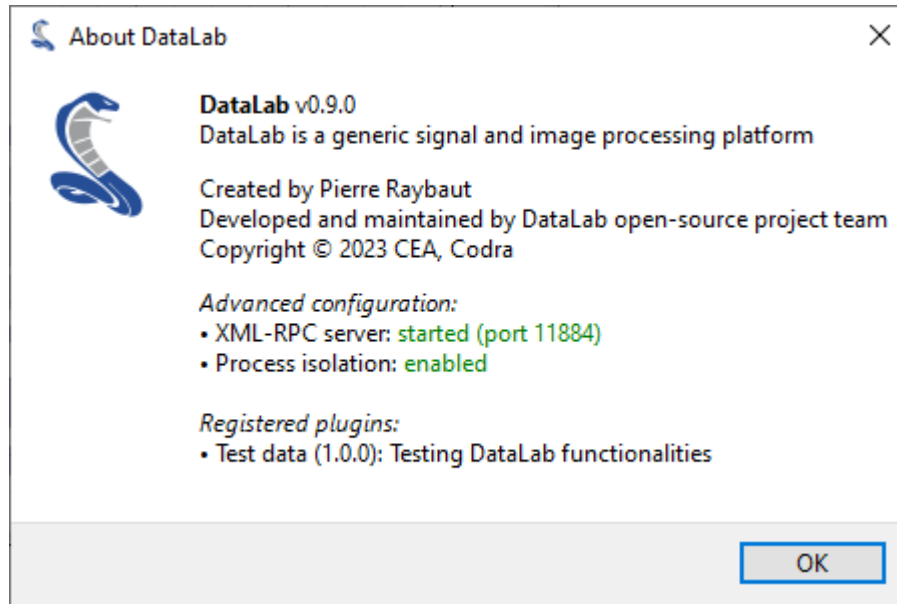
Show information regarding your DataLab installation (this is typically needed for submitting a bug report).

**See also:**

See [Installation and configuration viewer](#) for more details on this dialog box.

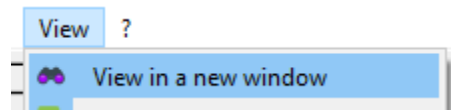
### About

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

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.

---

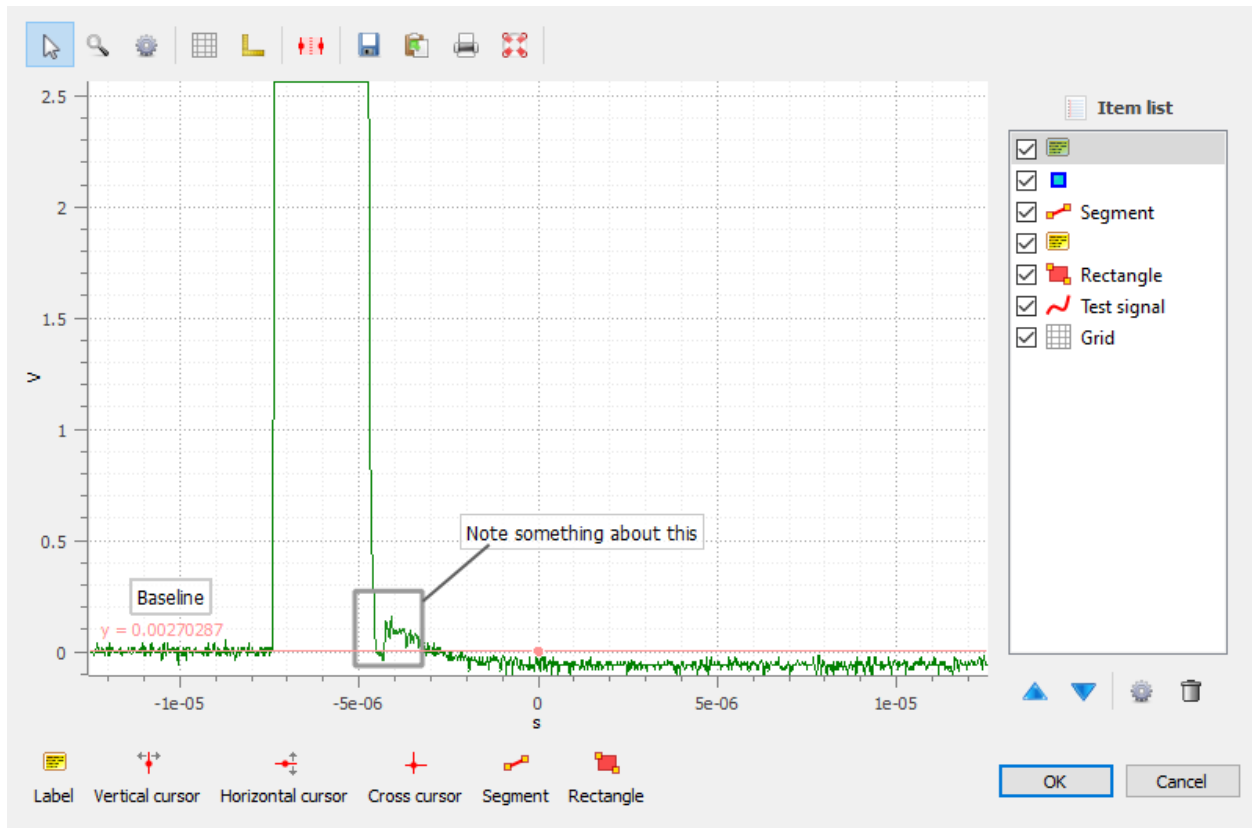
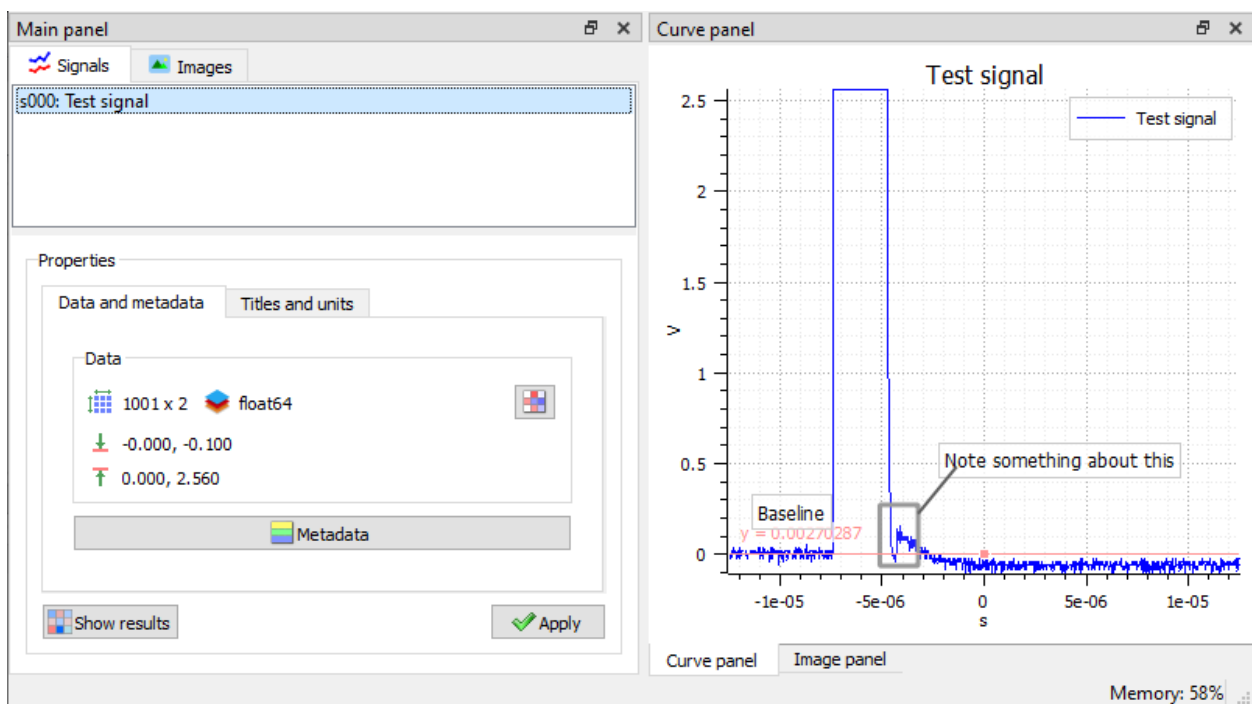


Fig. 7: Example of annotations.





## IMAGE PROCESSING

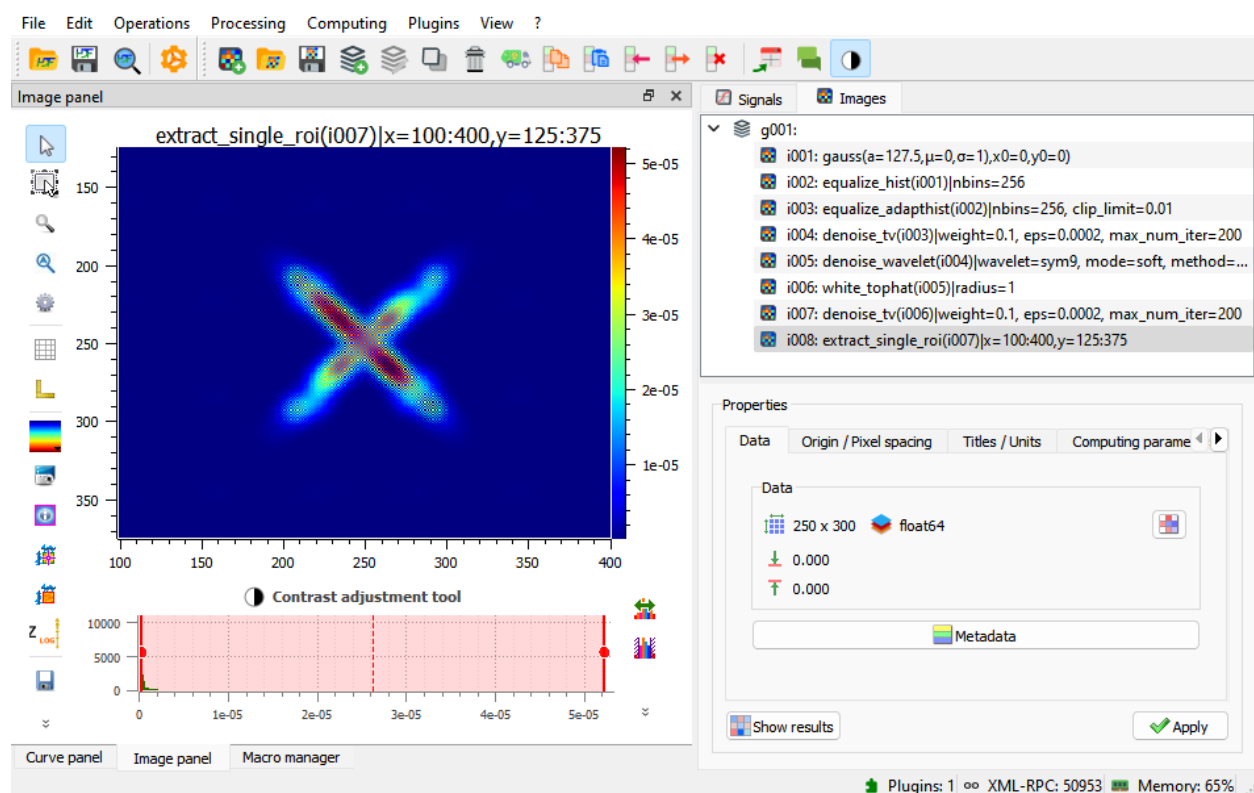
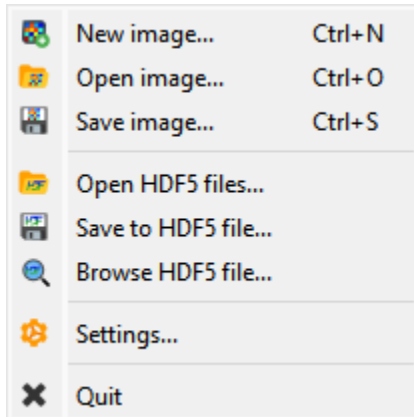


Fig. 1: DataLab main window: Image processing view

## 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	$z = A.exp(-\frac{(\sqrt{(x-x_0)^2 + (y-y_0)^2} - \mu)^2}{2\sigma^2})$

### Open image

Create a new image from the following supported filetypes:

File type	Extensions
PNG files	.png
TIFF files	.tif, .tiff
8-bit images	.jpg, .gif
NumPy arrays	.npy
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.

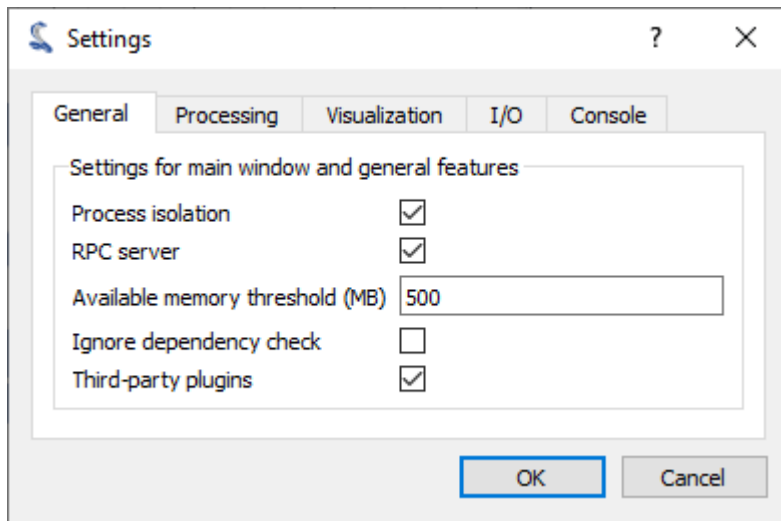
### Browse HDF5 file

Open the [HDF5 Browser](#) in a new window to browse and import data from HDF5 file.

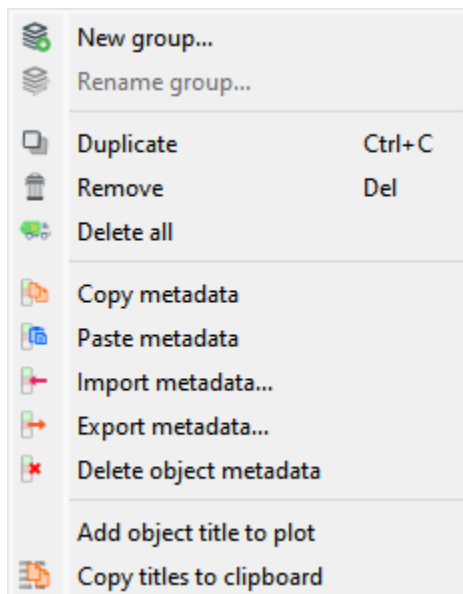


## Settings

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 additional information such as Region of Interest or results of computations

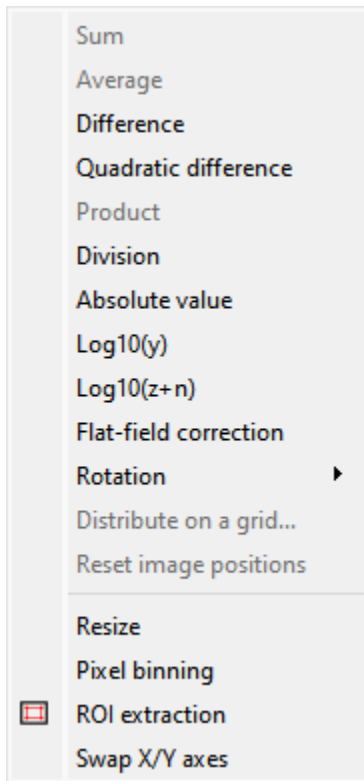
**Add object title to plot**

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:

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

**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:

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

where  $z_0$  is the raw image,  $z_f$  is the flat field image,  $z_{threshold}$  is an adjustable threshold and  $\overline{z_f}$  is the flat field image average value:

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

---

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

---

**Rotation**

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

**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.

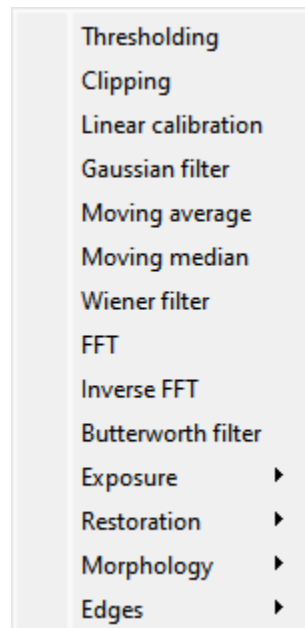
**ROI extraction**

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.

## 4.4 “Processing” menu



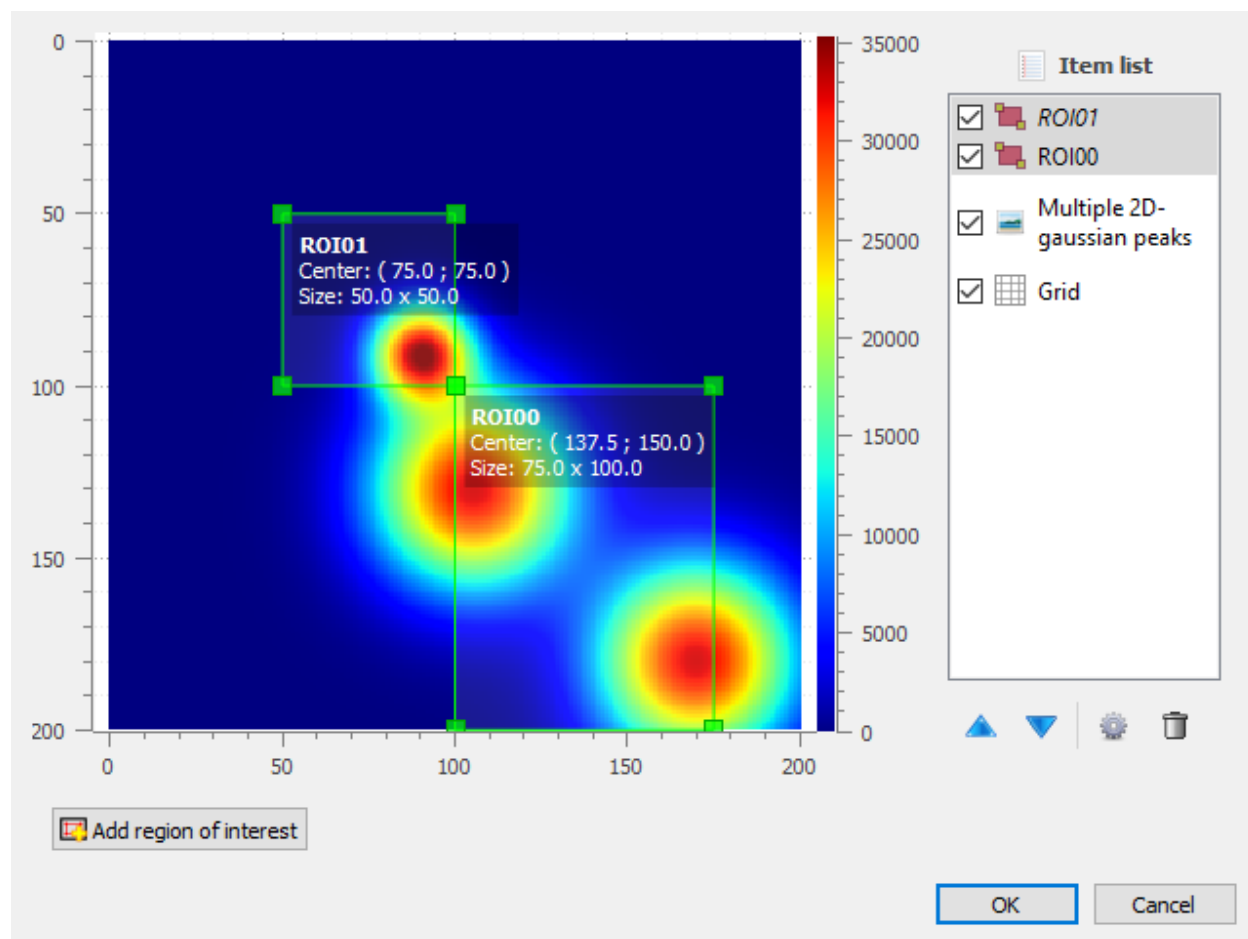


Fig. 2: ROI extraction dialog: the ROI is defined by moving the position and adjusting the size of a rectangle shape.

**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**

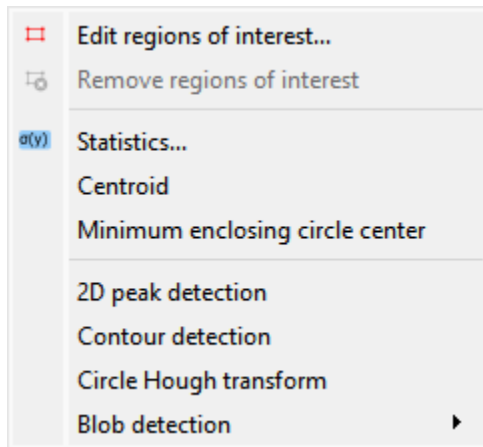
Perform all edge filtering algorithms (see above) on an image. Combined with the “distribute on a grid” option, this allows to compare the different edge filters on the same image.

**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).

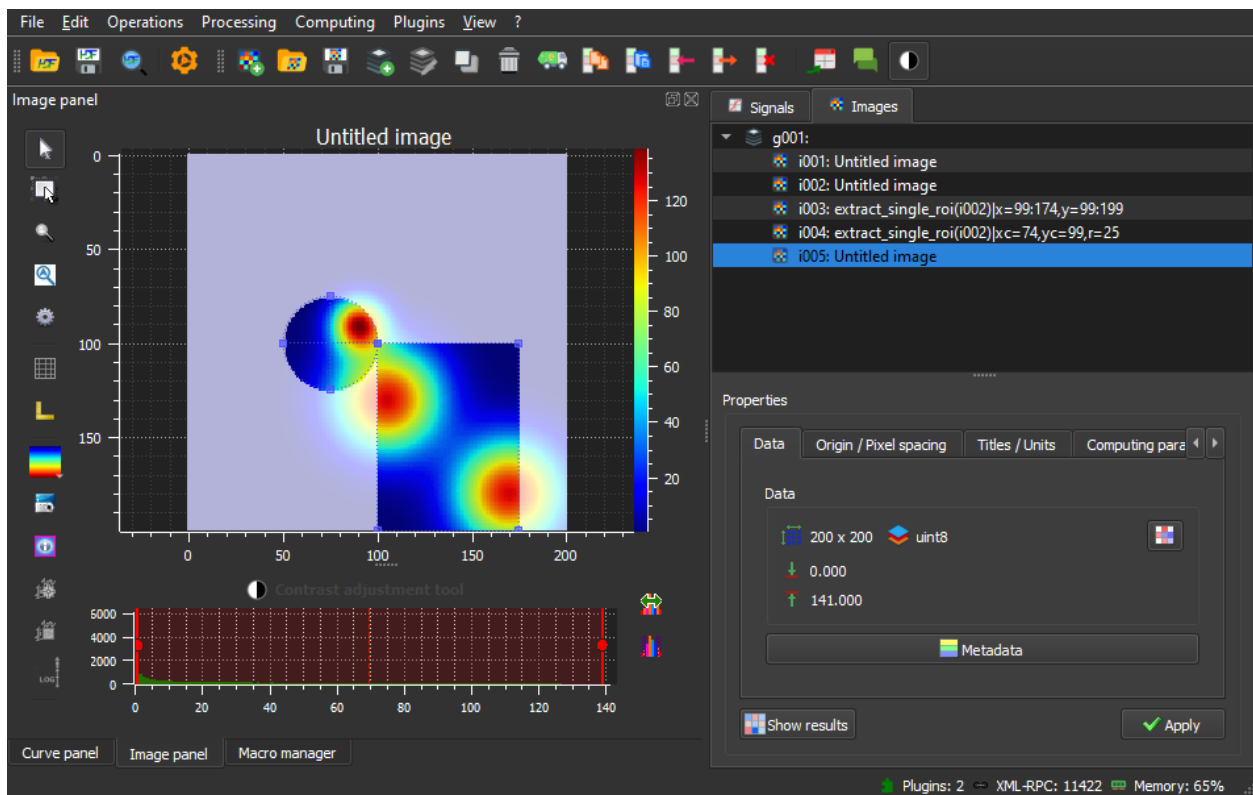


Fig. 3: An image with ROI.

### Remove regions of interest

Remove all defined ROI for selected object(s).

### Statistics

Compute statistics on selected image and show a summary table.

	min(z)	max(z)	<z>	$\sigma(z)$	$\Sigma(z)$	SNR(z)
i000	0	32754	512.558	2852.36	1.28139e+08	5.56494
i000 ROI00	0	32754	512.558	2852.36	6.40697e+07	5.56494
i000 ROI01	0	0	0	0	0	nan
i000 ROI02	0	32754	512.558	2852.36	3.20349e+07	5.56494

Format    Resize    ☒ Background color

Close

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

### 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.

#### 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](#)).

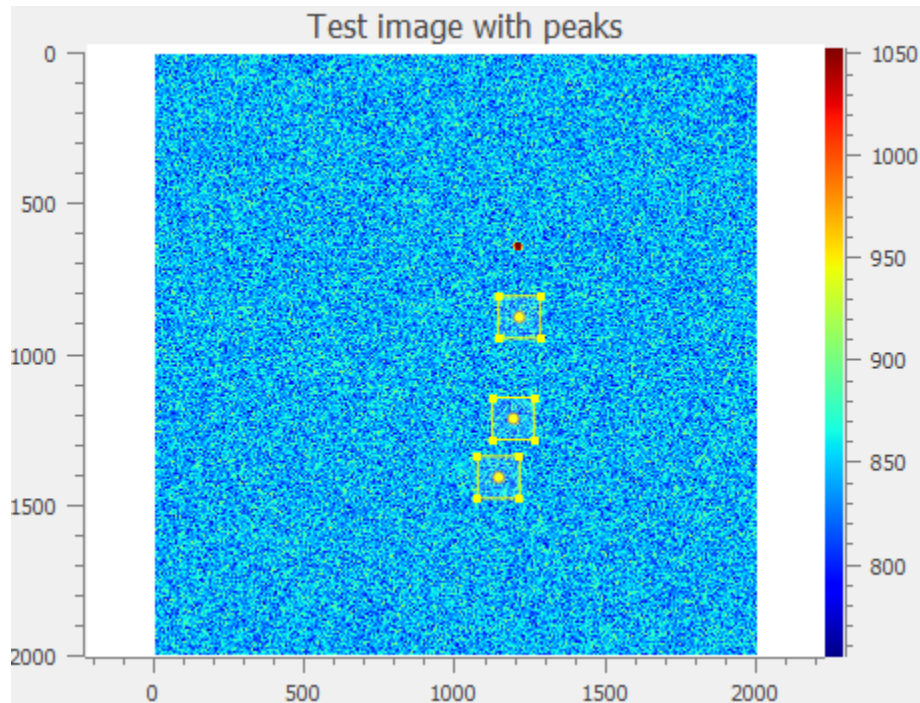


Fig. 5: Example of 2D peak detection.

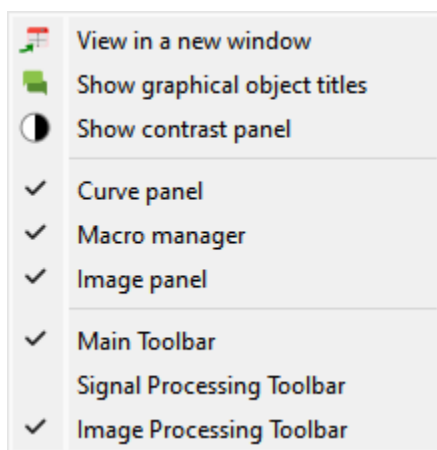
#### Blob detection (LOG)

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`.

## 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

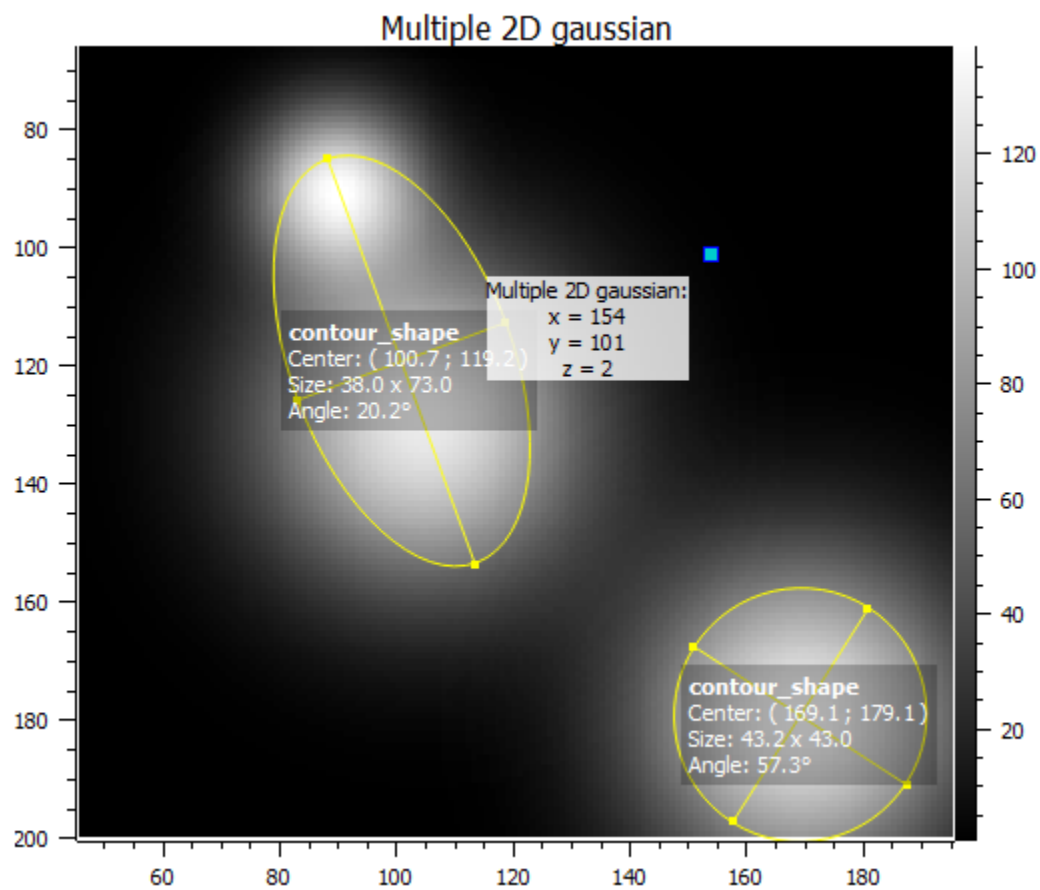


Fig. 6: Example of contour detection.

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.

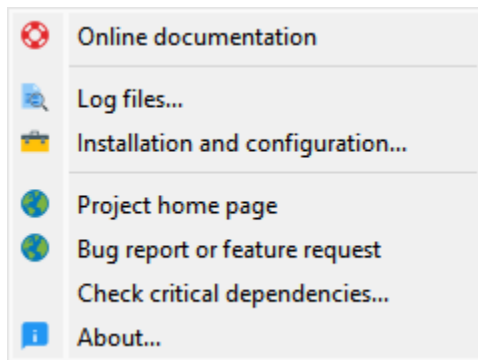
### Show contrast panel

Show/hide contrast adjustment panel.

### Other menu entries

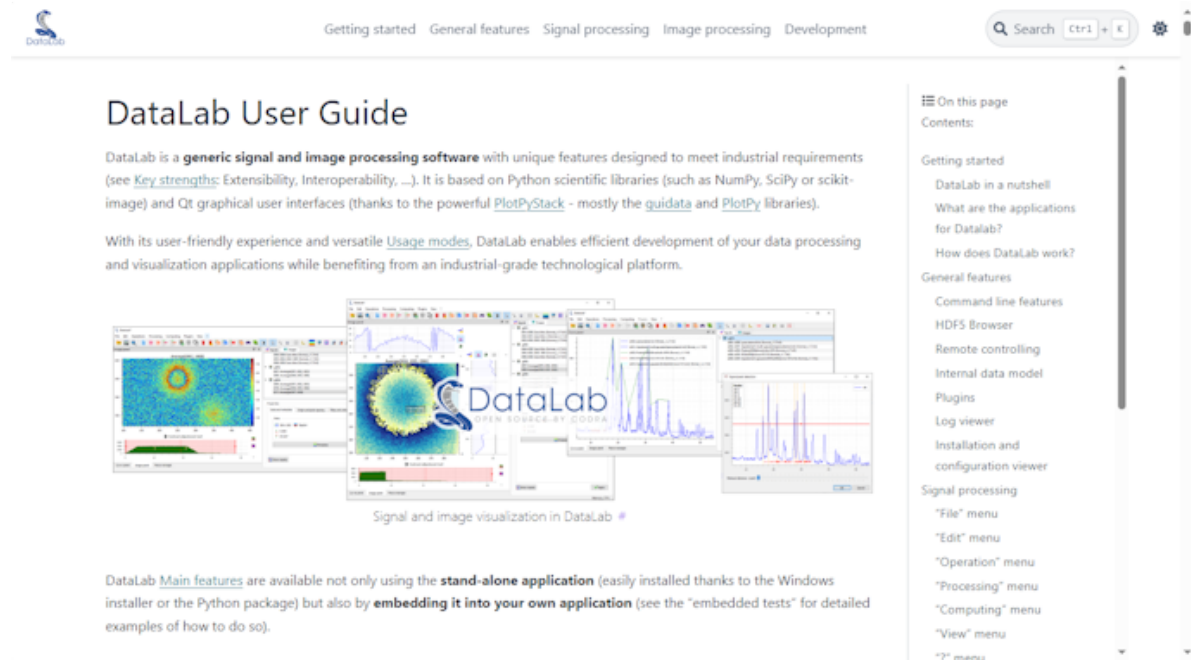
Show/hide panels or toolbars.

## 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

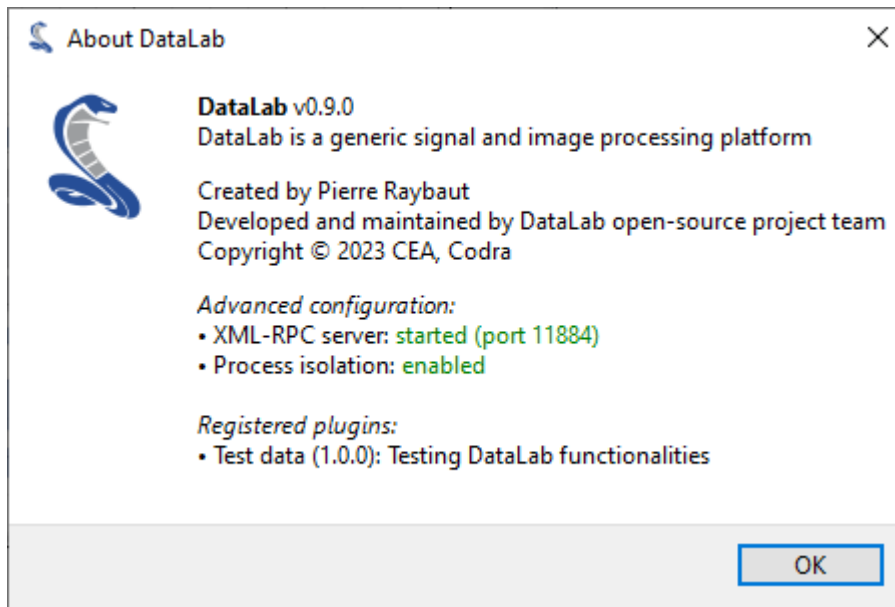
Show information regarding your DataLab installation (this is typically needed for submitting a bug report).

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.

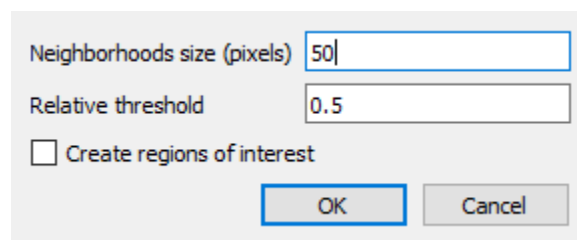


Fig. 7: 2D peak detection parameters.

### 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 threshold”

- Check “Create regions of interest” if you want a ROI defined for each detected peak (this may become useful when using another computation afterwards on each area around peaks, e.g. contour detection)



	ROI	x	y
Peaks(i000)	0	1366	638
Peaks(i000)	0	1416	1069
Peaks(i000)	0	1018	1135
Peaks(i000)	0	828	1229

Fig. 8: 2d peak detection results (see test “peak2d\_app.py”)

**Results are shown in a table:**

- Each row is associated to a detected peak
- First column shows the ROI index (0 if no ROI is defined on input image)
- Second and third columns show peak coordinates

**The 2d peak detection algorithm works in the following way:**

- First, the minimum and maximum filtered images are computed using a sliding window algorithm with a user-defined size (implementation based on `scipy.ndimage.minimum_filter` and `scipy.ndimage.maximum_filter`)
- Then, the difference between the maximum and minimum filtered images is clipped at a user-defined threshold
- Resulting image features are labeled using `scipy.ndimage.label`
- Peak coordinates are then obtained from labels center
- Duplicates are eventually removed

**The 2d peak detection parameters are the following:**

- “Neighborhoods size”: size of the sliding window (see above)
- “Relative threshold”: detection threshold

Feature is based on `get_2d_peaks_coords` function from `cdl.algorithms` module:

```
def get_2d_peaks_coords(
    data: np.ndarray, size: int | None = None, level: float = 0.5
) -> np.ndarray:
    """Detect peaks in image data, return coordinates.

    If neighborhoods size is None, default value is the highest value
```

(continues on next page)

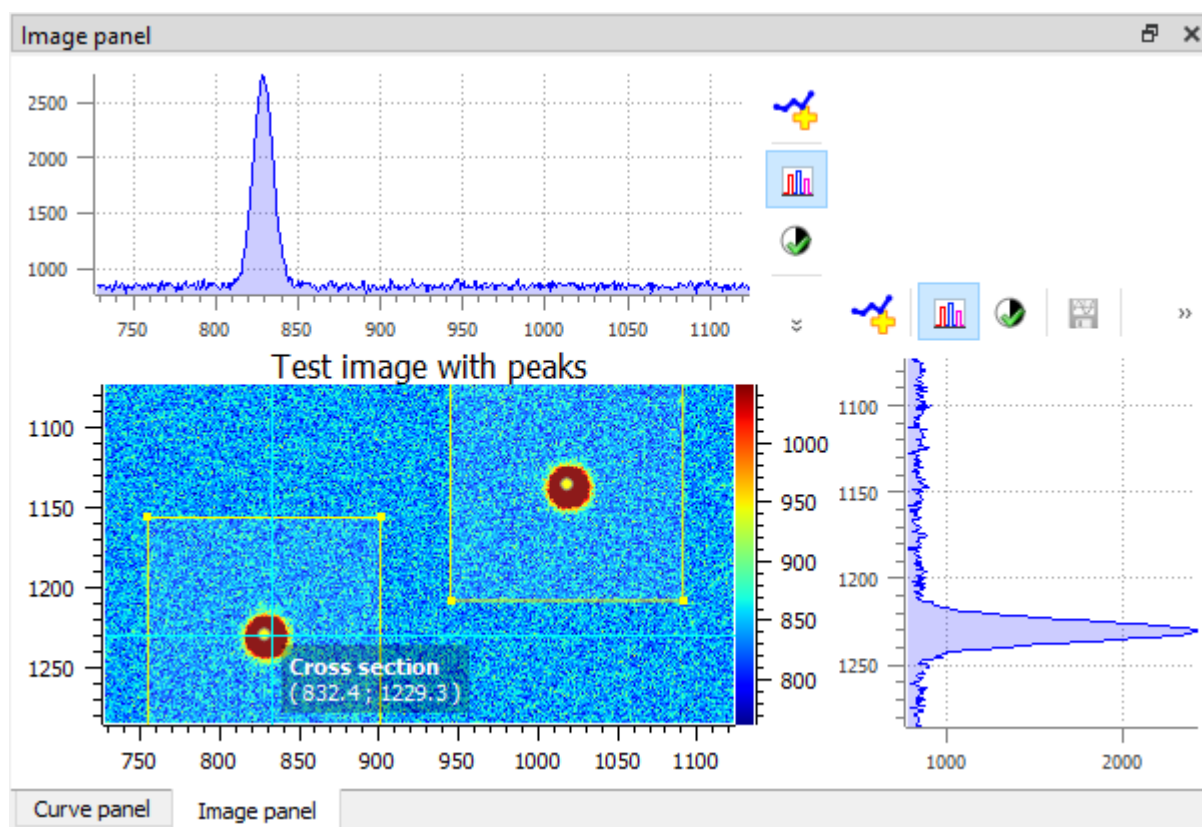


Fig. 9: Example of 2D peak detection.



(continued from previous page)

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

    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](#).

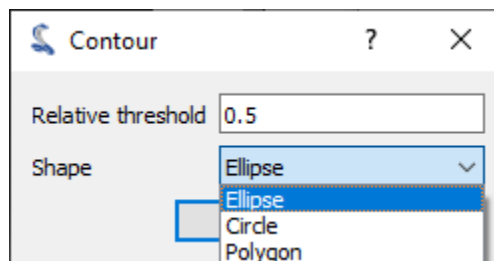
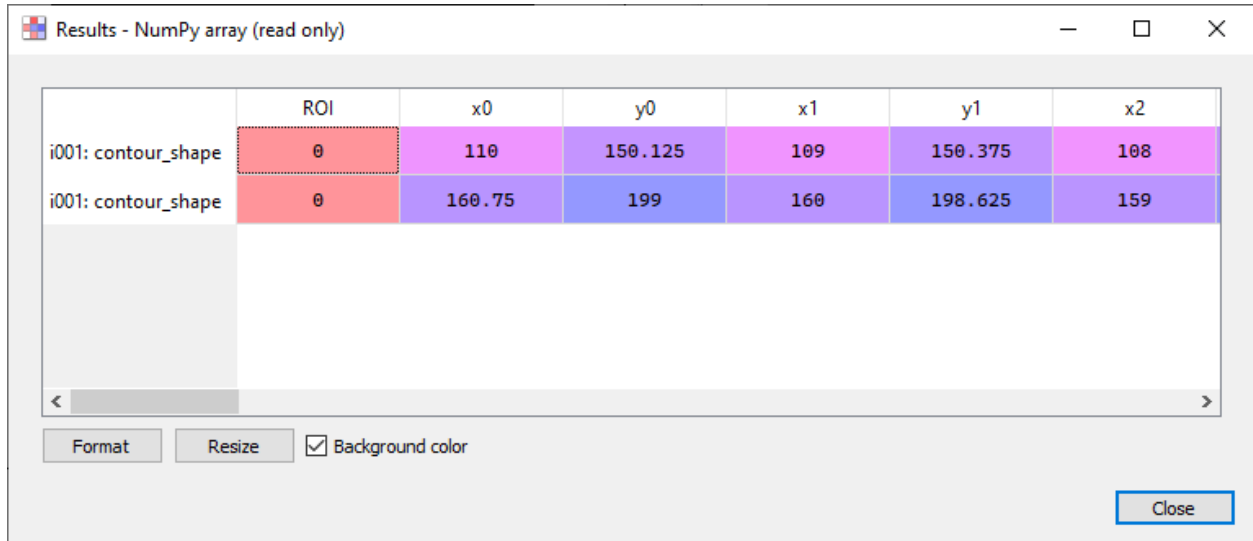


Fig. 10: Contour detection parameters.

**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”)



The screenshot shows a window titled "Results - NumPy array (read only)" with a table of contour detection results. The table has 7 columns: ROI, x0, y0, x1, y1, and x2. There are two rows of data, both labeled "i001: contour\_shape" in the first column. The first row has a red ROI cell with value 0, and the second row has a red ROI cell with value 0. The x0, y0, x1, y1, and x2 cells are colored purple.

	ROI	x0	y0	x1	y1	x2
i001: contour_shape	0	110	150.125	109	150.375	108
i001: contour_shape	0	160.75	199	160	198.625	159

Below the table are buttons for "Format", "Resize", and a checked checkbox for "Background color". A "Close" button is in the bottom right corner.

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)

**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:
    """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')
```

(continues on next page)

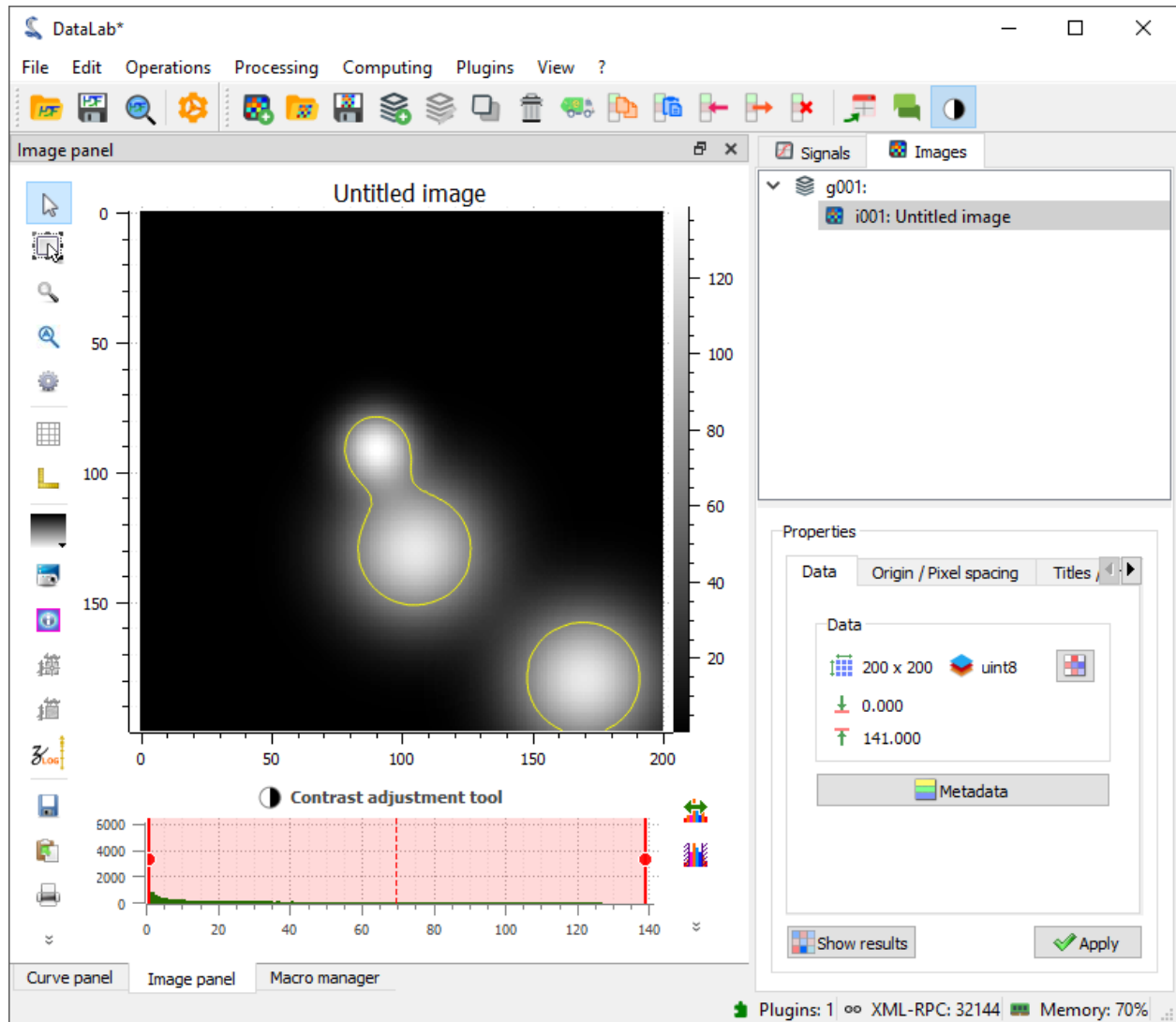


Fig. 12: Example of contour detection.

(continued from previous page)

```

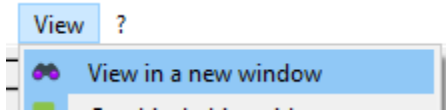
    level: Relative level (default: 0.5)

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):
            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.

---

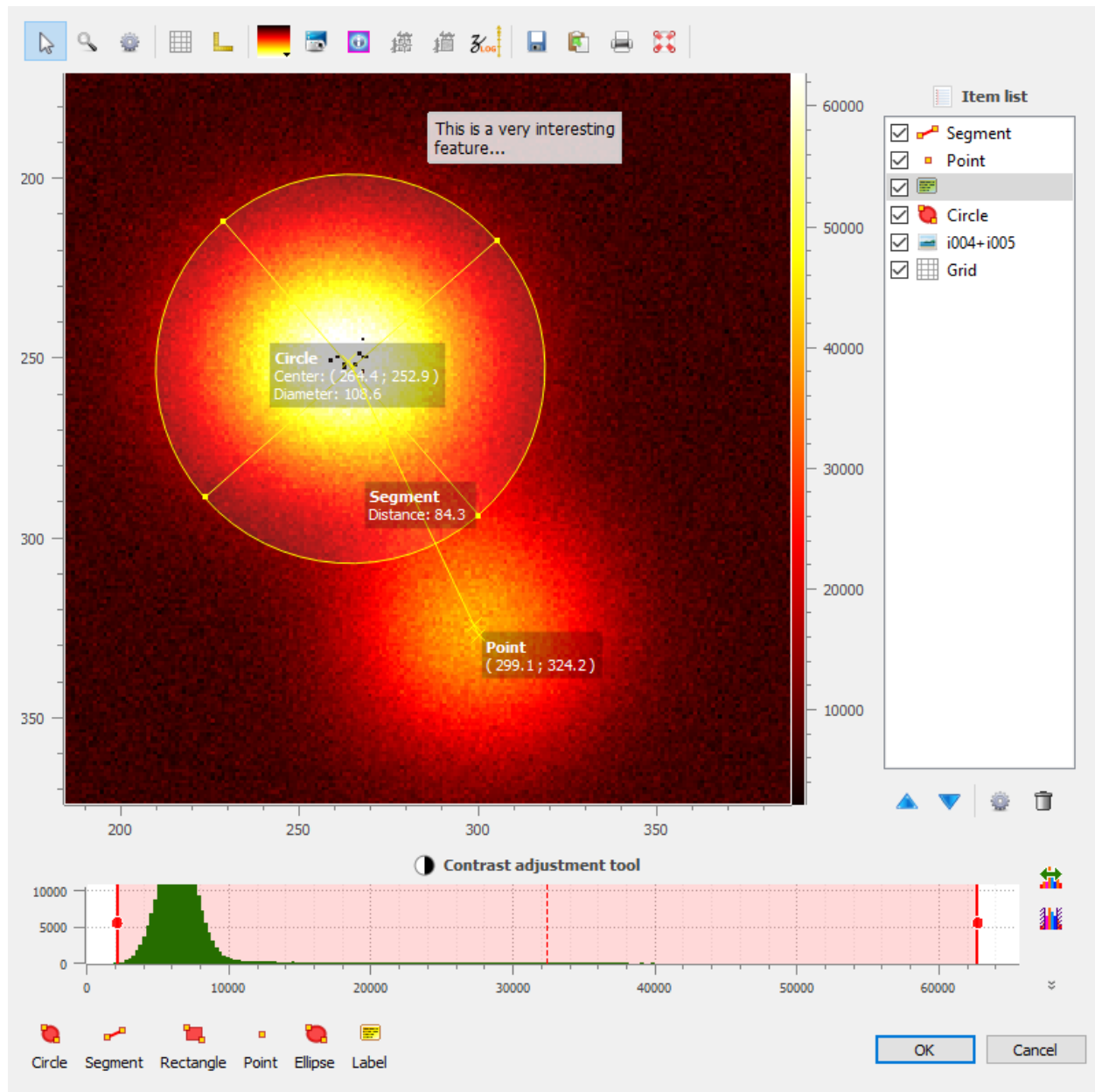
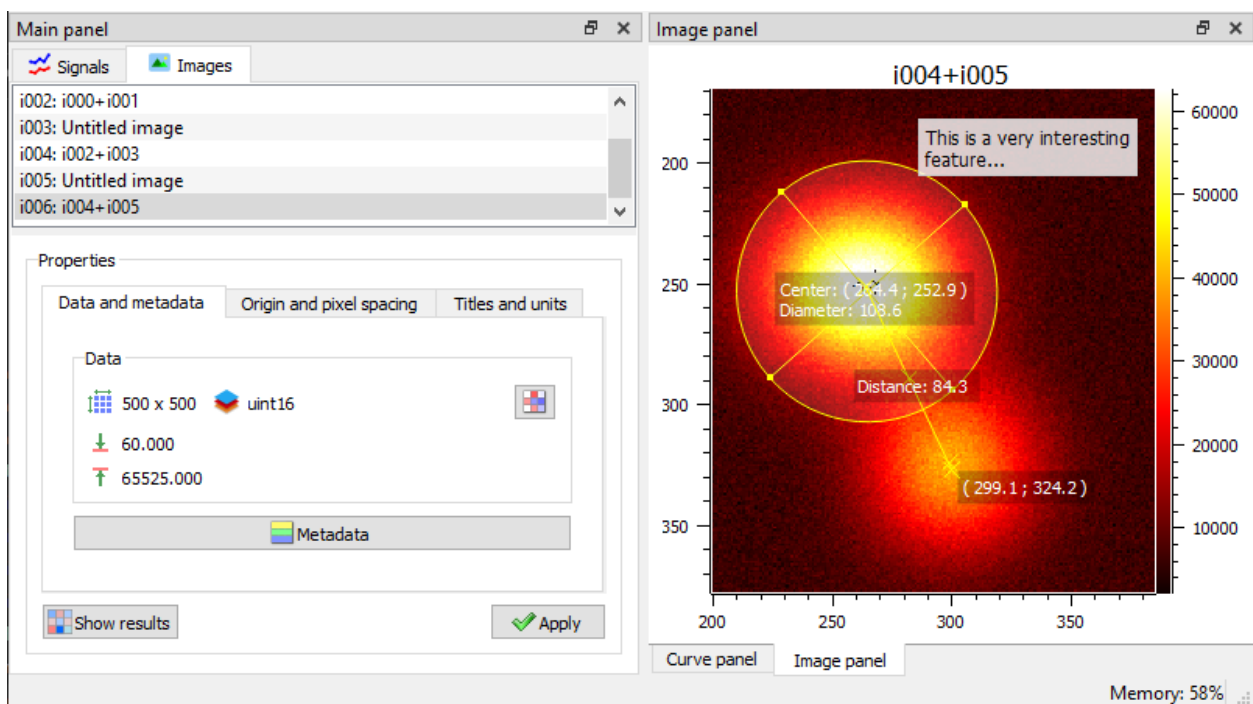


Fig. 13: Example of annotations.







## 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
<a href="#">Black Formatter</a>	Python code formatter
<a href="#">gettext</a>	Gettext syntax highlighting
<a href="#">isort</a>	Python import sorter
<a href="#">Pylance</a>	Python language server
<a href="#">Python</a>	Python extension
<a href="#">reStructuredText Syntax highlighting</a>	reStructuredText syntax highlighting
<a href="#">Ruff</a>	Extremely fast Python linter and code formatter
<a href="#">Todo Tree</a>	Todo tree

## 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:\\C20IQ-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
<a href="#">gettext</a>	Translations
<a href="#">Git</a>	Version control system
<a href="#">ImageMagick</a>	Image manipulation utilities
<a href="#">Inkscape</a>	Vector graphics editor
<a href="#">MiKTeX</a>	LaTeX distribution on Windows





## CHANGELOG

See DataLab [roadmap page](#) for future and past milestones.

### 6.1 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

## 6.2 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
    - \* Sigmoid correction
  - Added restoration processing features:
    - \* Total variation denoising filter (TV Chambolle)
    - \* Bilateral filter (denoising)
    - \* Wavelet denoising filter
    - \* White Top-Hat denoising filter
  - Added morphological transforms (disk footprint):
    - \* White Top-Hat
    - \* Black Top-Hat
    - \* Erosion
    - \* Dilation
    - \* Opening
    - \* Closing
  - Added edge detection features:
    - \* Roberts filter

- \* Prewitt filter (vertical, horizontal, both)
- \* Sobel filter (vertical, horizontal, both)
- \* Scharr filter (vertical, horizontal, both)
- \* Farid filter (vertical, horizontal, both)
- \* Laplace filter
- \* Canny filter
- Contour detection: added support for polygonal contours (in addition to circle and ellipse contours)
- Added circle Hough transform (circle detection)
- Added image intensity levels rescaling
- Added histogram equalization
- Added adaptative histogram equalization
- Added blob detection methods:
  - \* Difference of Gaussian
  - \* Determinant of Hessian method
  - \* Laplacian of Gaussian
  - \* Blob detection using OpenCV
- Result shapes and annotations are now transformed (instead of removed) when executing one of the following operations:
  - \* Rotation (arbitrary angle, +90°, -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

## 6.3 Older releases

### 6.3.1 CodraFT Version 2.2.0

New features:

- Images: added support for XYZ image files
- All shapes: removed shape drag symbols, so that background image is no longer masked by small-sized shapes
- At startup, restoring last current panel (image or signal panel)
- Plot cleanup and shape management: greatly optimized performance
- After removing object(s) (signal/image), the previous object in the list is selected
- Added default image visualization settings in .INI configuration file
- Using guiqwt v4.3.2: fixed pixel position (first pixel is centered at (0,0) coords)

### 6.3.2 CodraFT Version 2.1.4

Bug fixes:

- HDF5 import/browser features: added support for non-ASCII dataset names
- ANDOR SIF files:
  - Fixed compatibility issues for various SIF files
  - Fixed unicode error
- Image Contour detection:
  - Fixed level default value for 8-bit data
  - Added missing “level” parameter
- Dev/VSCode: simplified `launch.json` and fixed environment variable substitution issue

Other changes:

- Alpha/beta release: fixed installer, added warning

### 6.3.3 CodraFT Version 2.1.3

Bug fixes:

- Panel’s object list `select_rows` method: fixed plot refresh behavior in case of multiple selection (refresh widget only once)
- LMJ-formatted HDF5 file: now reading invalid compound datasets
- [Issue #16](#) - Embedding DataLab: “`add_object`” method call with invalid data should lead to app crash
  - Panel’s `add_object` method (public API): check data type before adding object to panel - this prevents DataLab from crashing when trying to plot invalid data type afterwards
  - Now handling exceptions in `add_object` and `insert_object` methods
- Multigaussian curve fitting: fixed default fit parameters
- Improved I/O application test with respect to unsupported filetypes

Other changes:

- Images: added support for `numpy.int32` datatype
- Added unit tests for all curve fitting dialogs

### 6.3.4 CodraFT Version 2.1.2

Bug fixes:

- [Pull Request #2](#) - Load / Save conventional CSVs, by [@aanastasiou](#)
- [Issue #3](#) - Wrong units/titles are displayed
- [Issue #6](#) - 2D peak detection: GUI freezes when creating ROIs
- [Issue #4](#) - Processing multiple images/signals: avoid unnecessary time-consuming plot updates
- [Issue #7](#) - Image/Circular ROI: `IndexError` when circle exceeds the image size
- [Issue #5](#) - ROI dialog box: unable to remove all ROIs and validate

- [Issue #8](#) - HDF5 import: unable to easily distinguish datasets with the same name but different path
- Average operation now merges ROI data (i.e. same behavior as sum)
- Fixed multiple regressions with ROI management (adding, removing ROI, ...)

Other changes:

- Optimized load time (especially for images): avoid unnecessary refresh when adding objects
- Added “Remove regions of interest” entry to “Computing” menu (and context menu)
- Signal/image list: added tooltip showing a summary of metadata values (e.g. when importing data from HDF5, this shows HDF5 filename and HDF5 dataset path) - [Issue #8](#)
- Dependencies hash check: feature is now OS-dependent (+ more explicit messages)
- Slightly improved test coverage

### 6.3.5 CodraFT Version 2.1.1

Changes:

- Image Regions Of Interest (ROI):
  - ROIs are now shown as masks (areas outside ROIs are shaded)
  - Added support for circular ROIs
  - ROIs now take into account pixel size (dx, dy) as well as origin (x0, y0)
- Signal and Image ROIs:
  - New default extract mode: creating as many signals/images as ROIs (each ROI is extracted into a single signal/image)
  - The old extract mode (single signal/image output) is still available and may be enabled using the new checkbox added in ROI extraction dialog box
- Image visualization:
  - Added “Show contrast panel” option in toolbar and view menu
  - By default, contrast panel is now visible
  - When multiple images are selected, the first image LUT range is applied to all
- “View in a new window”: now opens non-modal dialogs, thus allowing to visualize multiple signals or images in separate windows
- Added demo mode (from command line, simply run: `cdl-demo`)
- Command line option `-h5` is now a positional argument (`h5`)
- Added command line option `-b` (or `-h5browser`) to browse a HDF5 file at startup
- Added command line option `-version` to show DataLab version

Bug fixes:

- Image computations now takes into account origin (x0, y0), pixel size (dx, dy) as well as regions of interest (related features: centroid, enclosing circle, 2D peak detection and contour detection)
- Image ROI definition dialog: maximum rows and columns were erroneously truncated
- Centralized argument parsing in DataLab exec env object, thus avoiding conflicts

### 6.3.6 CodraFT Version 2.0.3

Bug fixes:

- Fixed `pen.setWidth` TypeError on Linux

Other changes:

- Added an option to ignore dependency check warning at startup
- Installation configuration viewer: added info on dependency check result
- Ignore when unable to save h5 in ima/sig test scenarios

### 6.3.7 CodraFT Version 2.0.2

The following major changes were introduced with DataLab V2:

- 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
- Segmentation fault and Python exception logging
- Customizable annotations for both signals and images

### 6.3.8 Release key features

- New data visualization and processing features:

Signal	Image	Feature
	•	Automatic 2D-peak detection
	•	Automatic contour extraction (circle/ellipse fit)
•	•	Multiple Regions of Interest (ROIs)
	•	User-defined annotations (labels and geometric shapes)
•	•	“Statistics” computing feature

- Automation of high-level processing features: added fully automated high-level test scenarios, and enhanced public API for embedding DataLab into a third-party application
- Test Driven Development with high quality standards (pylint score  $\geq 9.8/10$ , test coverage  $\geq 90\%$ )

### 6.3.9 Detailed feature list

New data visualization and processing features:

- Image:
  - New automatic image contour detection feature returning fitted circle/ellipse
  - New automatic 2D peak detection feature (optionally create ROIs)
- “View in a new window”: added customizable “Annotations” support for both signal and image panels - supports user-defined annotations (points, segments, circles, ellipses, labels,...) which are serialized in image metadata
- Added “Show graphical object titles” option in “View” menu to show or hide the title (or subtitle) of ROIs or any other graphical object
- Added support for **multiple** Regions of Interest (ROI):
  - All “Computing” menu features apply to multiple ROIs
  - Computation result arrays now contains ROI index (first column) and one row per ROI
  - ROI are merged when summing objects (signals or images)
  - ROI can be removed, modified or added at any time
- Added option “Show graphical object titles” (“View” menu) to show or hide ROI titles or any other geometrical shapes title (or subtitle)
- New computing “Statistics” feature showing a table with statistics on image/signal and eventually regions of interest (min, max, mean, standard deviation, sum, ...)

New general purpose features:

- Memory management:
  - New available memory indicator on main window status bar
  - New warning dialog box when trying to open/create data if available memory is below the “available\_memory\_threshold” defined in DataLab configuration file (default: 500MB)
- Error handling:
  - New integrated log file viewer
  - New warning dialog box at startup suggesting to view log files when logs were generated during last session
  - Logging segmentation faults in “.DataLab\_faulthandler.log”
  - Logging Python exceptions in “.DataLabL\_traceback.log”
- Signal/Image metadata:
  - New copy/paste feature: update object metadata from another one
  - New import/export feature: import-export object metadata (JSON text file) using the new “Import metadata into” / “Export metadata from” entries in “File” menu
- HDF5 browser feature: complete redesign (better compatibility, evolutive design, ...)
- Added support for multiple HDF5 files opening at once
- Added .DataLab.ini configuration file (user home directory):
  - New configuration file entry: current working directory
  - New configuration file entry: current main window size and position
  - New configuration file entry: embedded Python console enabled state



- New configuration file entry: available memory alarm threshold

New test-related features:

- Added non-interactive tests, opening the way for unit tests with better coverage
- Added “unattended” and “screenshot” execution modes respectively for testing and documentation purpose
- Added automated high-level test scenarios (signal and image processing)
- Tests are now splitted in two categories: unit tests (`*_unit.py`) and application tests (`*_app.py`).
- Added Coverage.py support
- Added “all\_tests.py” to run all tests in unattended mode

New dependencies:

- `scikit-image`
- `psutil`

Other changes (on existing features):

- Image and Signal:
  - Object properties panel: added data type information (feature refactored upstream to guidata)
  - New random signal/image: added support for both Normal and Uniform distributions
  - Operations “sum” and “average” now merge metadata results
  - Computed titles “s/i000” are now renamed after inserting/removing an object
  - Computing results (geometrical shapes: segment, circle, ellipse): numerical results are now automatically added to metadata (respectively: length, center and radius, center, a and b)
- Image:
  - Added support for image origin and pixel size
  - Flat field correction: added threshold parameter
  - “New image” now creates an image with the same data type as selected image
  - “New image” now supports uint16 data type
- Signal:
  - Peak detection: added minimal distance parameter
  - Fit dialog / plot: do auto scale at startup
  - Peak detection dialog: preselect horizontal cursor at startup
- `cdl.core.gui` code refactoring: added subpackage `core.gui.processor`
- Added “Browse HDF5” action to main window (“Open HDF5” now imports all data)

Bug fixes:

- HDF5 file import: converted bytes metadata to str
- Added h5py to requirements (setup.py)
- Plot: reintroduced pure white background in light mode (white background was removed unintentionally when introducing dark mode)
- Image:
  - “Clean-up data view” feature was accidently removing grid

- Fixed hard crash when trying to visualize images with NaNs (use case: result of any filter on `uint8` image)
- Fixed hard crash when using image Z-axis log scale on some images
- Fixed DICOM support
- Fixed hard crash in “to\_codraft” (cross section item with empty data)
- Fixed image visualization parameters update from metadata
- `MinEnclosingCircle`: fixed  $\sqrt{2}$  error
- Signal:
  - “Clean-up data view” feature was accidentally removing legend box and grid
  - Fixed integral (missing initial point)
  - Fixed plotting support for complex data
  - Fixed signal visualization parameters update from metadata

### 6.3.10 CodraFT Version 1.7.2

Bug fixes:

- Fixed unit test “app1\_test.py” (create a single `QApp`)
- Fixed progress bar cancel issues (when passing HDF5 files to `app.run` function)
- Fixed random hard crash when opening curve fitting dialog
- Fixed curve fitting dialog parenting
- ROI metadata is now removed (because potentially invalid) after performing a computation that changes X-axis or Y-axis data (e.g. ROI extraction, image flip, image rotation, etc.)
- Fixed image creation features (broken since major refactoring)

Other changes:

- Removed deprecated Qt `exec_` calls (replaced by `exec`)
- Added more infos on uninstaller registry keys
- Added documentation on key features

### 6.3.11 CodraFT Version 1.7.1

Added first page of documentation (there is a beginning to everything...).

Bug fixes:

- Cross section tool was working only on first image in item list
- Separate view was broken since major refactoring

### 6.3.12 CodraFT Version 1.7.0

New features:

- Python 3.8 is now the reference Python release
- Dropped Python 2 and PyQt 4 support
- Major code cleaning and refactoring
- Reorganized the whole code base
- Added more unit tests
- Added GUI-based test launcher
- Added isort/black code formatting
- Switched from cx\_Freeze to pyinstaller for generating the stand-alone version
- Improved pylint score up to 9.90/10 with strict quality criteria

### 6.3.13 CodraFT Version 1.6.0

New features:

- Added dependencies check on startup: warn the user if at least one dependency has been altered (i.e. the application has not been qualified in this context)
- Added py3compat (since QtPy is dropping Python 3 support)

### 6.3.14 CodraFT Version 1.5.0

New features:

- Sum, average, difference, multiplication: re-converting data to initial type.
- Now supporting PySide2/PyQt4/PyQt5 compatibility thanks to guidata >= v1.7.9 (using QtPy).
- Now supporting Python 3.9 and NumPy 1.20.

Bug fixes:

- Fixed cross section retrieval feature: in stand-alone mode, a new DataLab window was created (that is not the expected behavior).
- Fixed crash when enabling cross sections on main window (needs PythonQwt 0.9.2).
- Fixed ValueError when generating a 2D-gaussian image with floats.
- HDF5 file import feature:
  - Fixed unit processing (parsing) with Python 3.
  - Fixed critical bug when clicking on “Check all”.

### 6.3.15 CodraFT Version 1.4.4

New experimental features:

- Experimental support for PySide2/PyQt4/PyQt5 thanks to guidata  $\geq$  v1.7.9 (using QtPy).
- Experimental support for Python 3.9 and NumPy 1.20.

New minor features:

- ZAxisLogTool: update automatically Z-axis scale (+ showing real value)
- Added contrast test (following issues with “eliminate\_outliers”)

### 6.3.16 CodraFT Version 1.4.3

New minor features:

- New test script for global application test (test\_app.py).
- Improved DataLab launcher (app.py).

### 6.3.17 CodraFT Version 1.4.2

New minor features:

- LMJ-formatted HDF5 file import: tree widget item’s tooltip now shows item data “description”.

Bug fixes:

- Fixed runtime warnings when computing centroid coordinates on an image ROI filled with zeros.
- LMJ-formatted HDF5 file support: fixed truncated units.

### 6.3.18 CodraFT Version 1.4.1

Bug fixes:

- Fixed LMJ-formatted HDF5 files: strings are encoded in “latin-1” which is not the expected behavior (“utf-8” is the expected encoding for ensuring better compatibility).

### 6.3.19 CodraFT Version 1.4.0

New features:

- LMJ-formatted HDF5 file import: added support for axis units and labels.
- New curve style behavior (more readable): unselecting items by default, circling over curve colors when selecting multiple curve items.

Bug fixes:

- Fixed LMJ-formatted HDF5 file support in DataLab data import feature.

### 6.3.20 CodraFT Version 1.3.1

Bug fixes:

- Improved support for LMJ-formatted HDF5 files.
- Z-axis logscale feature: freeing memory when mode is off.
- CDLMainWindow.get\_instance: create instance if it doesn't already exist.
- to\_codraft: show DataLab main window on top, if not already visible.
- Patch/guiqwt.histogram: removing histogram curve (if necessary) when image item has been removed.

### 6.3.21 CodraFT Version 1.3.0

New features:

- Image computations: added "Smallest enclosing circle center" computation.
- Added support for FXD image file type.

Bug fixes:

- Fixed image levels "Log scale" feature for Python 3 compatibility.

### 6.3.22 CodraFT Version 1.2.2

New features:

- Added "Delete all" entry to "Edit" menu: this removes all objects (signals or images) from current view.
- Added an option "hide\_on\_close" to CDLMainWindow class constructor (default value is False): when set to True, DataLab main window will simply hide when "Close" button is clicked, which is the expected behavior when embedding DataLab in another application.

Bug fixes:

- The memory leak fix in app.py was accidentally commented before commit.

### 6.3.23 CodraFT Version 1.2.1

Bug fixes:

- When quitting DataLab, objects were not deleted: this was causing a memory leak when embedding DataLab in another Qt window.
- When canceling HDF5 import dialog box after selecting at least one signal or image, the progress bar was shown even if no data was being imported.
- When closing HDF5 import dialog box, preview signal/image widgets were not deleted, hence causing another memory leak.

### 6.3.24 CodraFT Version 1.2.0

New features:

- Added support for uint32 images (converting to int32 data)
- Added “Z-axis logarithmic scale” feature for image items (check out the new entries in standard image toolbar and context menu)
- Added “HDF5 I/O Toolbar” to avoid a frequently reported user confusion between HDF5 I/O icons and Signal/Image specific I/O icons (i.e. open and save actions)
- Cross-section panels are now configured to show only cross-section curves associated to the currently selected image (instead of showing all curves, including those associated to hidden images)
- Image subtraction: now handling integer underflow

Bug fixes:

- When “Clean up data view” option was enabled, image histogram was not updated properly when changing image selection (histogram was the sum of all images histograms).
- Changed default image levels histogram “eliminate outliers” value: .1% instead of 2% to avoid display bug for noise background images for example (i.e. images with high contrast and very narrow histogram levels)

### 6.3.25 CodraFT Version 1.1.2

Bug fixes:

- When the X/Y Cross Section widget is embedded into a main window other than DataLab’s, clicking on the “Process signal” button will send the signal to DataLab’s signal panel for further processing, as expected.

### 6.3.26 CodraFT Version 1.1.1

Bug fixes:

- Fixed a bug leading to “None” titles when importing signals/images from HDF5 files created outside DataLab.

### 6.3.27 CodraFT Version 1.1.0

New features:

- Added new icons.
- Images:
  - Added support for SPIRICON image files (single-frame support only).

Bug fixes:

- Fixed a critical bug when opening HDF5 file (bug from “guidata” package). Now guidata is patched inside DataLab to take into account the unusual/risky PyQt patch from Taurus package (PyQt API is set to 2 for QString objects and instead of raising an ImportError when importing QString from PyQt4.QtCore, QString still exists and is replaced by “str”...).
- Images:
  - Centroid feature: coordinates were mixed up in DataLab application.
- Signals:

- Curve fitting (gaussian and lorentzian): fixed amplitude initial value for automatic fitting feature
- FWHM and  $\text{FW1/e}^2$ : fixed amplitude computation for input fit parameters and output results

### 6.3.28 CodraFT Version 1.0.0

First release of CodraFT.

New features:

- Added support for both Python 3 and Python 2.7, and both PyQt5 and PyQt4.
- Added HDF5 file reading support, using a new HDF5 browser with embedded curve and image preview.
- Signal and Image:
  - Added menu “Computing” for computing scalar values from signals/images.
  - Added “ROI definition” for “Computing” features
  - Added absolute value operation.
  - Added 10 base logarithm operation.
  - Added moving average/median filtering feature.
- Images:
  - Added support for Andor SIF image files (support multiple frames).
  - Added centroid computing feature.
  - Added support for images containing NaN values.
- Signals:
  - Added FWHM computing feature (based on curve fitting)
  - Added Full Width at  $1/e^2$  computing feature (based on gaussian fitting)
  - Added derivative and integral computation features.
  - Added “lorentzian” and “Voigt” to “new signals” available.
- Added curve fitting feature supporting various models (polynomial, gaussian, lorentzian, Voigt and multi-gaussian). Computed fitting parameters are stored in signal’s metadata (a new dictionary item for the Signal objects)
- Edit menu: added a new “View in a new window” action
- Added standard keyboard shortcuts (new, open, copy, etc.)
- “New image”: added new 2D-gaussian creation feature
- Added a GUI-based ROI extraction feature for both signal and image views
- Added a pop-up dialog when double-clicking on a signal/image to allow visualizing things on a possibly large window
- Added a peak detection feature
- Added centroid coordinates in image statistics tool
- Added support for curve/image titles, axis labels and axis units (those can be modified through the editable form within the “Properties” groupbox)
- Added support for cross section extraction from the image widget to the signal tab ; the extracted curve’s title shows the associated coordinates

- Added deployment script for building self-consistent executable distribution using the cx\_Freeze tool
- Improved curve visual: background is now flat and white

Bug fixes:

- Console dockwidget is now created after the **View** menu so that it appears in it, as expected. It is now hidden by default.
- Improved curve visual when selected: instead of adding big black squares along a selected curve, the curve line is simply broader when selected.



## COPYRIGHTS AND LICENSING

- Copyright © 2023 [Codra, Pierre Raybaut](#)
- Licensed under the terms of the [BSD 3-Clause](#)



## PYTHON MODULE INDEX

### C

- `cdl.core.gui.processor.image`, 32
- `cdl.core.gui.processor.signal`, 31
- `cdl.core.model.image`, 42
- `cdl.core.model.signal`, 37
- `cdl.core.remote`, 22
- `cdl.obj`, 37
- `cdl.param`, 37
- `cdl.plugins`, 56