



DataLab
Version 0.10.0

déc. 22, 2023

Table des matières :

1	Débuter	3
1.1	DataLab en bref	3
1.2	Quelles sont les applications de DataLab ?	3
1.3	Comment fonctionne DataLab ?	4
2	Fonctionnalités générales	11
2.1	Ligne de commande	12
2.2	Explorateur HDF5	15
2.3	Contrôle à distance	15
2.4	Modèle de données interne	34
2.5	Plugins	45
2.6	Journaux de bord	55
2.7	Installation et configuration	55
3	Traitement du signal	57
3.1	Menu « Fichier »	58
3.2	Menu « Edition »	59
3.3	Menu « Opérations »	60
3.4	Menu « Traitement »	63
3.5	Menu « Calculs »	64
3.6	Menu « Affichage »	66
3.7	Menu « ? »	67
3.8	Annotations (Signaux)	69
4	Traitement d'image	71
4.1	Menu « Fichier »	72
4.2	Menu « Edition »	73
4.3	Menu « Opérations »	74
4.4	Menu « Traitement »	78
4.5	Menu « Calculs »	81
4.6	Menu « Affichage »	85
4.7	Menu « ? »	86
4.8	Détection de pics 2D	87
4.9	Détection de contours	90
4.10	Annotations (Images)	93
5	Développement	97

5.1	Roadmap	97
5.2	Comment contribuer	99
5.3	Règles de codage	100
5.4	Mise en place de l'environnement de développement	101
6	Changelog	105
6.1	DataLab Version 0.10.0	105
6.2	DataLab Version 0.9.2	107
6.3	DataLab Version 0.9.1	107
6.4	DataLab Version 0.9.0	108
7	Licence et copyrights	111
	Index des modules Python	113

DataLab est un **logiciel générique de traitement du signal et de l'image** avec des fonctionnalités uniques conçues pour répondre à des niveaux d'exigence industriels (cf. *Points forts* : Extensibilité, Interopérabilité, ...). Il est basé sur des bibliothèques scientifiques Python (telles que NumPy, SciPy ou encore scikit-image) et sur les interfaces graphiques Qt (grâce à la puissante pile logicielle *PlotPyStack* - principalement aux bibliothèques *guidata* et *PlotPy*).

Avec son expérience utilisateur conviviale et ses *Modes d'utilisation* polyvalents, DataLab permet le développement efficace de vos applications de traitement et de visualisation de données tout en bénéficiant d'une plateforme technologique de qualité industrielle.

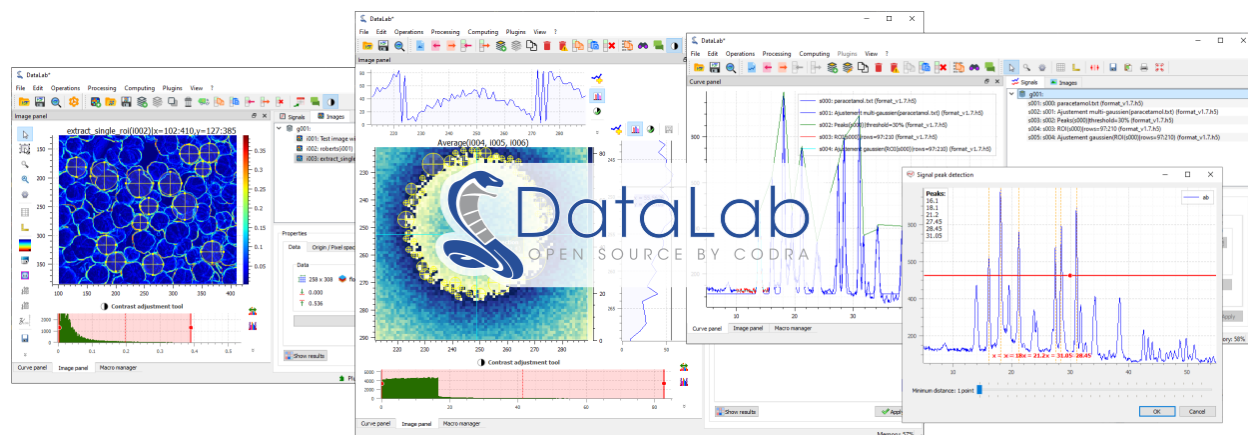


FIG. 1 – Visualisation de signaux et d'images dans DataLab

Les *Principales fonctionnalités* de DataLab sont disponibles non seulement en utilisant l'**application autonome** (facilement installée grâce à l'installateur Windows ou au paquet Python) mais aussi en l'**intégrant dans votre propre application** (voir les « embedded tests » pour des exemples détaillés).



FIG. 2 – DataLab est propulsé par *PlotPyStack*, les outils de visualisation et d'interface graphique scientifiques Python-Qt.

Note : DataLab a été créé par *Codra/Pierre Raybaut* en 2023. Il est développé et maintenu par l'équipe du projet open-source DataLab avec le soutien de *Codra*.

Ressources externes :

Home	Site de DataLab
PyPI	Paquet Python officiel
GitHub	Signalement d'anomalies et demandes d'évolutions

1.1 DataLab en bref

DataLab est une plateforme ouverte de traitement de signaux et d'images. Son périmètre fonctionnel est volontairement large. Avec ses nombreuses fonctions, certaines techniquement avancées, DataLab permet le traitement et la visualisation de tous types de données scientifiques. Ainsi, les acteurs scientifiques, industriels et de l'innovation peuvent disposer d'un outil facile à utiliser, simple à adapter et offrant la fiabilité d'un logiciel industriel.

1.2 Quelles sont les applications de DataLab ?

1.2.1 Exemples concrets

Quelques exemples concrets et spécifiques illustrent la nature des travaux pouvant être réalisés avec DataLab :

- Traitement de signaux expérimentaux acquis sur une installation scientifique
- Détection automatique des positions des tâches laser dans une scène
- Alignement d'instrument par traitement d'image
- Détection automatique de motifs et corrections géométriques

1.2.2 Modes d'utilisation

Selon l'application, DataLab peut être utilisé selon trois modes différents :

- **Mode autonome** : DataLab est une application de traitement à part entière qui peut être adaptée aux besoins du client par l'ajout de plugins spécifiques à son métier.
- **Mode embarqué** : DataLab est intégré dans votre application pour apporter les fonctionnalités de traitement et de visualisation nécessaires.
- **Mode commandé à distance** : DataLab communique avec votre application, lui permettant de bénéficier de ses fonctionnalités sans perturber l'expérience utilisateur.

Note : DataLab peut également être piloté depuis votre environnement de développement familial (par exemple Visual Studio Code, Spyder, ...) pour réaliser des calculs en utilisant vos fonctions de traitement tout en bénéficiant des fonctionnalités avancées de DataLab.

Avec son expérience utilisateur conviviale et ses modes d'utilisation polyvalents, DataLab permet un développement efficace de vos applications de traitement et de visualisation de données tout en bénéficiant d'une plateforme technologique industrielle.

1.3 Comment fonctionne DataLab ?

DataLab est une plateforme de traitement et de visualisation de données (signaux ou images) qui intègre de nombreuses fonctions. Développé en Python, il bénéficie de la richesse de l'écosystème associé en termes de bibliothèques scientifiques et techniques.

1.3.1 Principales fonctionnalités

Les principales fonctionnalités techniques de DataLab sont :

- Prise en charge de nombreux formats de données standards et propriétaires
- Ouverture d'un nombre arbitraire d'objets (signaux ou images) pour un traitement par lot, avec possibilité de définir des groupes d'objets
- Visualisation simultanée de plusieurs objets avec prise en charge des annotations
- Opérations et traitements standards sur les signaux et les images
- Traitement d'image avancé (restauration, morphologie, détection de contours, etc.)
- Gestion de plusieurs régions d'intérêt (calculs, extractions)
- Éditeur de macro-commandes
- API pilotable à distance
- Console Python interactive embarquée

1.3.2 Points forts

DataLab met en avant quatre points forts :

1. **Extensibilité** : Le système de plugins de DataLab permet de coder facilement de nouvelles fonctionnalités (traitement spécifique, formats de fichiers spécifiques, interfaces graphiques personnalisées). Il peut également être utilisé comme une plateforme personnalisable.
2. **Interopérabilité** : DataLab peut également être intégré dans votre propre application. Par exemple, au sein de logiciels de traitement de données, de systèmes de contrôle de niveau machine ou d'applications de banc d'essai.
3. **Automatisation** : une API publique de haut niveau permet de piloter à distance DataLab pour ouvrir et traiter des données.
4. **Maintenabilité et testabilité** : DataLab est un logiciel de traitement scientifique et technique industriel. Les tests automatisés intégrés à DataLab couvrent 90 % de ses fonctionnalités, ce qui est significatif pour un logiciel avec des interfaces graphiques et permet de réduire les risques de régression.

Chercheurs, ingénieurs, scientifiques, vous bénéficierez sans aucun doute des capacités de DataLab. Son modèle de logiciel open source vous permettra également de réinvestir vos réalisations dans la communauté open source, dont tout éditeur de renom devrait être un membre actif.

Installation

Dépendances

Note : L'installateur Windows de DataLab inclut déjà toutes les dépendances, ainsi que Python lui-même.

Le paquet cd1 requiert les modules Python suivants :

Nom	Version	Description
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 :

Nom	Version	Description
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 :

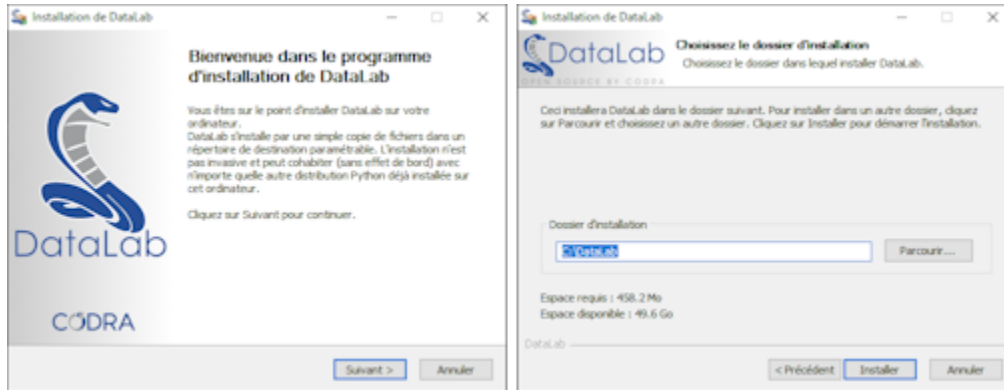
Nom	Version	Description
PyQt5		Python bindings for the Qt cross platform application toolkit
sphinx		Python documentation generator
sphinx_intl		Sphinx utility that make it easy to translate and to apply translation.
myst_parser		An extended [CommonMark](https://spec.commonmark.org/) compliant parser,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

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

How to install

Windows installer :

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



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

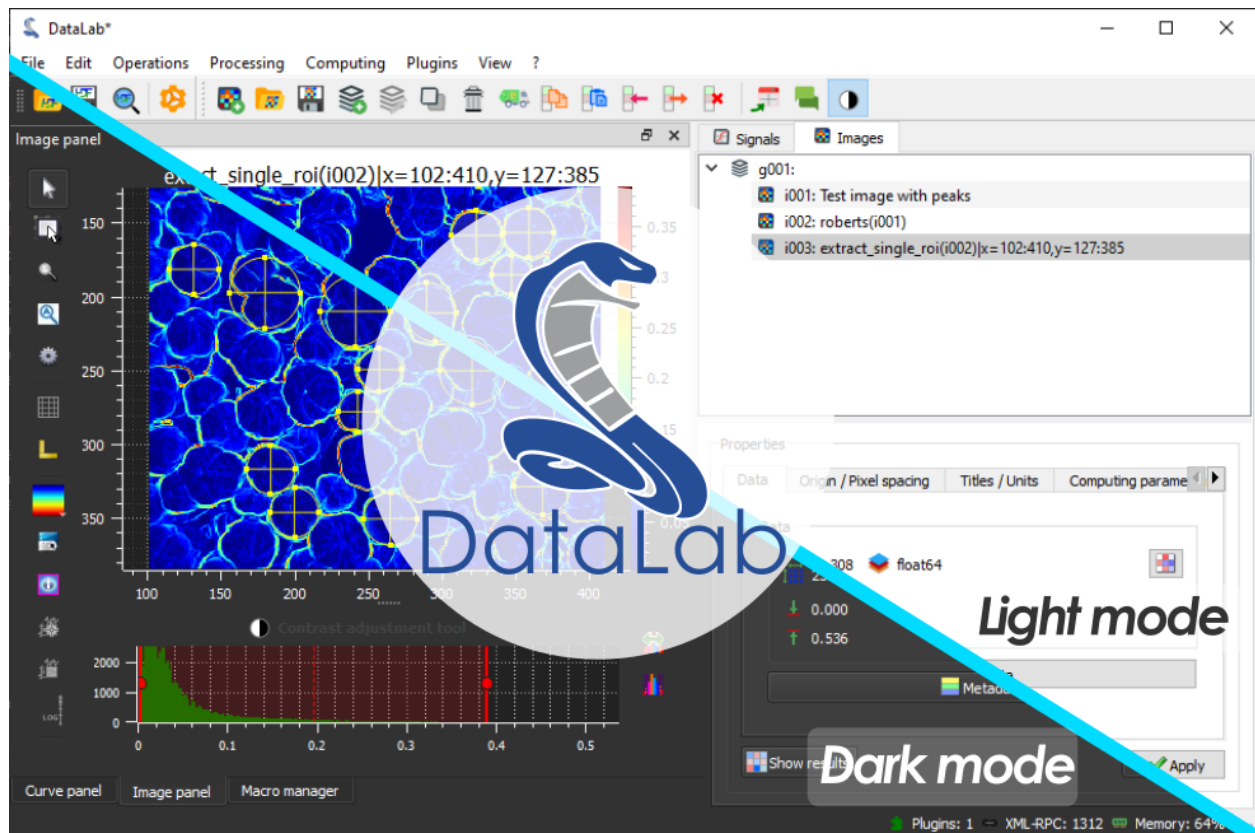


FIG. 1 – DataLab supports dark and light mode depending on your OS settings

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

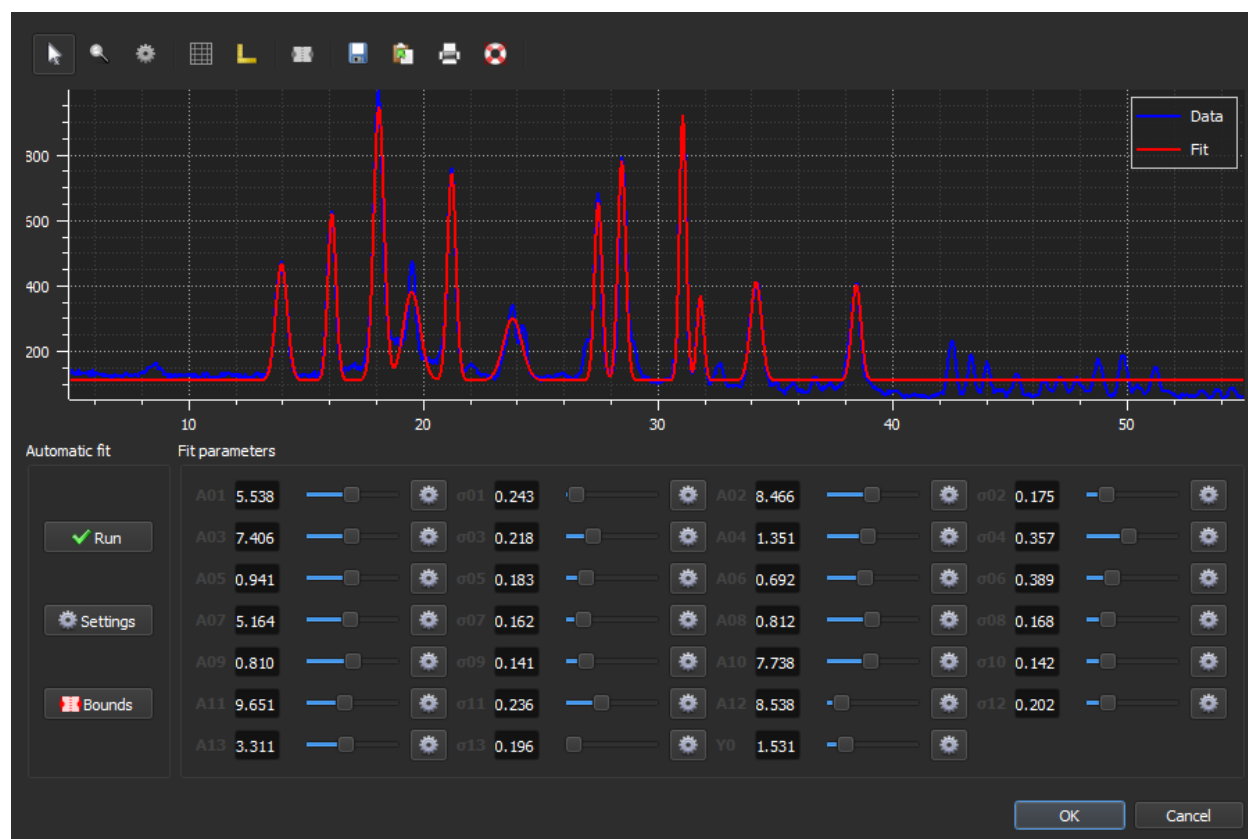


FIG. 2 – Example of a multi-gaussian curve fit

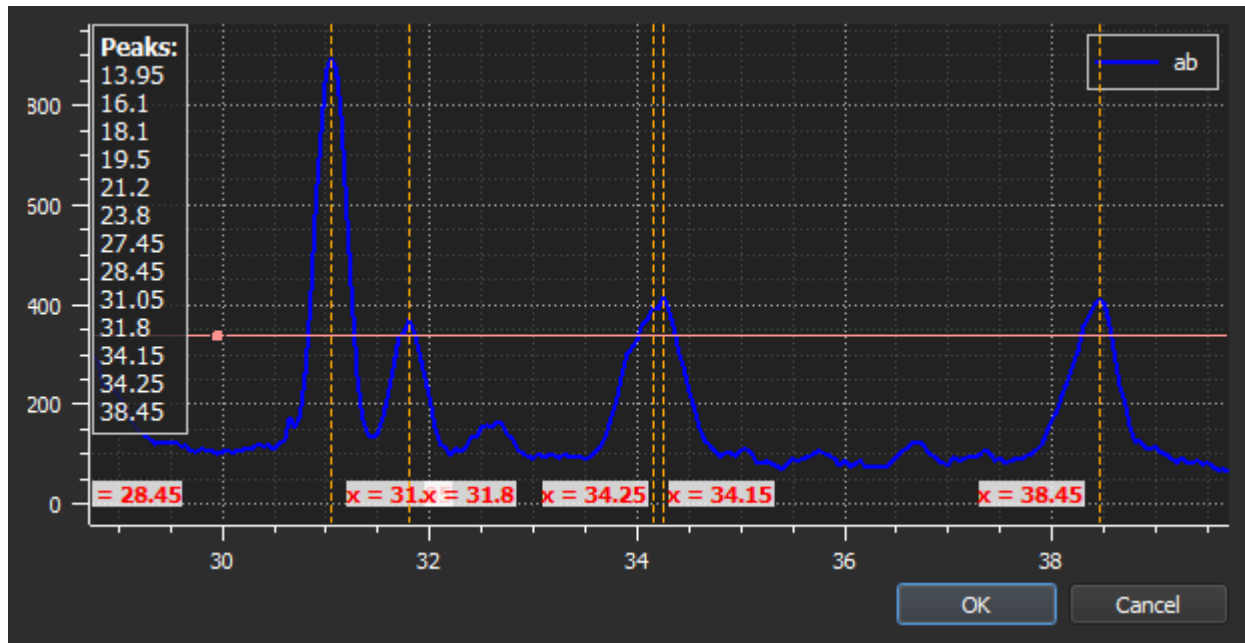


FIG. 3 – Semi-automatic peak detection

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

CHAPITRE 2

Fonctionnalités générales

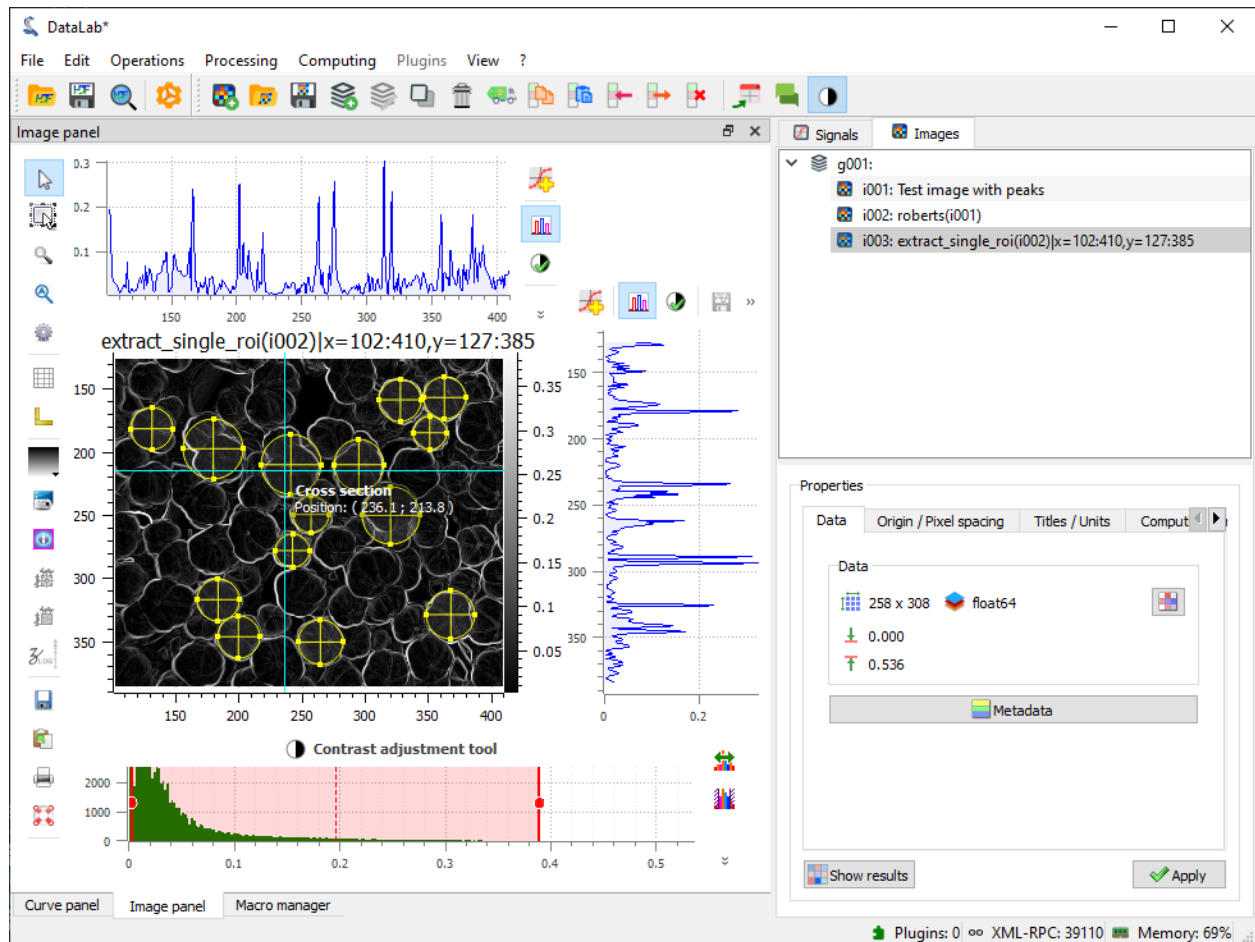


FIG. 1 – Fenêtre principale de DataLab

2.1 Ligne de commande

2.1.1 Exécuter DataLab

Pour exécuter DataLab depuis la ligne de commande, taper la commande suivante :

```
$ cdl
```

Pour afficher l'aide de l'utilisation en ligne de commande, taper simplement :

```
$ 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 Ouvrir un fichier HDF5 au démarrage

Pour ouvrir des fichiers HDF5, en important éventuellement un dataset précis du fichier HDF5, utiliser l'une des commandes suivantes :

```
$ 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 Ouvrir l'explorateur de fichiers HDF5 au démarrage

Pour ouvrir l'explorateur de fichiers HDF5 au démarrage, utiliser l'une des commandes suivantes :

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

2.1.4 Mode démonstration de DataLab

Pour exécuter le mode de démonstration de DataLab, taper la commande suivante :

```
$ cdl-demo
```

2.1.5 Exécuter les tests unitaires

Note : Cette suite de tests est basée sur le mécanisme de découverte de *guidata.guitest*. Elle n'est pas compatible avec *pytest* car la plupart des tests de haut niveau doivent être exécutés dans un processus séparé (par exemple, les tests de scénario échoueront s'ils sont exécutés dans le même processus que les autres tests).

Pour exécuter l'ensemble des tests unitaires de DataLab, taper la commande suivante :

```
$ 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=.
```

(suite sur la page suivante)

DEBUG=

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

```

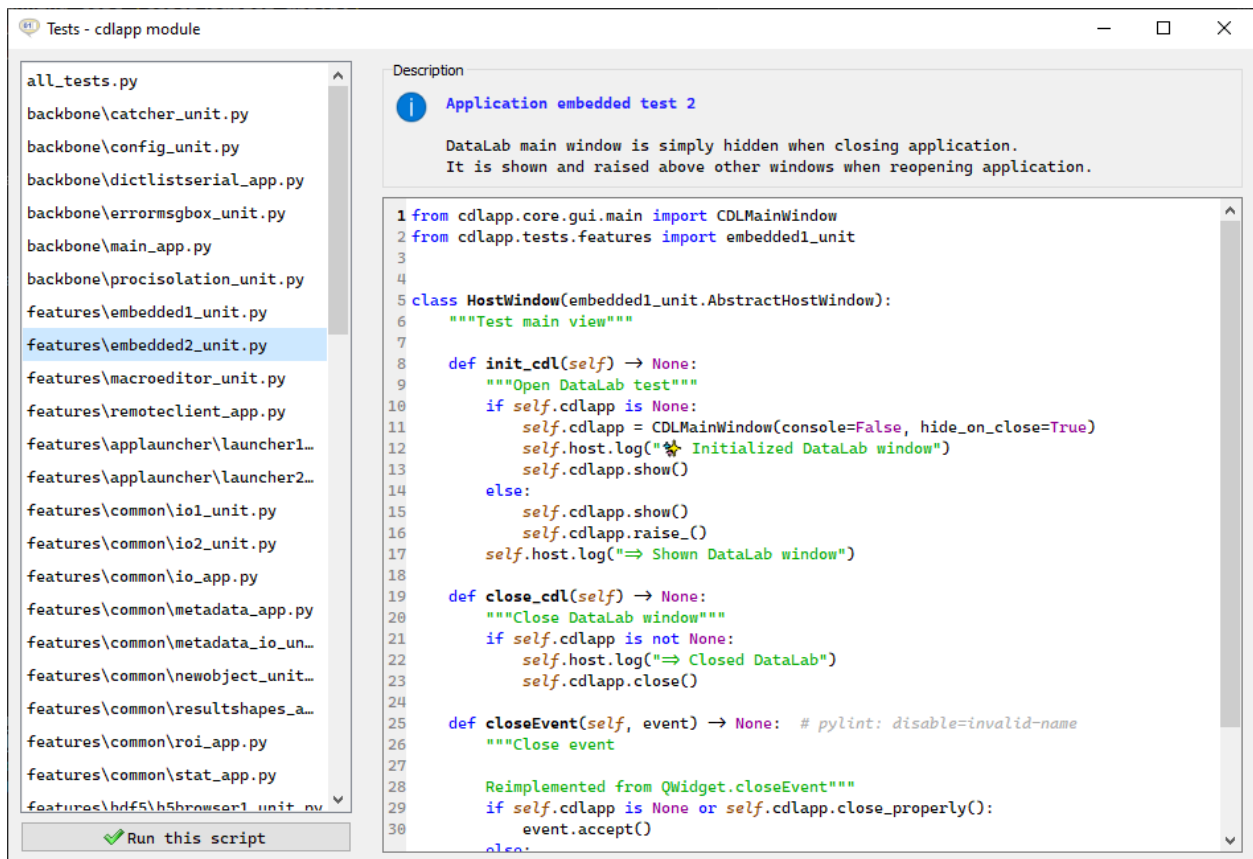
===[01/51]=== Running test [tests\annotations_app.py]
===[02/51]=== Running test [tests\annotations_unit.py]
===[03/51]=== Running test [tests\auto_app.py]
===[04/51]=== Running test [tests\basic1_app.py]
===[05/51]=== Running test [tests\basic2_app.py]
===[06/51]=== Running test [tests\basic3_app.py]

```

2.1.6 Exécuter les tests interactifs

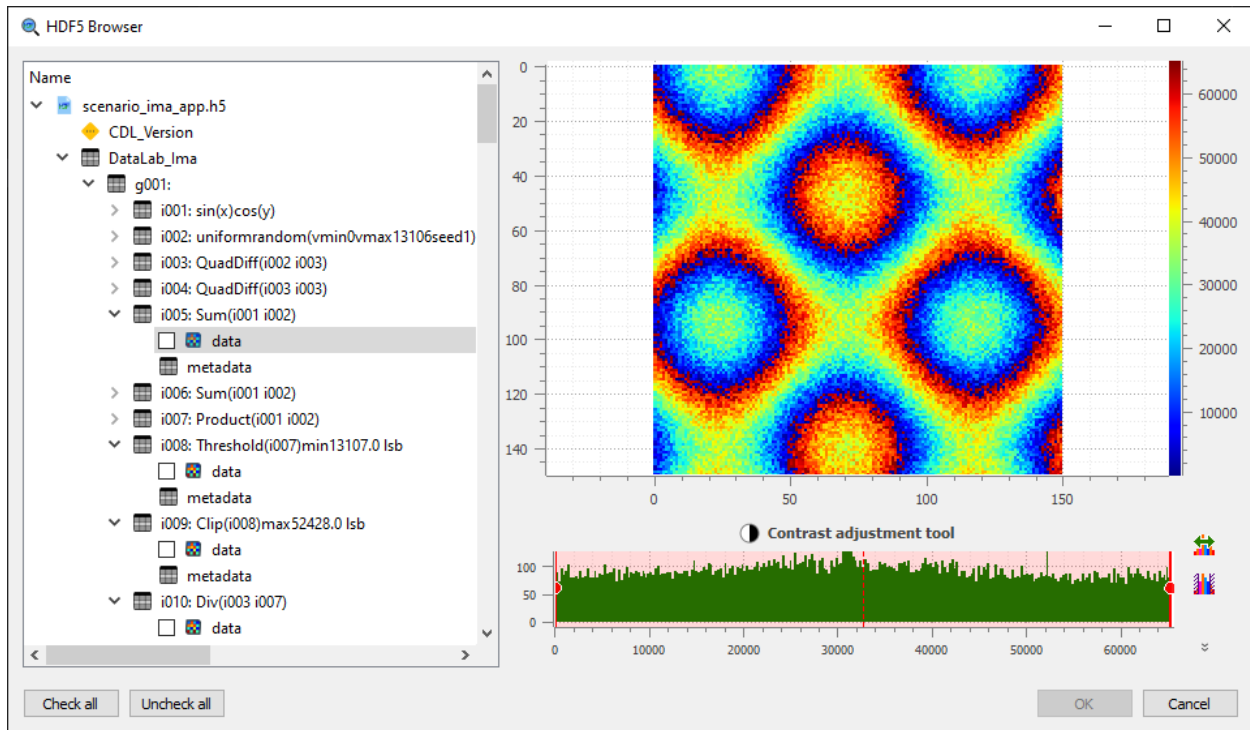
Pour exécuter les tests interactifs de DataLab, taper la commande suivante :

```
$ cdl-tests
```



2.2 Explorateur HDF5

L'explorateur HDF5 est une boîte de dialogue modale permettant d'importer presque n'importe quelles données à 1 ou 2 dimensions dans l'espace de travail de DataLab. Dans certains cas, des métadonnées sont également importées.



Les données de type courbe ou image sont affichées sous la forme d'une vue hiérarchique dans le panneau de gauche, de même que les données scalaires (les valeurs scalaires sont simplement affichées à titre d'informations contextuelles et ne peuvent pas être importées dans l'espace de travail de DataLab).

L'explorateur de fichiers HDF5 est très simple à utiliser :

- Dans le panneau de gauche, sélectionner la courbe ou l'image à importer
- Les données sélectionnées peuvent être visualisées graphiquement dans le panneau de droite
- Cliquer sur « Cocher tout » pour importer toutes les données compatibles
- Valider ensuite en cliquant sur « OK »

2.3 Contrôle à distance

DataLab peut être contrôlé à distance en utilisant le protocole [XML-RPC](#) qui est nativement supporté par Python (et beaucoup d'autres langages). Le contrôle à distance permet d'accéder aux principales fonctionnalités de DataLab à partir d'un processus séparé.

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 Depuis un IDE

DataLab peut être contrôlé à distance depuis un IDE (par exemple [Spyder](#) ou tout autre IDE, ou même un Jupyter Notebook) qui exécute un script Python. Il permet de se connecter à une instance de DataLab en cours d'exécution, d'ajouter un signal et une image, puis d'exécuter des calculs. Cette fonctionnalité est exposée par la classe `RemoteProxy` fournie dans le module `cdl.proxy`.

2.3.2 Depuis une application tierce

DataLab peut également être contrôlé à distance depuis une application tierce, dans le même but.

Si l'application tierce est écrite en Python 3, elle peut directement utiliser la classe `RemoteProxy` comme mentionné ci-dessus. Depuis un autre langage, c'est également réalisable, mais cela nécessite d'implémenter un client XML-RPC dans ce langage en utilisant les mêmes méthodes de serveur proxy que dans la classe `RemoteProxy`.

Les données (signaux et images) peuvent également être échangées entre DataLab et l'application cliente distante, dans les deux sens.

L'application cliente distante peut être écrite dans n'importe quel langage qui prend en charge XML-RPC. Par exemple, il est possible d'écrire une application cliente distante en Python, Java, C++, C#, etc. L'application cliente distante peut être une application graphique ou une application en ligne de commande.

L'application cliente distante peut être exécutée sur le même ordinateur que DataLab ou sur un ordinateur différent. Dans ce dernier cas, l'application cliente distante doit connaître l'adresse IP de l'ordinateur exécutant DataLab.

L'application cliente distante peut être exécutée avant ou après DataLab. Dans ce dernier cas, l'application cliente distante doit essayer de se connecter à DataLab jusqu'à ce qu'elle réussisse.

2.3.3 Fonctionnalités prises en charge

Les fonctionnalités prises en charge sont les suivantes :

- Basculer vers le panneau de signal ou d'image
- Supprimer tous les signaux et images
- Enregistrer la session en cours dans un fichier HDF5
- Ouvrir des fichiers HDF5 dans la session en cours
- Parcourir un fichier HDF5
- Ouvrir un signal ou une image à partir d'un fichier
- Ajouter un signal
- Ajouter une image
- Obtenir la liste des objets
- Exécuter un calcul avec des paramètres

Note : Les objets signal et image sont décrits dans cette section : [Modèle de données interne](#).

Quelques exemples sont fournis pour aider à implémenter une telle communication entre votre application et DataLab :

- Voir le module : `cdl.tests.remoteclient_app`
- Voir le module : `cdl.tests.remoteclient_unit`

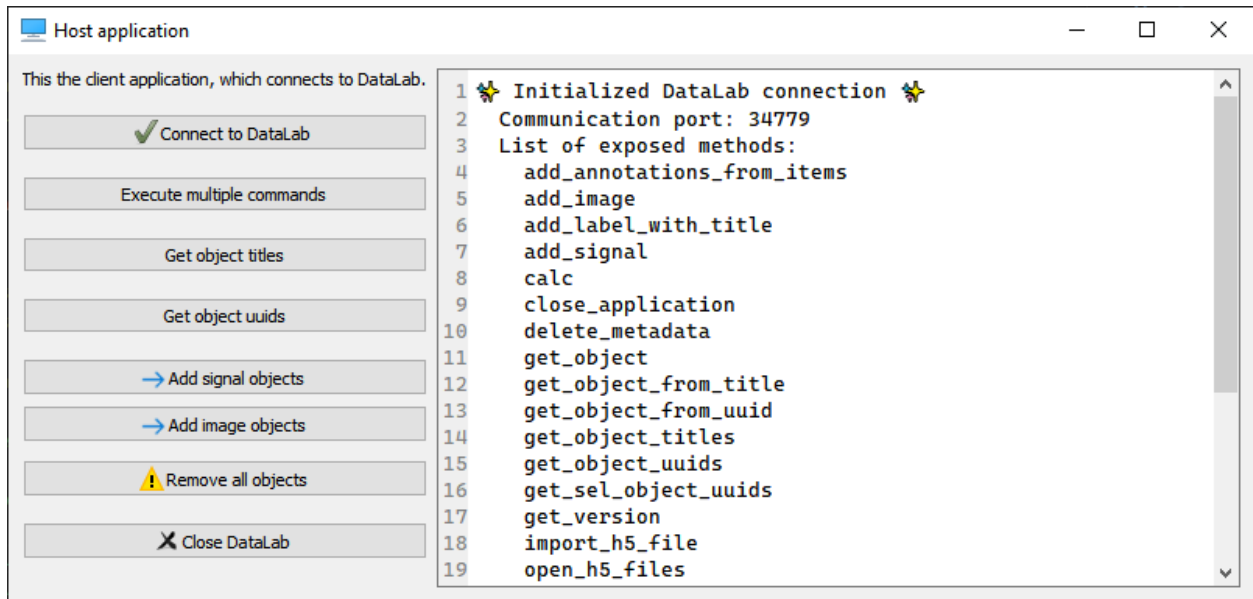


FIG. 2 – Capture d’écran du test de l’application cliente distante (`cdl.tests.remoteclient_app`)

2.3.4 Exemples

Lorsque vous utilisez Python 3, vous pouvez utiliser directement la classe *RemoteProxy* comme dans les exemples cités ci-dessus ou ci-dessous.

Voici un exemple en Python 3 d’un script qui se connecte à une instance de DataLab en cours d’exécution, ajoute un signal et une image, puis exécute des calculs (la structure de cellule du script le rend pratique à utiliser dans l’IDE Spyder) :

```

# -*- 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()

```

(suite sur la page suivante)

(suite de la page précédente)

```

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

z = np.random.rand(20, 20)
proxy.add_image("toto", z)

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

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

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

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

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

proxy.set_current_panel("image")

# %% Executing a lot of commands without refreshing DataLab

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

```

Voici une réimplémentation de cette classe en Python 2.7 :

```

# -*- 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

```

(suite sur la page suivante)

(suite de la page précédente)

```

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():
    """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):

```

(suite sur la page suivante)

(suite de la page précédente)

```

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

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

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

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

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

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

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

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

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

```

(suite sur la page suivante)

(suite de la page précédente)

```

        ylabel=None,
        xlabel=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, xlabel)

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

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

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

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

if __name__ == "__main__":
    test_remote_client()

```

2.3.5 Boîte de dialogue de connexion

Le package DataLab fournit également une boîte de dialogue de connexion qui peut être utilisée pour se connecter à une instance de DataLab en cours d'exécution. Elle est exposée par la classe `cdl.widgets.connection.ConnectionDialog`.

Exemple d'utilisation :

```

# -*- 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

```

(suite sur la page suivante)

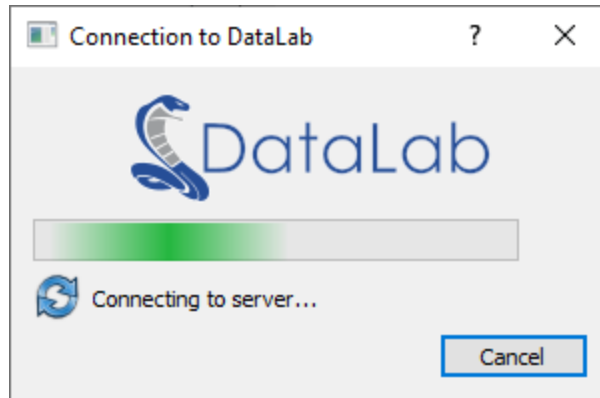


FIG. 3 – Capture d'écran de la boîte de dialogue de connexion (`cdl.widgets.connection.ConnectionDialog`)

(suite de la page précédente)

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

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

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

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

2.3.6 API publique : client distant

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.

Paramètres

- **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.

Lève

- **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.

Paramètres

- **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*.

Renvoie

True if signal was added successfully, False otherwise

Type renvoyé

bool

Lève

- **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.

Paramètres

- **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*.

Renvoie

True if image was added successfully, False otherwise

Type renvoyé

bool

Lève

- **ValueError** – Invalid data dtype

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

Call compute function name in current panel's processor.

Paramètres

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

Renvoie

Compute function result

Type renvoyé*guidata.dataset.DataSet*

get_object(nb_id_title : *int* | *str* | *None* = *None*, panel : *str* | *None* = *None*) → *SignalObj* | *ImageObj*

Get object (signal/image) from index.

Paramètres

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

Renvoie

Object

Lève

- **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).

Paramètres

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

Renvoie

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).

Paramètres

- **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.

Paramètres**obj** (*SignalObj* | *ImageObj*) – Signal or image object**add_label_with_title**(*title* : *str* | *None* = *None*, *panel* : *str* | *None* = *None*) → *None*

Add a label with object title on the associated plot

Paramètres

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

close_application() → *None*

Close DataLab application

context_no_refresh() → *Callable*

Return a context manager to temporarily disable auto refresh.

Renvoie

Context manager

Exemple

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

delete_metadata(*refresh_plot* : *bool* = *True*) → *None*

Delete metadata of selected objects

Paramètres**refresh_plot** (*bool* | *None*) – Refresh plot. Defaults to *True*.**get_current_panel**() → *str*

Return current panel name.

Renvoie

Panel name (valid values : « signal », « image », « macro »)

Type renvoyé*str*

get_group_titles_with_object_infos() → tuple[list[str], list[list[str]], list[list[str]]]

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

Renvoie

groups titles, lists of inner objects uuids and titles

Type renvoyé

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.

Paramètres

panel (str / None) – panel name (valid values : « signal », « image »). If None, current panel is used.

Renvoie

list of object titles

Type renvoyé

list[str]

Lève

ValueError – if panel not found

get_object_uuids(panel : str | None = None) → list[str]

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

Paramètres

panel (str / None) – panel name (valid values : « signal », « image »). If None, current panel is used.

Renvoie

list of object uuids

Type renvoyé

list[str]

Lève

ValueError – if panel not found

classmethod get_public_methods() → list[str]

Return all public methods of the class, except itself.

Renvoie

List of public methods

Type renvoyé

list[str]

get_sel_object_uuids(include_groups : bool = False) → list[str]

Return selected objects uuids.

Paramètres

include_groups – If True, also return objects from selected groups.

Renvoie

List of selected objects uuids.

get_version() → str

Return DataLab version.

Renvoie

DataLab version

Type renvoyé

str

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

Open DataLab HDF5 browser to Import HDF5 file.

Paramètres

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

open_h5_files(h5files : *list[str]* | *None* = *None*, import_all : *bool* | *None* = *None*, reset_all : *bool* | *None* = *None*) → *None*

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

Paramètres

- **h5files** (*list[str]* | *None*) – List of HDF5 files to open. Defaults to *None*.
- **import_all** (*bool* | *None*) – Import all objects from HDF5 files. Defaults to *None*.
- **reset_all** (*bool* | *None*) – Reset all application data. Defaults to *None*.

open_object(filename : *str*) → *None*

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

Paramètres

- filename** (*str*) – File name

raise_window() → *None*

Raise DataLab window

reset_all() → *None*

Reset all application data

save_to_h5_file(filename : *str*) → *None*

Save to a DataLab HDF5 file.

Paramètres

- filename** (*str*) – HDF5 file name

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

Select groups in current panel.

Paramètres

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

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

Select objects in current panel.

Paramètres

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

set_current_panel(panel : *str*) → *None*

Switch to panel.

Paramètres

- panel** (*str*) – Panel name (valid values : « signal », « image », « macro »)

toggle_auto_refresh(state : *bool*) → *None*

Toggle auto refresh state.

Paramètres

- state** (*bool*) – Auto refresh state

toggle_show_titles(*state* : *bool*) → *None*

Toggle show titles state.

Paramètres

state (*bool*) – Show titles state

2.3.7 API publique : méthodes supplémentaires

The remote control class methods (either using the proxy or the remote client) may be completed with additional methods which are dynamically added at runtime. This mechanism allows to access the methods of the « processor » objects of DataLab.

Signal Processor

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

Object handling signal processing : operations, processing, computing

compute_sum() → *None*

Compute sum

compute_average() → *None*

Compute average

compute_product() → *None*

Compute product

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

Extract Region Of Interest (ROI) from data

compute_swap_axes() → *None*

Swap data axes

compute_abs() → *None*

Compute absolute value

compute_re() → *None*

Compute real part

compute_im() → *None*

Compute imaginary part

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

Convert data type

compute_log10() → *None*

Compute Log10

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

Compute difference between two signals

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

Compute quadratic difference between two signals

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

Compute division between two signals

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

Detect peaks from data

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

Normalize data

compute_derivative() → *None*

Compute derivative

compute_integral() → *None*

Compute integral

compute_calibration(*param* : *XYCalibrateParam* | *None* = *None*) → *None*

Compute data linear calibration

compute_threshold(*param* : *ThresholdParam* | *None* = *None*) → *None*

Compute threshold clipping

compute_clip(*param* : *ClipParam* | *None* = *None*) → *None*

Compute maximum data clipping

compute_gaussian_filter(*param* : *GaussianParam* | *None* = *None*) → *None*

Compute gaussian filter

compute_moving_average(*param* : *MovingAverageParam* | *None* = *None*) → *None*

Compute moving average

compute_moving_median(*param* : *MovingMedianParam* | *None* = *None*) → *None*

Compute moving median

compute_wiener() → *None*

Compute Wiener filter

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

Compute iFFT

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

Compute FFT

compute_fit(*name*, *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* : *FWHMParm* | *None* = *None*) → *None*

Compute FWHM

compute_fw1e2() → *None*

Compute FW at $1/e^2$

Image Processor

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

Object handling image processing : operations, processing, computing

compute_sum() → *None*

Compute sum

compute_average() → *None*

Compute average

compute_product() → *None*

Compute product

compute_logp1(*param* : *LogP1Param* | *None* = *None*) → *None*

Compute base 10 logarithm

compute_rotate(*param* : *RotateParam* | *None* = *None*) → *None*

Rotate data arbitrarily

compute_rotate90() → *None*

Rotate data 90°

compute_rotate270() → *None*

Rotate data 270°

compute_fliph() → *None*

Flip data horizontally

compute_flipv() → *None*

Flip data vertically

distribute_on_grid(*param* : *GridParam* | *None* = *None*) → *None*

Distribute images on a grid

reset_positions() → *None*

Reset image positions

compute_resize(*param* : *ResizeParam* | *None* = *None*) → *None*

Resize image

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

Binning image

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

Extract Region Of Interest (ROI) from data

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

Compute profile

compute_average_profile(*param* : *AverageProfileParam* | *None* = *None*) → *None*

Compute average profile

compute_swap_axes() → *None*

Swap data axes

compute_abs() → *None*

Compute absolute value

compute_re() → *None*
 Compute real part

compute_im() → *None*
 Compute imaginary part

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

compute_log10() → *None*
 Compute Log10

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

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

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

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

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

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

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

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

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

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

compute_wiener() → *None*
 Compute Wiener filter

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

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

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

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

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

compute_adjust_sigmoid(*param* : *AdjustSigmoidParam* | *None* = *None*) → *None*

Compute sigmoid correction

compute_rescale_intensity(*param* : *RescaleIntensityParam* | *None* = *None*) → *None*

Rescale image intensity levels

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

Histogram equalization

compute_equalize_adapthist(*param* : *EqualizeAdaptHistParam* | *None* = *None*) → *None*

Adaptive histogram equalization

compute_denoise_tv(*param* : *DenoiseTVParam* | *None* = *None*) → *None*

Compute Total Variation denoising

compute_denoise_bilateral(*param* : *DenoiseBilateralParam* | *None* = *None*) → *None*

Compute bilateral filter denoising

compute_denoise_wavelet(*param* : *DenoiseWaveletParam* | *None* = *None*) → *None*

Compute Wavelet denoising

compute_denoise_tophat(*param* : *MorphologyParam* | *None* = *None*) → *None*

Denoise using White Top-Hat

compute_all_denoise(*params* : *list* | *None* = *None*) → *None*

Compute all denoising filters

compute_white_tophat(*param* : *MorphologyParam* | *None* = *None*) → *None*

Compute White Top-Hat

compute_black_tophat(*param* : *MorphologyParam* | *None* = *None*) → *None*

Compute Black Top-Hat

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

Compute Erosion

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

Compute Dilation

compute_opening(*param* : *MorphologyParam* | *None* = *None*) → *None*

Compute morphological opening

compute_closing(*param* : *MorphologyParam* | *None* = *None*) → *None*

Compute morphological closing

compute_all_morphology(*param* : *MorphologyParam* | *None* = *None*) → *None*

Compute all morphology filters

compute_canny(*param* : *CannyParam* | *None* = *None*) → *None*

Compute Canny filter

compute_roberts() → *None*

Compute Roberts filter

compute_prewitt() → *None*

Compute Prewitt filter

compute_prewitt_h() → *None*

Compute Prewitt filter (horizontal)

compute_prewitt_v() → *None*
 Compute Prewitt filter (vertical)

compute_sobel() → *None*
 Compute Sobel filter

compute_sobel_h() → *None*
 Compute Sobel filter (horizontal)

compute_sobel_v() → *None*
 Compute Sobel filter (vertical)

compute_scharr() → *None*
 Compute Scharr filter

compute_scharr_h() → *None*
 Compute Scharr filter (horizontal)

compute_scharr_v() → *None*
 Compute Scharr filter (vertical)

compute_farid() → *None*
 Compute Farid filter

compute_farid_h() → *None*
 Compute Farid filter (horizontal)

compute_farid_v() → *None*
 Compute Farid filter (vertical)

compute_laplace() → *None*
 Compute Laplace filter

compute_all_edges() → *None*
 Compute all edges

compute_centroid() → *None*
 Compute image centroid

compute_enclosing_circle() → *None*
 Compute minimum enclosing circle

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

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

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

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

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

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

```
compute_blob_opencv(param : BlobOpenCVParam | None = None) → None
```

Compute blob detection using OpenCV

2.4 Modèle de données interne

Dans son modèle de données interne, DataLab stocke les données à l'aide de deux classes principales :

- `cdl.core.model.signal.SignalObj`, qui représente un objet signal, et
- `cdl.core.model.image.ImageObj`, qui représente un objet image.

Ces classes sont définies dans le paquet `cdl.core.model` mais sont exposées publiquement dans le paquet `cdl.obj`.

Par ailleurs, DataLab utilise de nombreux jeux de données différents (basés sur la classe `DataSet` de `guidata`) pour stocker les paramètres des calculs. Ces jeux de données sont définis dans différents modules mais sont exposés publiquement dans le paquet `cdl.param`.

2.4.1 API publique

L'API publique est la suivante :

DataLab Public API's object model module

This modules aims at providing all the necessary classes and functions to create and manipulate DataLab signal and image objects.

Those classes and functions are defined in other modules :

- `cdl.core.model.base`
- `cdl.core.model.image`
- `cdl.core.model.signal`
- `cdl.core.io`

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

DataLab Base Computation parameters module

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

Those datasets are defined other modules :

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

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

Signal object and related classes

```
class cdl.core.model.signal.CurveStyles
```

Bases : `object`

Object to manage curve styles

```
style_generator()
```

Cycling through curve styles

classmethod `apply_style(param : CurveParam)`

Apply style to curve

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

Bases : `DataSet`

Signal ROI parameters

class `cdl.core.model.signal.SignalObj(title=None, comment=None, icon="")`

Bases : `DataSet`, `BaseObj`

Signal object

regenerate_uuid()

Regenerate UUID

This method is used to regenerate UUID after loading the object from a file. This is required to avoid UUID conflicts when loading objects from file without clearing the workspace first.

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

Copy object.

Paramètres

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

Renvoie

copied object

Type renvoyé

SignalObj

set_data_type(dtype : dtype) → None

Change data type.

Paramètres

- 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

Paramètres

- **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).

Paramètres

roi_index (*int*) – ROI index

Renvoie

data

Type renvoyé

numpy.ndarray

update_plot_item_parameters(*item* : *CurveItem*) → *None*

Update plot item parameters from object data/metadata

Takes into account a subset of plot item parameters. Those parameters may have been overridden by object metadata entries or other object data. The goal is to update the plot item accordingly.

This is *almost* the inverse operation of *update_metadata_from_plot_item*.

Paramètres

item – plot item

update_metadata_from_plot_item(*item* : *CurveItem*) → *None*

Update metadata from plot item.

Takes into account a subset of plot item parameters. Those parameters may have been modified by the user through the plot item GUI. The goal is to update the metadata accordingly.

This is *almost* the inverse operation of *update_plot_item_parameters*.

Paramètres

item – plot item

make_item(*update_from* : *CurveItem* = *None*) → *CurveItem*

Make plot item from data.

Paramètres

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

Renvoie

plot item

Type renvoyé

CurveItem

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

Update plot item from data.

Paramètres

— **item** (*CurveItem*) – plot item

— **data_changed** (*bool*) – if *True*, data has changed

roi_coords_to_indexes(*coords* : *list*) → *ndarray*

Convert ROI coordinates to indexes.

Paramètres

coords (*list*) – coordinates

Renvoie

indexes

Type renvoyé

numpy.ndarray

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

Return ROI parameters dataset.

Paramètres

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

static params_to_roidata(*params* : *DataSetGroup*) → *ndarray*

Convert ROI dataset group to ROI array data.

Paramètres

- params** (*DataSetGroup*) – ROI dataset group

Renvoie

ROI array data

Type renvoyé

numpy.ndarray

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

Return a new ROI item from scratch

Paramètres

- **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.

Paramètres

- **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

Paramètres

- 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.

Paramètres

- **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)

Renvoie

signal object

Type renvoyé*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.

Paramètres

- **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)

Renvoie

new signal dataset instance

Type renvoyé*NewSignalParam*

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

Triangle function

Paramètres

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.

Paramètres

- **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

Renvoie

signal object or None if canceled

Type renvoyé

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

Paramètres

- **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

Paramètres

- **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.

Paramètres

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

Paramètres

— **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

Paramètres

— **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

Paramètres

— **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

Paramètres

— **index** (*int* | *None*) – ROI index

— **fmt** (*str*) – format string

— **lbl** (*bool*) – if True, show label

— **editable** (*bool*) – if True, ROI is editable

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

Returns name_{index}

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

Bases : *DataSet*

ROI parameters

get_suffix()

Get suffix text representation for ROI extraction

get_coords()

Get ROI coordinates

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

Bases : *DataSet*

ROI parameters

get_single_roi()

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

get_suffix()

Get suffix text representation for ROI extraction

get_coords()

Get ROI coordinates

property x0

Return rectangle top left corner X coordinate

property x1

Return rectangle bottom right corner X coordinate

property y0

Return rectangle top left corner Y coordinate

property y1

Return rectangle bottom right corner Y coordinate

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

Bases : `DataSet`, `BaseObj`

Image object

regenerate_uuid()

Regenerate UUID

This method is used to regenerate UUID after loading the object from a file. This is required to avoid UUID conflicts when loading objects from file without clearing the workspace first.

property size: tuple[int, int]

Returns (width, height)

set_metadata_from(*obj : Mapping | dict*) → *None*

Set metadata from object : dict-like (only string keys are considered) or any other object (iterating over supported attributes)

Paramètres

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).

Paramètres

roi_index (*int*) – ROI index

Renvoie

masked data

Type renvoyé`numpy.ndarray`**copy**(*title* : *str* | *None* = *None*, *dtype* : *dtype* | *None* = *None*) → *ImageObj*

Copy object.

Paramètres

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

Renvoie

copied object

Type renvoyé*ImageObj***set_data_type**(*dtype* : *dtype*) → *None*

Change data type.

Paramètres**dtype** (*numpy.dtype*) – data type**update_plot_item_parameters**(*item* : *MaskedImageItem*) → *None*

Update plot item parameters from object data/metadata

Takes into account a subset of plot item parameters. Those parameters may have been overridden by object metadata entries or other object data. The goal is to update the plot item accordingly.

This is *almost* the inverse operation of *update_metadata_from_plot_item*.

Paramètres**item** – plot item**update_metadata_from_plot_item**(*item* : *MaskedImageItem*) → *None*

Update metadata from plot item.

Takes into account a subset of plot item parameters. Those parameters may have been modified by the user through the plot item GUI. The goal is to update the metadata accordingly.

This is *almost* the inverse operation of *update_plot_item_parameters*.

Paramètres**item** – plot item**make_item**(*update_from* : *MaskedImageItem* | *None* = *None*) → *MaskedImageItem*

Make plot item from data.

Paramètres**update_from** (*MaskedImageItem* | *None*) – update from plot item**Renvoie**

plot item

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

Update plot item from data.

Paramètres

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

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

Return ROI parameters dataset.

Paramètres

```

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

static params_to_roidata(params : DataSetGroup) → ndarray | None
    Convert ROI dataset group to ROI array data.

    Paramètres
    params (DataSetGroup) – ROI dataset group

    Renvoie
    ROI array data

    Type renvoyé
    numpy.ndarray

new_roi_item(fmt : str, lbl : bool, editable : bool, geometry : RoiDataGeometries) → MaskedImageItem
    Return a new ROI item from scratch

    Paramètres
    — 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.

    Paramètres
    coords (list) – coordinates

    Renvoie
    indexes

    Type renvoyé
    numpy.ndarray

iterate_roi_items(fmt : str, lbl : bool, editable : bool = True) → Iterator
    Make plot item representing a Region of Interest.

    Paramètres
    — 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)

    Renvoie
    masked data

    Type renvoyé
    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

    Paramètres
    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

Paramètres

- **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)

Renvoie

image object

Type renvoyé

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.

Paramètres

- **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)

Renvoie

new image dataset instance

Type renvoyé

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.

Paramètres

- **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

Renvoie

new image object or None if user cancelled

Type renvoyé

ImageObj

2.5 Plugins

DataLab est une application modulaire. Il est possible d'ajouter de nouvelles fonctionnalités à DataLab en écrivant des plugins. Un plugin est un module Python qui est chargé au démarrage par DataLab. Un plugin peut ajouter de nouvelles fonctionnalités à DataLab, ou modifier des fonctionnalités existantes.

Le système de plugins prend actuellement en charge les fonctionnalités suivantes :

- Fonctionnalités de traitement : ajouter de nouvelles tâches de traitement au système de traitement DataLab, y compris des interfaces graphiques spécifiques.
- Entrée/sortie : ajouter de nouveaux formats de fichiers au système d'entrée/sortie de DataLab.
- Fonctionnalités HDF5 : ajouter de nouveaux formats de fichiers HDF5 au système d'entrée/sortie HDF5 de DataLab.

2.5.1 Qu'est-ce qu'un plugin ?

Un plugin est un module Python qui est chargé au démarrage par DataLab. Un plugin peut ajouter de nouvelles fonctionnalités à DataLab, ou modifier des fonctionnalités existantes.

Un plugin est un module Python qui contient une classe dérivée de la classe *PluginBase*. Le nom de la classe n'est pas important, tant qu'elle est dérivée de *PluginBase*. La classe doit avoir un attribut *PLUGIN_INFO* qui est une instance de la classe *PluginInfo*. L'attribut *PLUGIN_INFO* est utilisé par DataLab pour récupérer des informations sur le plugin.

2.5.2 Où est positionné un plugin ?

Etant donné que les plugins sont des modules Python, ils peuvent être placés n'importe où dans le chemin Python de l'installation de DataLab.

Des emplacements supplémentaires spéciaux sont disponibles pour les plugins :

- Le répertoire *plugins* dans le dossier de configuration de l'utilisateur (par exemple *C:\Users\John-Doe\DataLab\plugins* sur Windows ou *~/DataLab/plugins* sur Linux).
- Le répertoire *plugins* dans le même dossier que l'exécutable *DataLab* en cas d'installation autonome.
- Le répertoire *plugins* dans le package *cdl* en cas de plugins internes uniquement (c'est-à-dire qu'il n'est pas recommandé d'y placer vos propres plugins).

2.5.3 Exemple : plugin de traitement

Voici un exemple simple d'un plugin qui ajoute une nouvelle fonctionnalité à 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_l1(src, "add_gaussian_noise", f"mu={p.mu},sigma={p.sigma}")
    test_data.add_gaussian_noise_to_signal(dst, p)
    return dst

def add_noise_to_image(src: dlo.ImageObj, p: dlo.NormalRandomParam) -> dlo.ImageObj:
    """Add gaussian noise to image"""
    dst = cpima.dst_l1(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"),
```

(suite sur la page suivante)

(suite de la page précédente)

```

)

# 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))
    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

```

(suite sur la page suivante)

(suite de la page précédente)

```

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)

# Plugin menu entries -----
def create_actions(self) -> None:
    """Create actions"""
    # Signal panel -----
    sah = self.signalpanel.athandler
    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,

```

(suite sur la page suivante)

(suite de la page précédente)

```

    )
    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"),
            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 Exemple : plugin d'entrée/sortie

Voici un exemple simple d'un plugin qui ajoute de nouveaux formats de fichiers à 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 = "<llllllffl"

    def __init__(self, fname: str = None, debug: bool = False) -> None:
        self.__debug = debug
        self.file_format = None # long
        self.nbcols = None # long
        self.nbrows = None # long
        self.nbframes = None # long
        self.pixeltype = None # long
        self.quantlevels = None # long
        self.maxlevel = None # float
        self.minlevel = None # float
        self.comment_length = None # long
        self.fname = None
        self.data = None
        if fname is not None:
```

(suite sur la page suivante)

(suite de la page précédente)

```

        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,
            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

```

(suite sur la page suivante)

(suite de la page précédente)

```

        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"""

    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

```

(suite sur la page suivante)

(suite de la page précédente)

```

Args:
    filename (str): path to XYZ file

Returns:
    np.ndarray: image data
"""
with open(filename, "rb") as fdesc:
    cols = int(np.fromfile(fdesc, dtype=np.uint16, count=1)[0])
    rows = int(np.fromfile(fdesc, dtype=np.uint16, count=1)[0])
    arr = np.fromfile(fdesc, dtype=np.uint16, count=cols * rows)
    arr = arr.reshape((rows, cols))
return np.fliplr(arr)

```

2.5.5 Other examples

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

2.5.6 API publique

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]
```

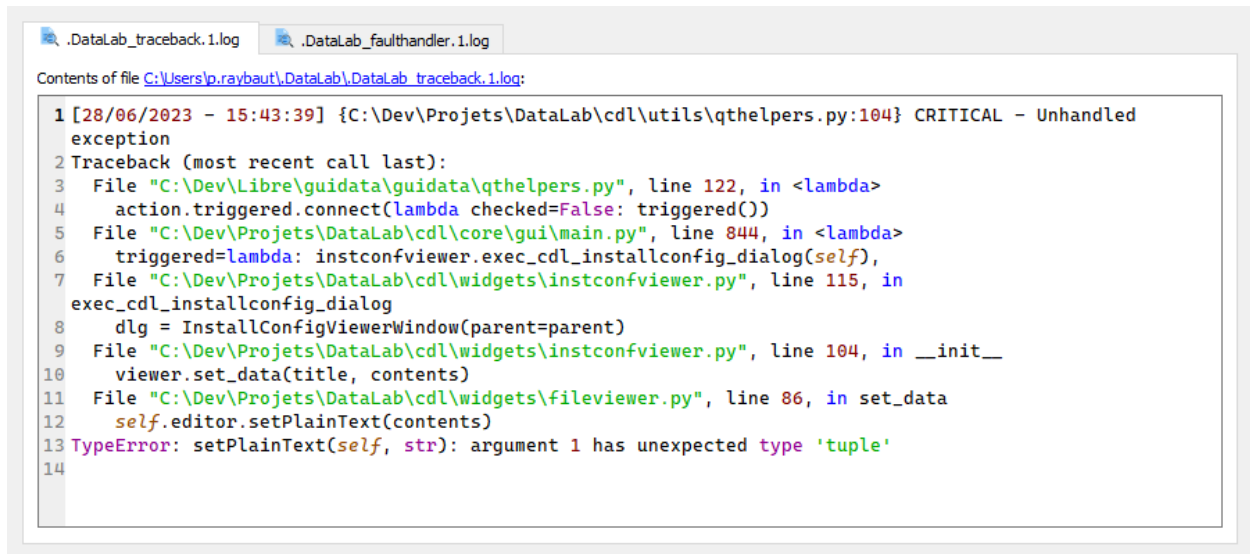
Discover plugins using naming convention

2.6 Journaux de bord

Malgré des efforts considérables en matière de tests (tests unitaires, couverture de tests, etc.), DataLab peut s'arrêter inopinément ou se comporter de façon inattendue.

Pour traiter ce type de situation, DataLab fournit deux types de journaux de bord (localisés dans votre répertoire utilisateur) :

- « Traceback log », pour les exceptions Python
- « Faulthandler log », pour les erreurs système (p.ex. crash lié à Qt)



The screenshot shows a log viewer window with two tabs: ".DataLab_traceback.1.log" and ".DataLab_faulthandler.1.log". The active tab is ".DataLab_traceback.1.log", displaying the contents of the file "C:\Users\p.raybaut\DataLab\DataLab_traceback.1.log". The log content shows a critical unhandled exception on 28/06/2023 at 15:43:39, originating from a lambda function in "C:\Dev\Libre\guidata\guidata\qthelpers.py" at line 104. The traceback shows the call stack leading to a "TypeError: setPlainText(self, str): argument 1 has unexpected type 'tuple'" in "C:\Dev\Projets\DataLab\cdl\widgets\instconfviewer.py" at line 115.

```

1 [28/06/2023 - 15:43:39] {C:\Dev\Projets\DataLab\cdl\utils\qthelpers.py:104} CRITICAL - Unhandled
exception
2 Traceback (most recent call last):
3   File "C:\Dev\Libre\guidata\guidata\qthelpers.py", line 122, in <lambda>
4     action.triggered.connect(lambda checked=False: triggered())
5   File "C:\Dev\Projets\DataLab\cdl\core\gui\main.py", line 844, in <lambda>
6     triggered=lambda: instconfviewer.exec_cdl_installconfig_dialog(self),
7   File "C:\Dev\Projets\DataLab\cdl\widgets\instconfviewer.py", line 115, in
exec_cdl_installconfig_dialog
8     dlg = InstallConfigViewerWindow(parent=parent)
9   File "C:\Dev\Projets\DataLab\cdl\widgets\instconfviewer.py", line 104, in __init__
10     viewer.set_data(title, contents)
11   File "C:\Dev\Projets\DataLab\cdl\widgets\fileviewer.py", line 86, in set_data
12     self.editor.setPlainText(contents)
13 TypeError: setPlainText(self, str): argument 1 has unexpected type 'tuple'
14

```

FIG. 4 – Journaux de bord DataLab (voir le menu « ? »)

Lorsque DataLab s'arrête brutalement en raison d'une erreur système ou si une exception Python est levée durant son exécution, ces journaux de bord sont mis à jour en conséquence. DataLab notifie même l'utilisateur de la disponibilité de nouvelles informations dans les journaux de bord, lors du prochain démarrage de l'application. Ceci est une invitation à soumettre un rapport d'anomalie.

Signaler un comportement inattendu ou tout autre type d'anomalie sur la page [GitHub Issues](#) sera grandement apprécié, d'autant plus si vous prenez soin de joindre le contenu des journaux de bord (ainsi que des informations sur votre configuration, cf. *Installation et configuration*).

2.7 Installation et configuration

En raison des multiples façons d'installer DataLab sur votre machine, comprendre l'origine d'un dysfonctionnement de l'application sans information sur votre configuration peut s'avérer très difficile.

C'est pourquoi DataLab fournit la boîte de dialogue « Installation et configuration » qui rassemble toutes les informations sur votre installation et votre configuration utilisateur.

Signaler un comportement inattendu ou tout autre type d'anomalie sur la page [GitHub Issues](#) sera grandement apprécié, d'autant plus si vous prenez soin de joindre les informations ci-dessus (ainsi que les journaux d'historique, cf. *Journaux de bord*).

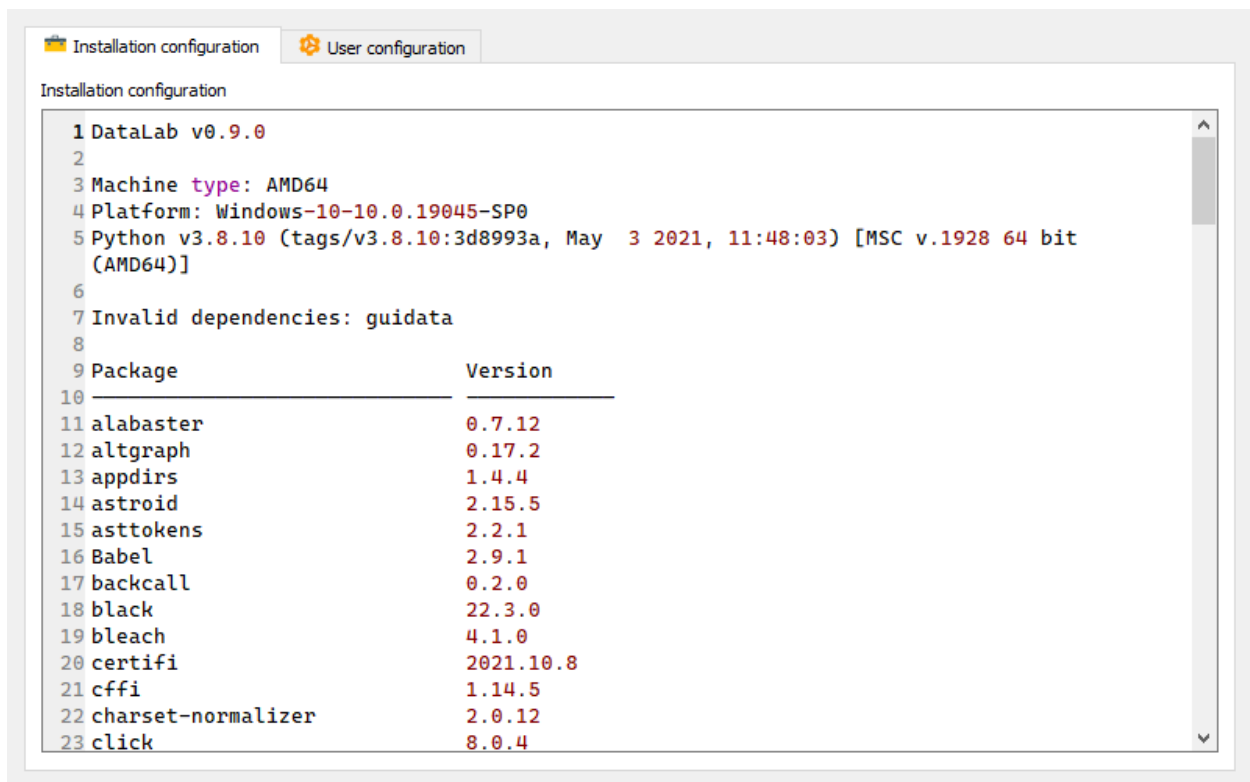


FIG. 5 – Installation et configuration (voir menu « ? »)

CHAPITRE 3

Traitement du signal

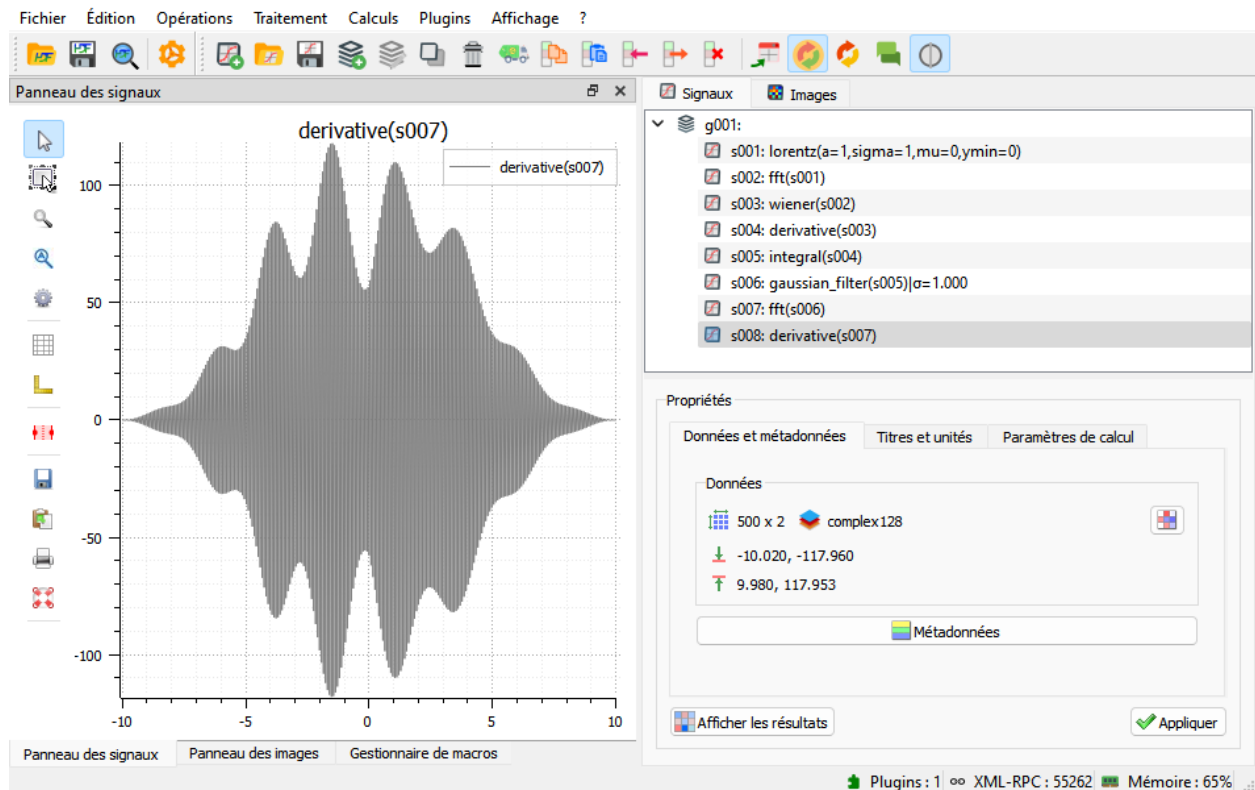
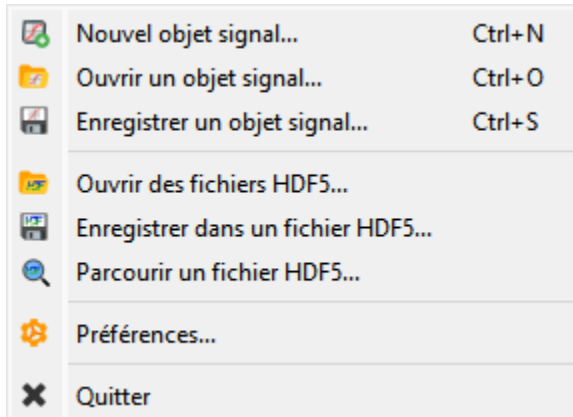


FIG. 1 – Fenêtre principale de DataLab - Traitement du signal

3.1 Menu « Fichier »



Nouveau signal

Crée un nouveau signal depuis différents modèles :

Modèle	Equation
Zéros	$y[i] = 0$
Aléatoire	$y[i] \in [-0.5, 0.5]$
Gaussienne	$y = y_0 + \frac{A}{\sqrt{2\pi} \cdot \sigma} \cdot \exp\left(-\frac{1}{2} \cdot \left(\frac{x - x_0}{\sigma}\right)^2\right)$
Lorentzienne	$y = y_0 + \frac{A}{\sigma \cdot \pi} \cdot \frac{1}{1 + \left(\frac{x - x_0}{\sigma}\right)^2}$
Voigt	$y = y_0 + A \cdot \frac{\text{Re}(\exp(-z^2)) \cdot \text{erfc}(-j \cdot z)}{\sqrt{2\pi} \cdot \sigma}$ avec $z = \frac{x - x_0 - j \cdot \sigma}{\sqrt{2} \cdot \sigma}$

Ouvrir un signal

Crée un signal depuis l'un des types de fichiers pris en charge :

Type de fichier	Extensions
Fichiers texte	.txt, .csv
Tableaux NumPy	.npy

Enregistrer un signal

Enregistre le signal sélectionné dans l'un des types de fichier pris en charge :

Type de fichier	Extensions
Fichiers texte	.csv

Ouvrir un fichier HDF5

Importer les données d'un fichier HDF5.

Enregistrer un fichier HDF5

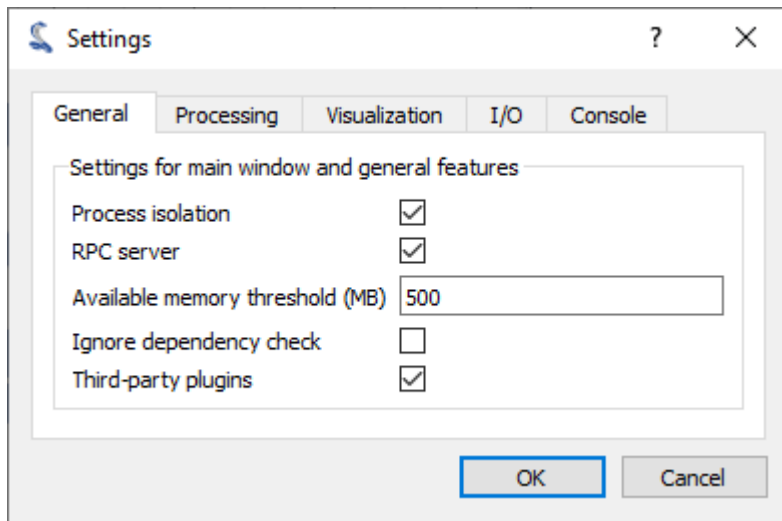
Exporter l'ensemble de la session DataLab (tous les signaux et images) vers un fichier HDF5.

Explorer un fichier HDF5

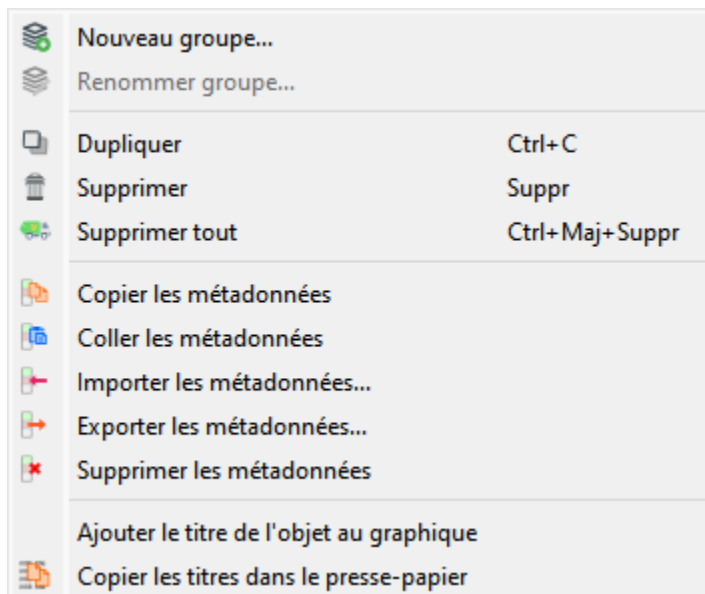
Ouvrir l'*Explorateur HDF5* dans une nouvelle fenêtre pour explorer et éventuellement importer des données depuis un fichier HDF5.

Préférences

Ouvrir la boîte de dialogue « Préférences ».



3.2 Menu « Edition »



Dupliquer

Crée un nouveau signal identique à l'objet sélectionné.

Supprimer

Supprimer le signal sélectionné.

Supprimer tout

Supprimer tous les signaux.

Copier les métadonnées

Copier les métadonnées de l'image sélectionnée vers le presse-papier.

Coller les métadonnées

Coller les métadonnées depuis le presse-papier vers l'image sélectionnée.

Importer les métadonnées dans le signal

Importer les métadonnées depuis un fichier texte JSON.

Exporter les métadonnées du signal

Exporter les métadonnées vers un fichier texte JSON.

Supprimer les métadonnées

Supprime les métadonnées du signal sélectionné. Les métadonnées contiennent des informations additionnelles telles que les régions d'intérêt ou encore des résultats de calcul.

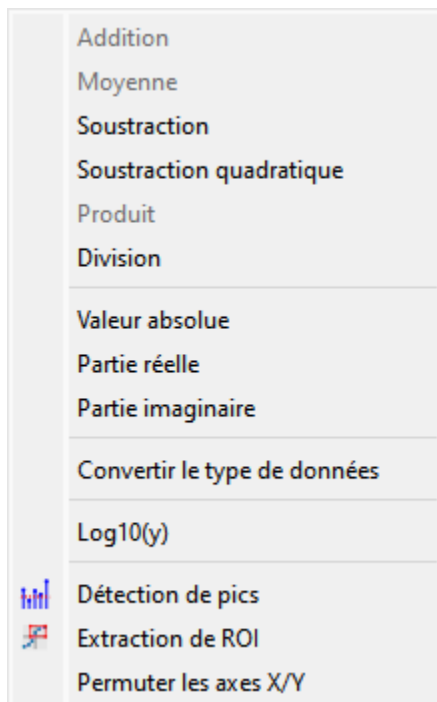
Ajouter le titre de l'objet au graphique

Ajoute le titre du signal sélectionné au graphique associé.

Copier les titres dans le presse-papier

Copie les titres de tous les signaux dans le presse-papier, sous la forme d'un texte multiligne. Ce texte peut être ensuite utilisé pour reproduire une chaîne de traitement, par exemple.

3.3 Menu « Opérations »

**Addition**

Crée un signal à partir de la somme des signaux sélectionnés :

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

Moyenne

Crée un signal à partir de la moyenne des signaux sélectionnés :

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

Soustraction

Crée un signal à partir de la différence des **deux** signaux sélectionnés :

$$y_2 = y_1 - y_0$$

Produit

Crée un signal à partir du produit de tous les signaux sélectionnés :

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

Division

Crée un signal à partir de la division des **deux** signaux sélectionnés :

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

Valeur absolue

Crée un signal à partir de la valeur absolue de chaque signal sélectionné :

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

Partie réelle

Crée un signal à partir de la partie réelle de chaque signal sélectionné :

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

Partie imaginaire

Crée un signal à partir de la partie imaginaire de chaque signal sélectionné :

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

Convertir le type de données

Crée un signal à partir de la conversion du type de données du signal sélectionné.

Note : La conversion du type de données utilise la fonction `numpy.ndarray.astype()` avec les paramètres par défaut (`casting="unsafe"`).

Log10(y)

Crée un signal à partir du logarithme base 10 de chaque signal sélectionné :

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

Détection de pics

Crée un signal à partir de la détection automatique des pics de chaque signal sélectionné :

Extraction de ROI

Crée un signal à partir d'une région d'intérêt (ROI) définie par l'utilisateur.

Permuter les axes X/Y

Crée un signal à partir des données inversées X/Y du signal sélectionné.

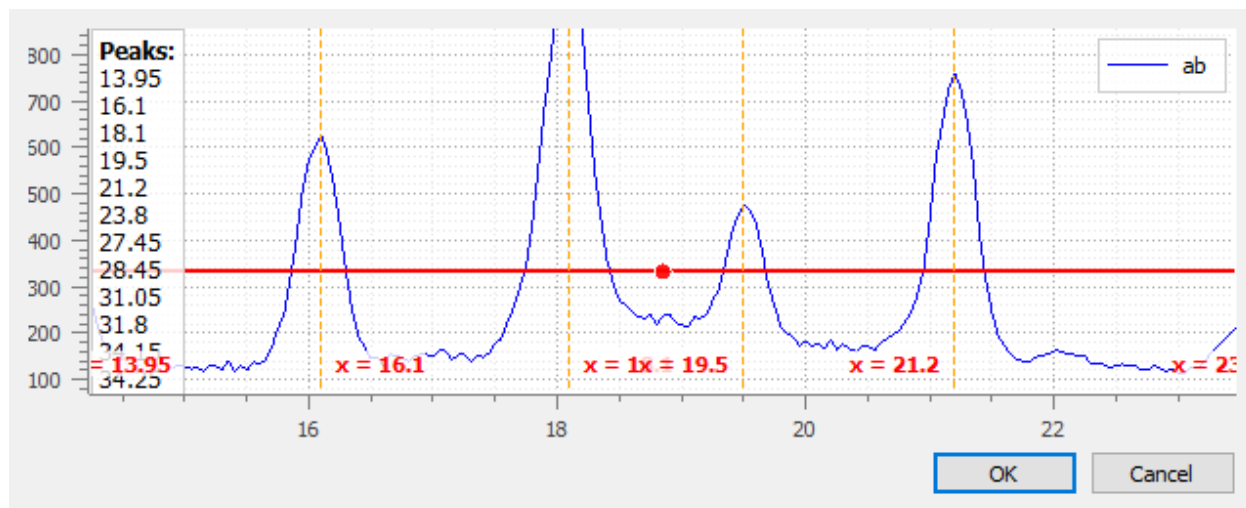


FIG. 2 – Boîte de dialogue de détection de pics : le seuil de détection est ajustable en déplaçant le curseur horizontal, les pics sont détectés automatiquement (des marqueurs verticaux indiquent les pics détectés avec leur position)

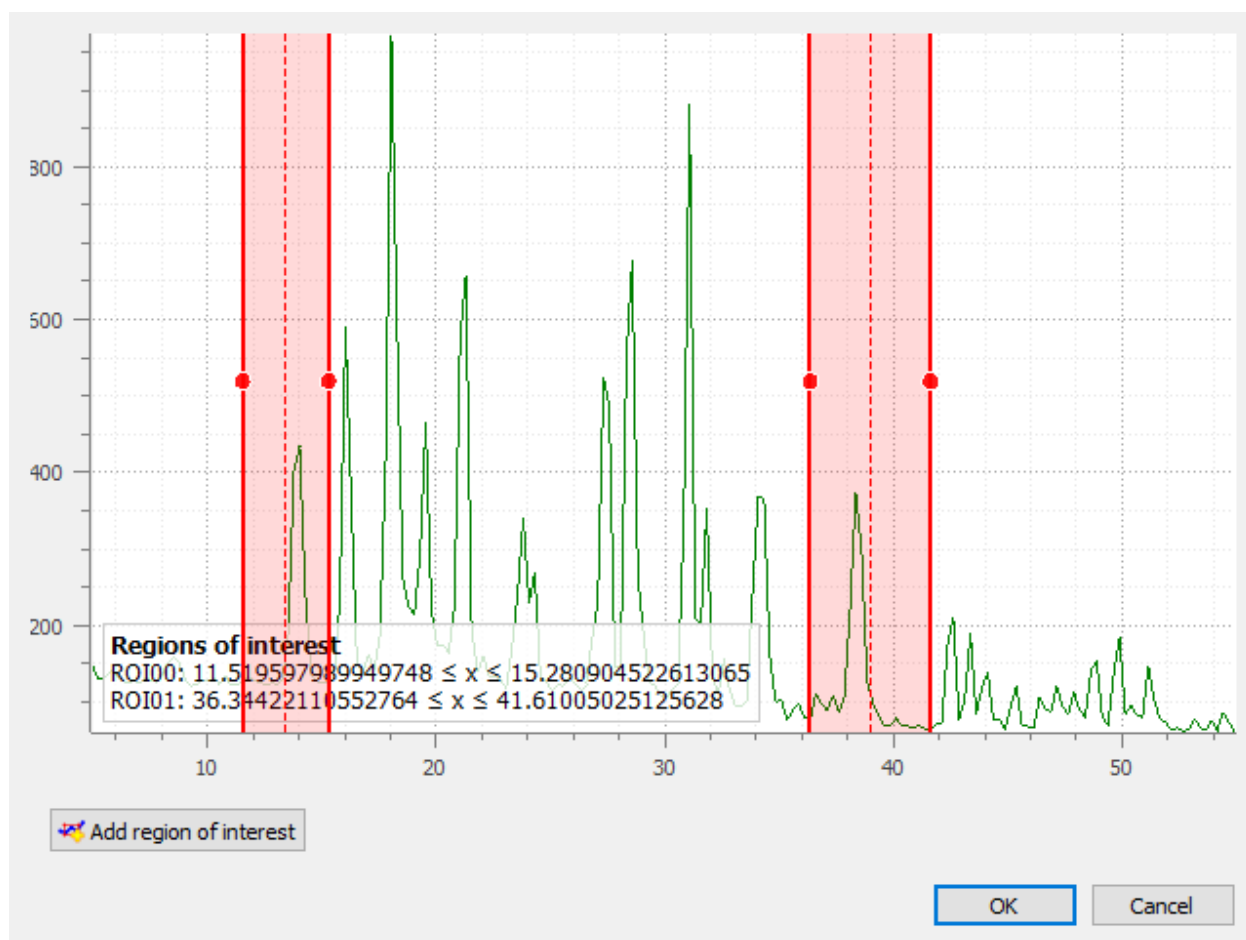
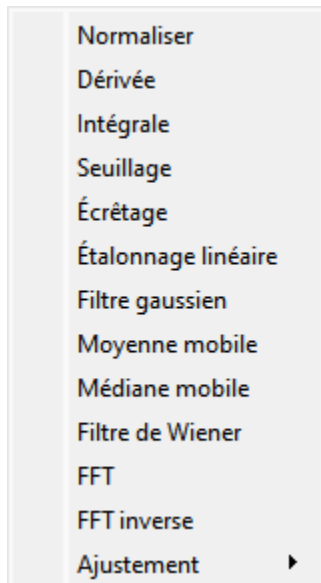


FIG. 3 – Boîte de dialogue d'extraction de ROI : la région d'intérêt (ROI) est définie en ajustant la position et la largeur de l'échelle horizontale de sélection.

3.4 Menu « Traitement »



Normaliser

Crée un signal à partir de la normalisation de chaque signal sélectionné (normalisation par le maximum, en amplitude, en somme ou en énergie) :

Paramètre	Normalisation
Maximum	$y_1 = \frac{y_0}{\max(y_0)}$
Amplitude	$y_1 = \frac{y'_0}{\max(y'_0)}$ avec $y'_0 = y_0 - \min(y_0)$
Addition	$y_1 = \frac{y_0}{\sum_{n=0}^N y_0[n]}$
Energie	$y_1 = \frac{y_0}{\sum_{n=0}^N y_0[n] ^2}$

Dérivée

Crée un signal à partir de la dérivée de chaque signal sélectionné.

Intégrale

Crée un signal à partir de l'intégrale de chaque signal sélectionné.

Étalonnage linéaire

Crée un signal à partir de l'étalonnage linéaire (par rapport aux axes X et Y) de chaque signal sélectionné.

Paramètre	Étalonnage linéaire
Axe des X	$x_1 = a.x_0 + b$
Axe des Y	$y_1 = a.y_0 + b$

Filtre gaussien

Calcule le résultat du filtre gaussien 1D de chaque signal sélectionné (implémentation basée sur `scipy.ndimage.gaussian_filter1d`).

Moyenne mobile

Calcule le résultat de la moyenne mobile sur M points de chaque signal sélectionné, sans effet de bord :

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

Médiane mobile

Calcule le résultat de la médiane mobile de chaque signal sélectionné (implémentation basée sur `scipy.signal.medfilt`).

Filtre de Wiener

Calcule le résultat du filtre de Wiener sur chaque signal sélectionné (implémentation basée sur `scipy.signal.wiener`).

FFT

Crée un signal à partir de la transformée de Fourier rapide (FFT) de chaque signal sélectionné.

FFT inverse

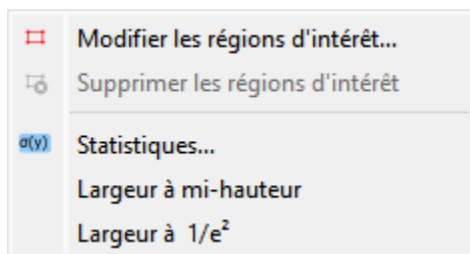
Crée un signal à partir de la transformée de Fourier rapide inverse (FFT inverse) de chaque signal sélectionné.

Ajustements lorentzien, Voigt, polynomial et multi-gaussien

Ouvre une boîte de dialogue permettant de réaliser des ajustements de courbe de manière interactive.

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

3.5 Menu « Calculs »



Modifier les régions d'intérêt

Ouvre une boîte de dialogue pour définir des régions d'intérêt (ROI) multiples. Les ROI sont stockées sous la forme de métadonnées ; elles sont donc attachées au signal.

La boîte de dialogue de définition de ROI est exactement la même que celle utilisée pour l'extraction de ROI (voir plus haut) : la ROI est définie en ajustant la position et la largeur de l'échelle horizontale de sélection.

Supprimer les régions d'intérêt

Supprimer toutes les ROI définies pour l'objet ou les objets sélectionné(s).

Statistiques

Calcule des statistiques sur les signaux sélectionnés et affiche un tableau récapitulatif.

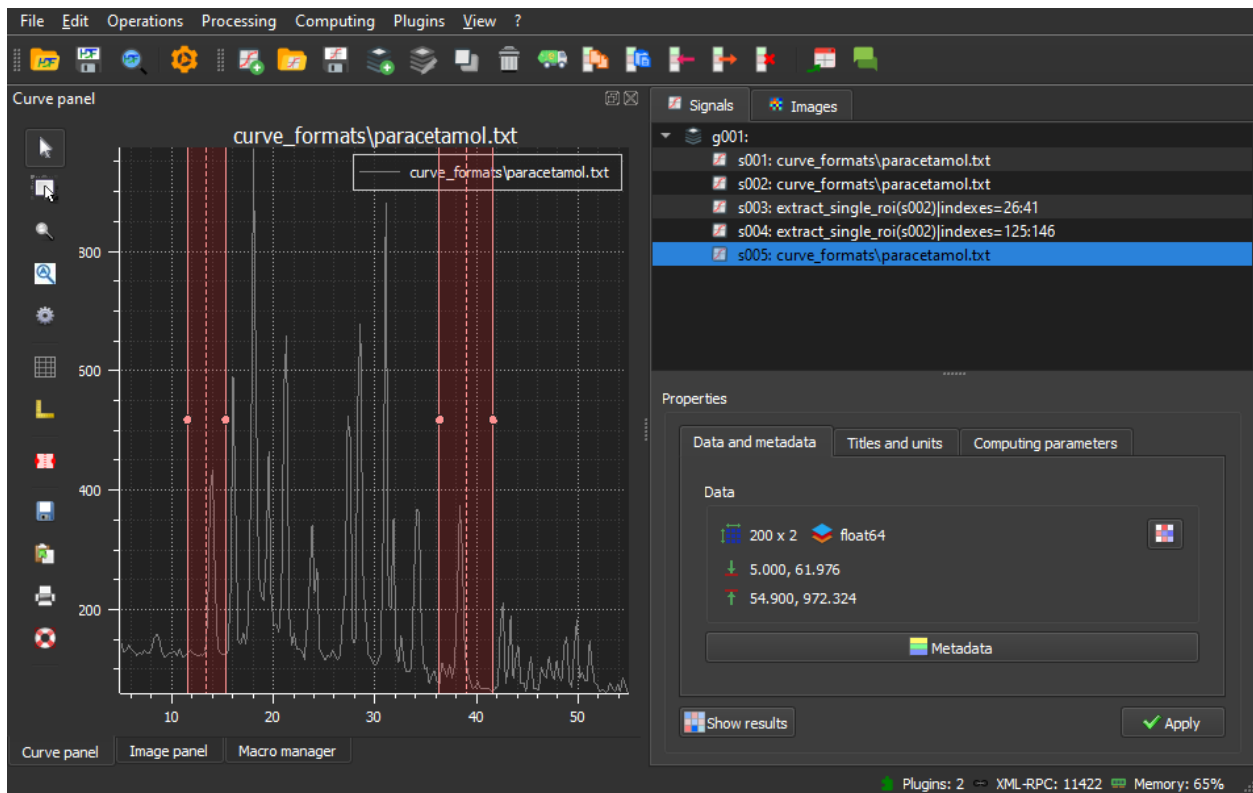


FIG. 4 – Un signal avec une ROI.

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

Format Resize ☒ Background color

Close

FIG. 5 – Exemple de tableau récapitulatif de statistiques : chaque ligne est associée à une ROI (à l'exception de la première qui correspond aux statistiques calculées sur la totalité des données).

Largeur à mi-hauteur

Réalise l'ajustement des données à une gaussienne, une lorentzienne ou une courbe de Voigt en utilisant un algorithme de moindres carrés. Calcule ensuite la largeur à mi-hauteur du modèle d'ajustement.

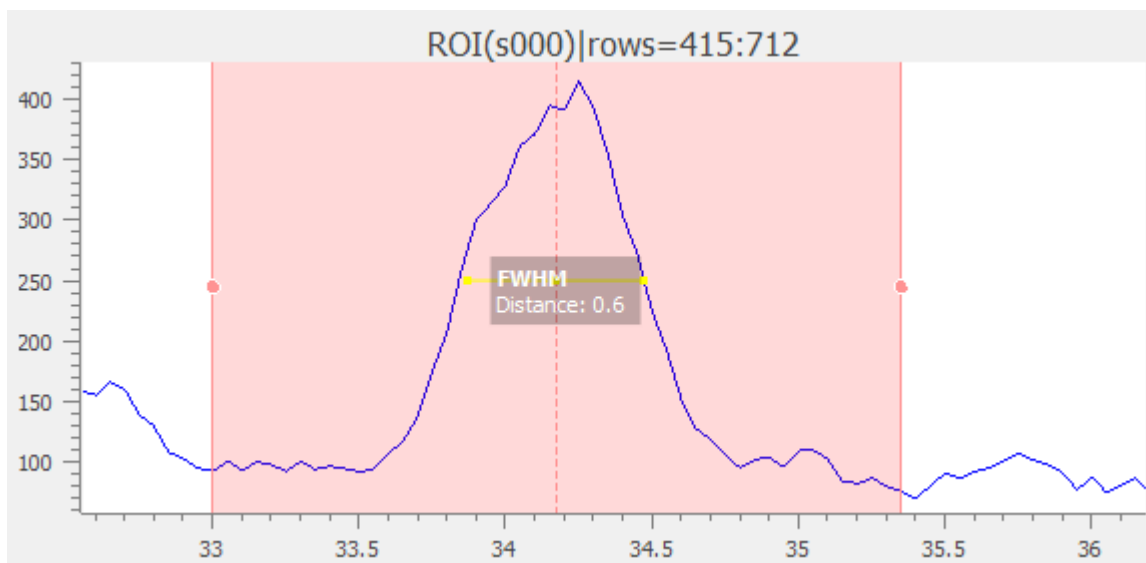


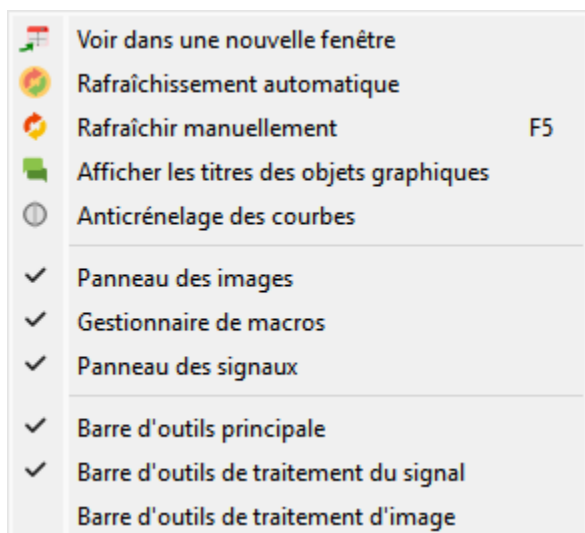
FIG. 6 – Le résultat du calcul est affiché sous la forme d'un segment annoté.

Largeur à $1/e^2$

Réalise l'ajustement des données à une gaussienne en utilisant un algorithme de moindres carrés. Calcule ensuite la largeur à $1/e^2$ du modèle d'ajustement.

Note : Les résultats de calcul scalaires sont systématiquement stockés dans les métadonnées. Les métadonnées sont attachées au signal et sérialisées avec ce dernier par exemple lors de l'export d'une session de DataLab vers un fichier HDF5.

3.6 Menu « Affichage »



Voir dans une nouvelle fenêtre

Ouvre une nouvelle fenêtre pour visualiser les signaux sélectionnés.

Dans cette nouvelle fenêtre, la visualisation des données est plus aisée (p.ex. en maximisant la fenêtre) et des annotations peuvent être ajoutées aux données.

Voir aussi :

Voir *Annotations (Signaux)* pour plus de détails sur les annotations.

Afficher les titres des objets graphiques

Affiche/cache les titres des objets graphiques liés aux résultats de calculs et aux annotations.

Rafraîchissement automatique

Rafraîchit automatiquement la visualisation quand les données changent. Quand ce réglage est activé (par défaut), la visualisation est automatiquement rafraîchie quand les données changent. Quand il est désactivé, la visualisation n'est pas rafraîchie tant que vous ne cliquez pas sur le bouton « Rafraîchir manuellement » dans la barre d'outils. Même si l'algorithme de rafraîchissement est optimisé, il peut prendre du temps pour rafraîchir la visualisation quand les données changent, surtout quand le jeu de données est grand. Par conséquent, vous pouvez désactiver le rafraîchissement automatique quand vous travaillez avec des données volumineuses, et l'activer à nouveau quand vous avez terminé. Cela évitera des rafraîchissements inutiles.

Rafraîchir manuellement

Rafraîchit la visualisation manuellement. Cela déclenche un rafraîchissement de la visualisation, même si le rafraîchissement automatique est désactivé.

Anticrénelage des courbes

Active/désactive l'anticrénelage sur les courbes. L'anticrénelage rend les courbes plus lisses, mais peut aussi les rendre moins nettes.

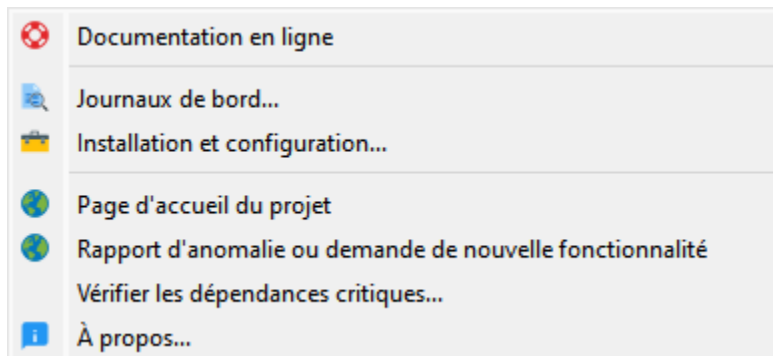
Note : L'anticrénelage est activé par défaut.

Avertissement : L'anticrénelage peut ralentir significativement l'affichage, surtout quand on travaille avec des données volumineuses.

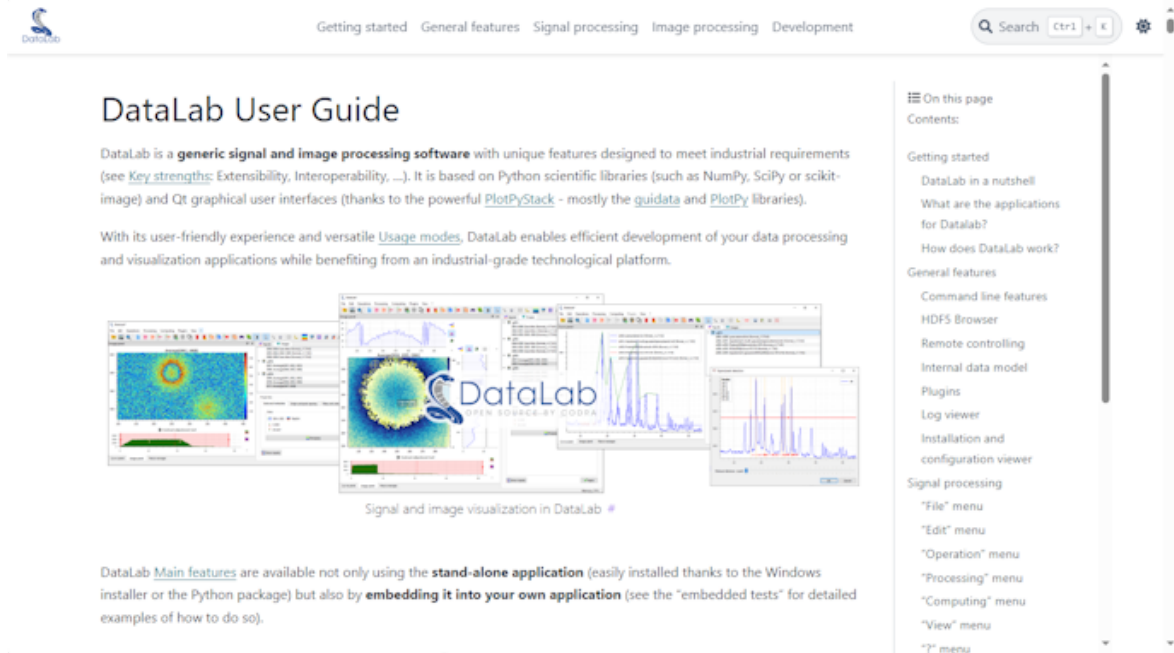
Autres entrées du menu

Affiche/cache les panneaux et barres d'outils.

3.7 Menu « ? »

**Documentation en ligne ou locale**

Affiche la documentation en ligne ou locale (disponible uniquement en anglais pour la version en ligne) :



Afficher les journaux de bords

Affiche l'explorateur de journaux de bord de DataLab

Voir aussi :

Voir la page [Journaux de bord](#) pour plus de détails sur les journaux de bord.

Configuration d'installation de DataLab

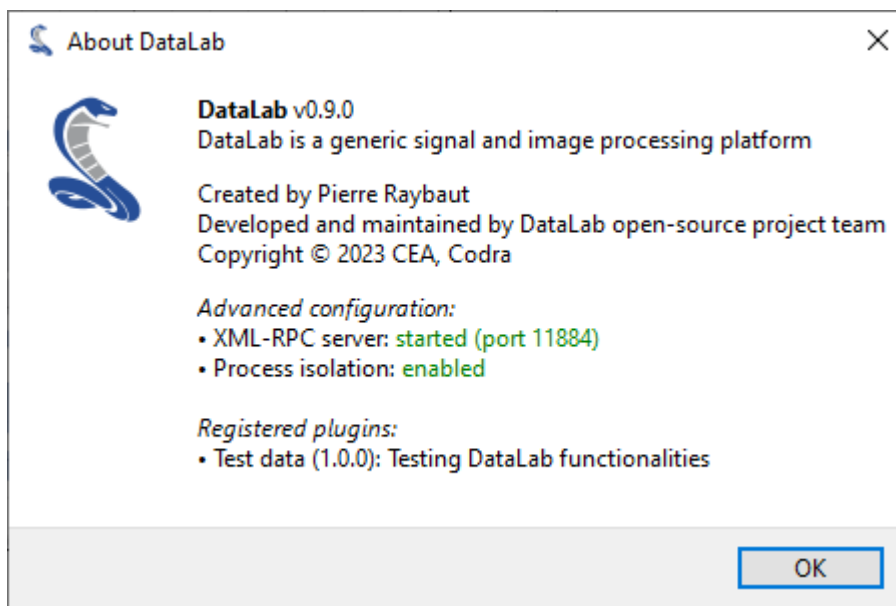
Affiche des informations sur votre configuration d'installation de DataLab (particulièrement utile pour signaler des anomalies de manière efficace).

Voir aussi :

Voir la page [Installation et configuration](#) pour plus de détails sur cette boîte de dialogue.

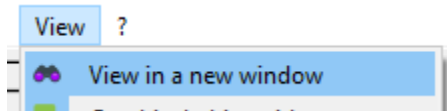
À propos

Affiche la boîte de dialogue « A propos de DataLab »



3.8 Annotations (Signaux)

DataLab dispose d'une fonctionnalité d'annotation pour les signaux (ainsi que pour les images).



La fonctionnalité s'utilise de la façon suivante :

- Créer ou ouvrir un signal dans l'espace de travail de DataLab
- Double-cliquer sur le signal ou sélectionner « Voir dans une nouvelle fenêtre » dans le menu « Affichage »
- Ajouter des annotations (étiquettes, curseurs, rectangles et segments)
- Personnaliser éventuellement les annotations (clic-droit, « Paramètres »)
- Valider vos changements en cliquant sur le bouton « OK »
- A présent, les annotations sont attachées au signal et seront ainsi sauvegardées avec l'espace de travail DataLab.

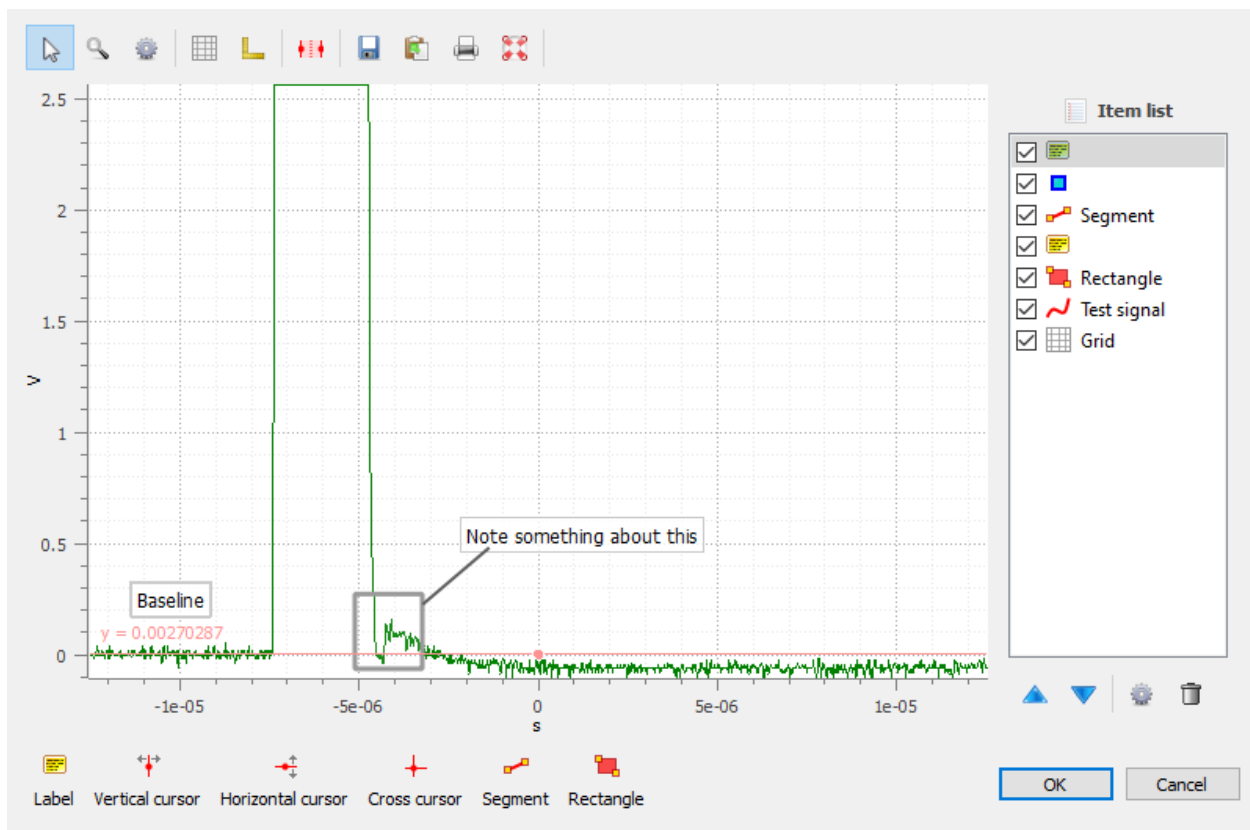
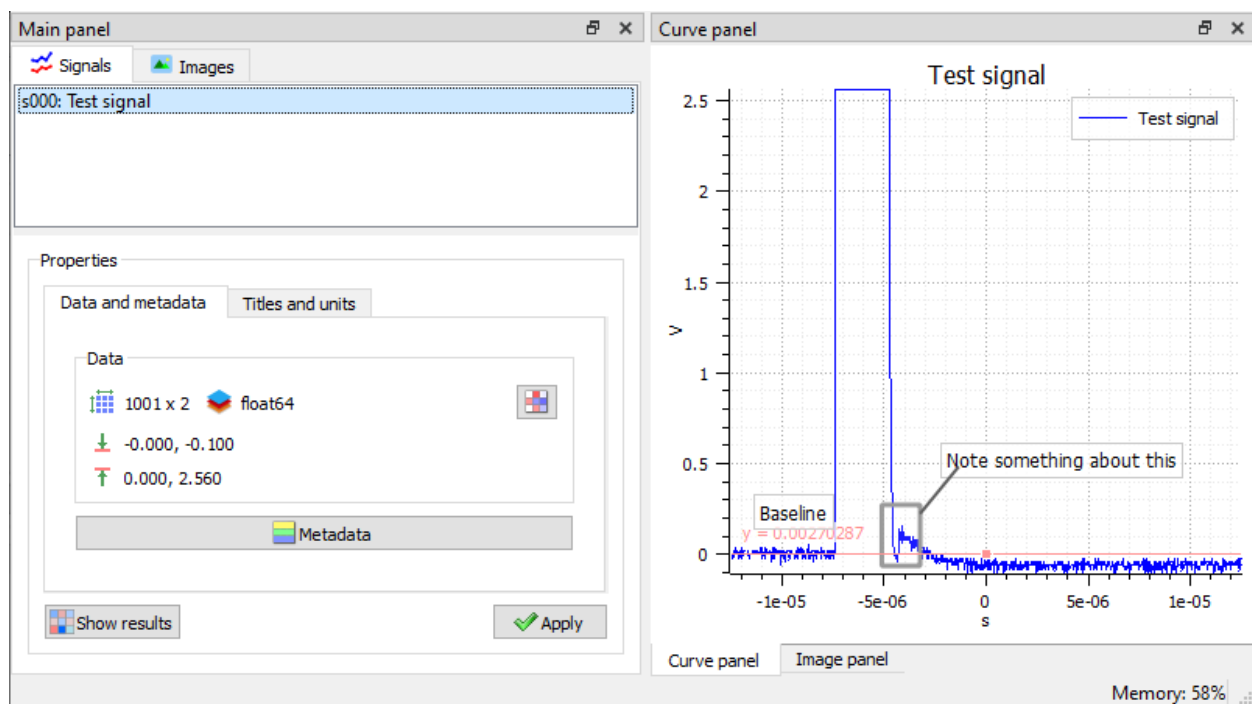


FIG. 7 – Exemple d'annotations.

Une fois que les annotations ont été ajoutées dans la fenêtre séparée (cf. ci-dessus), elles sont intégrées aux métadonnées de l'objet signal (cf. ci-dessous).

Note : Les annotations peuvent être copiées d'un signal à l'autre en utilisant la fonctionnalité de « Copier/coller » des métadonnées.



CHAPITRE 4

Traitement d'image

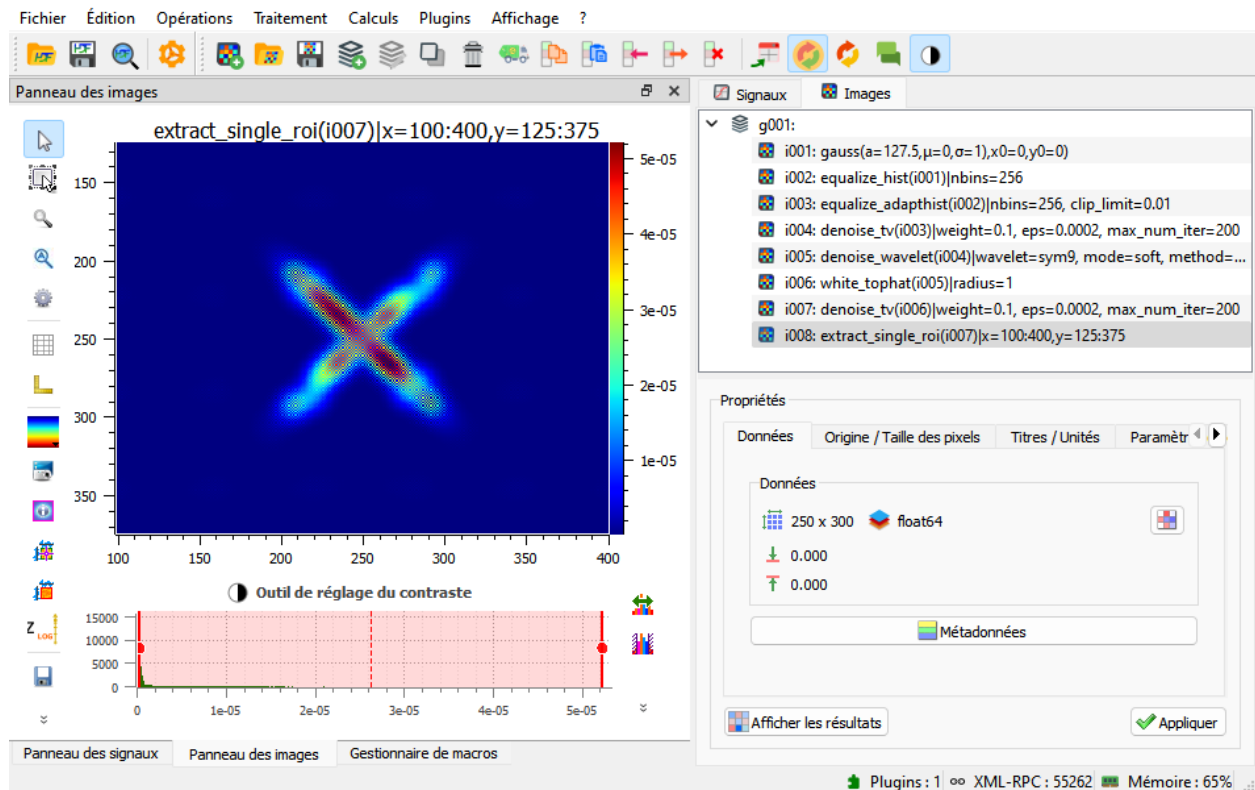
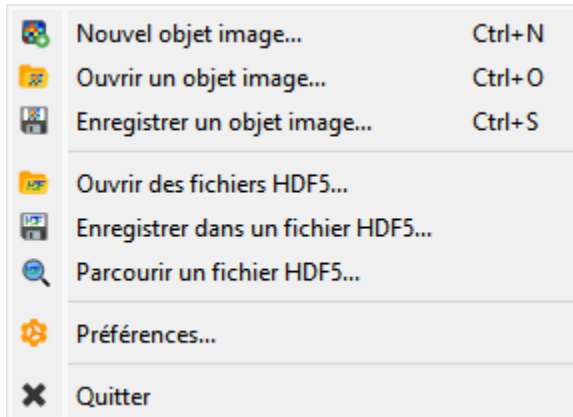


FIG. 1 – Fenêtre principale de DataLab - Traitement d'image

4.1 Menu « Fichier »



Nouvelle image

Crée une image à partir de différents modèles (types de données pris en charge : uint8, uint16, int16, float32, float64) :

Modèle	Equation
Zéros	$z[i] = 0$
Vide	Données mémoire en l'état
Aléatoire	$z[i] \in [0, z_{max}]$ où z_{max} est la valeur maximale correspondant au type de données
Gaussienne 2D	$z = A.exp(-\frac{(\sqrt{(x-x_0)^2 + (y-y_0)^2} - \mu)^2}{2\sigma^2})$

Ouvrir une image

Crée une image depuis l'un des types de fichiers pris en charge :

Type de fichier	Extensions
Fichiers PNG	.png
Fichiers TIFF	.tif, .tiff
Images 8bits	.jpg, .gif
Tableaux NumPy	.npy
Fichiers texte	.txt, .csv, .asc
Fichiers Andor SIF	.sif
Fichiers SPIRICON	.scor-data
Fichiers FXD	.fxd
Images Bitmap	.bmp

Enregistrer l'image

Enregistre l'image sélectionnée dans l'un des types de fichier pris en charge :

Ouvrir un fichier HDF5

Importer les données d'un fichier HDF5.

Enregistrer un fichier HDF5

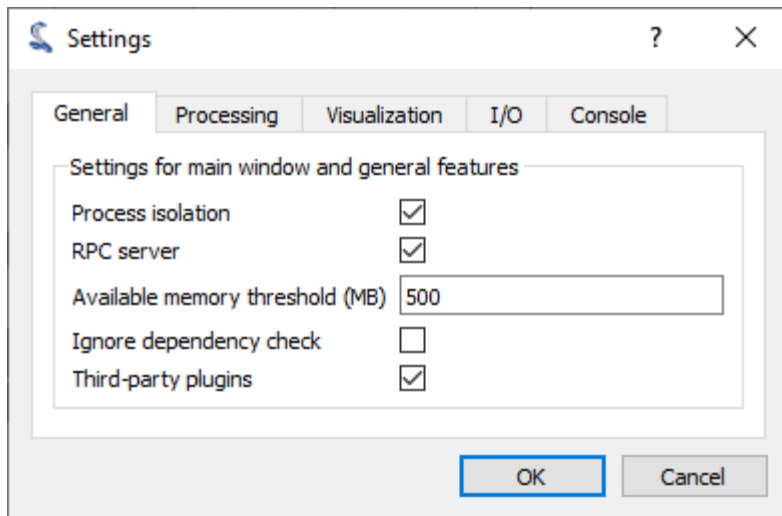
Exporter l'ensemble de la session DataLab (tous les signaux et images) vers un fichier HDF5.

Explorer un fichier HDF5

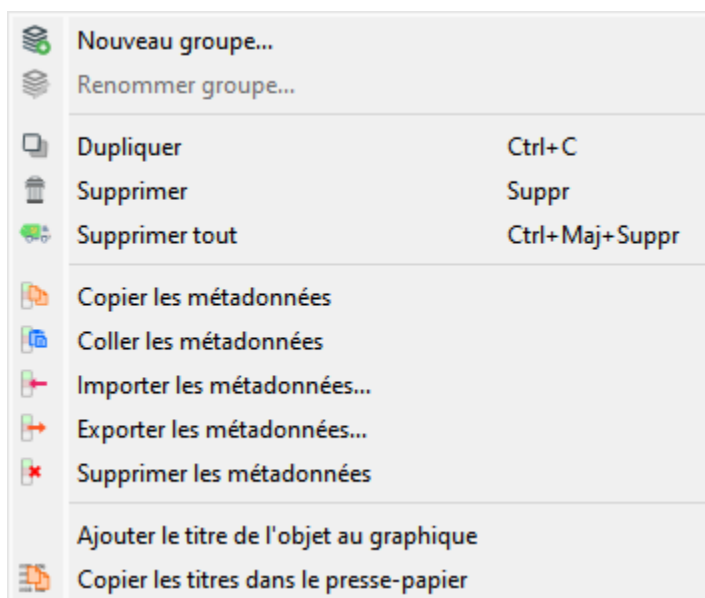
Ouvrir l'*Explorateur HDF5* dans une nouvelle fenêtre pour explorer et éventuellement importer des données depuis un fichier HDF5.

Préférences

Ouvrir la boîte de dialogue « Préférences ».



4.2 Menu « Edition »



Dupliquer

Crée une nouvelle image identique à l'objet sélectionné.

Supprimer

Supprimer l'image sélectionnée.

Supprimer tout

Supprimer toutes les images.

Copier les métadonnées

Copier les métadonnées de l'image sélectionnée vers le presse-papier.

Coller les métadonnées

Coller les métadonnées depuis le presse-papier vers l'image sélectionnée.

Importer les métadonnées dans l'image

Importer les métadonnées depuis un fichier texte au format JSON.

Exporter les métadonnées de l'image

Exporter les métadonnées vers un fichier texte au format JSON.

Supprimer les métadonnées

Supprime les métadonnées de l'image sélectionnée. Les métadonnées contiennent des informations additionnelles telles que les régions d'intérêt ou encore des résultats de calcul.

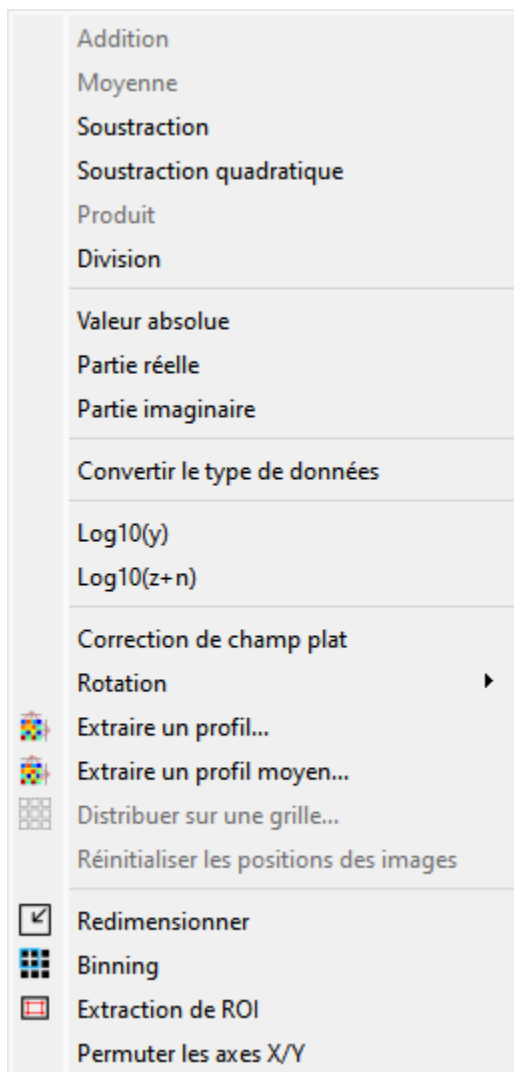
Ajouter le titre de l'objet au graphique

Ajoute le titre de l'image sélectionnée au graphique associé.

Copier les titres dans le presse-papier

Copie les titres de toutes les images dans le presse-papier, sous la forme d'un texte multiligne. Ce texte peut être ensuite utilisé pour reproduire une chaîne de traitement, par exemple.

4.3 Menu « Opérations »

**Addition**

Crée une image à partir de la somme des images sélectionnées :

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

Moyenne

Crée une image à partir de la moyenne des images sélectionnées :

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

Soustraction

Crée une image à partir de la différence des **deux** images sélectionnées :

$$z_2 = z_1 - z_0$$

Soustraction quadratique

Crée une image à partir de la différence quadratique des **deux** images sélectionnées :

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

Produit

Crée une image à partir du produit de toutes les images sélectionnées :

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

Division

Crée une image à partir de la division des **deux** images sélectionnées :

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

Valeur absolue

Crée une image à partir de la valeur absolue de chaque image sélectionnée :

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

Partie réelle

Crée une image à partir de la partie réelle de chaque image sélectionnée :

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

Partie imaginaire

Crée une image à partir de la partie imaginaire de chaque image sélectionnée :

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

Convertir le type de données

Crée une image à partir de la conversion du type de données de l'image sélectionnée.

Note : La conversion du type de données utilise la fonction `numpy.ndarray.astype()` avec les paramètres par défaut (*casting="unsafe"*).

Log10(z)

Crée une image à partir du logarithme base 10 de chaque image sélectionnée :

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

Log10(z+n)

Crée une image à partir du $\text{Log}_{10}(z+n)$ de chaque image sélectionnée (effet d'augmentation de la dynamique d'un logarithme en évitant de calculer $\text{Log}_{10}(0)$ sur l'arrière-plan de l'image) :

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

Correction de champ plat

Calcule la correction de champ plat à partir des **deux** images sélectionnées :

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

où z_0 est l'image brute, z_f est l'image d'homogénéité, $z_{threshold}$ est un seuil ajustable et \bar{z}_f est la valeur moyenne de l'image d'homogénéité :

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

Note : L'image brute et l'image d'homogénéité sont supposées avoir déjà été corrigées par soustraction d'image de noir.

Rotation

Crée une image qui est le résultat de la rotation (90° , 270° ou angle arbitraire) ou de l'inversion (horizontale ou verticale) des données de l'image sélectionnée.

Extraire un profil

Extraire un profil horizontal ou vertical de chaque image sélectionnée et créer un nouveau signal à partir de ces profils.

Extraire un profil moyen

Extraire un profil horizontal ou vertical moyenné sur une zone rectangulaire de chaque image sélectionnée et créer un nouveau signal à partir de ces profils.

Distribuer sur une grille

Distribuer les images sélectionnées sur une grille régulière.

Réinitialiser les positions

Réinitialiser les positions des images sélectionnées sur les coordonnées (x0, y0) de la première image.

Redimensionner

Crée une image qui est le résultat du redimensionnement de chaque image sélectionnée.

Binning

Regroupe des pixels adjacents de l'image en un seul pixel (somme, moyenne, médiane, minimum ou maximum de la valeur des pixels adjacents).

Extraction de ROI

Crée une image à partir d'une région d'intérêt (ROI) définie par l'utilisateur.

Permuter les axes X/Y

Crée une image à partir des données inversées X/Y de l'image sélectionnée.

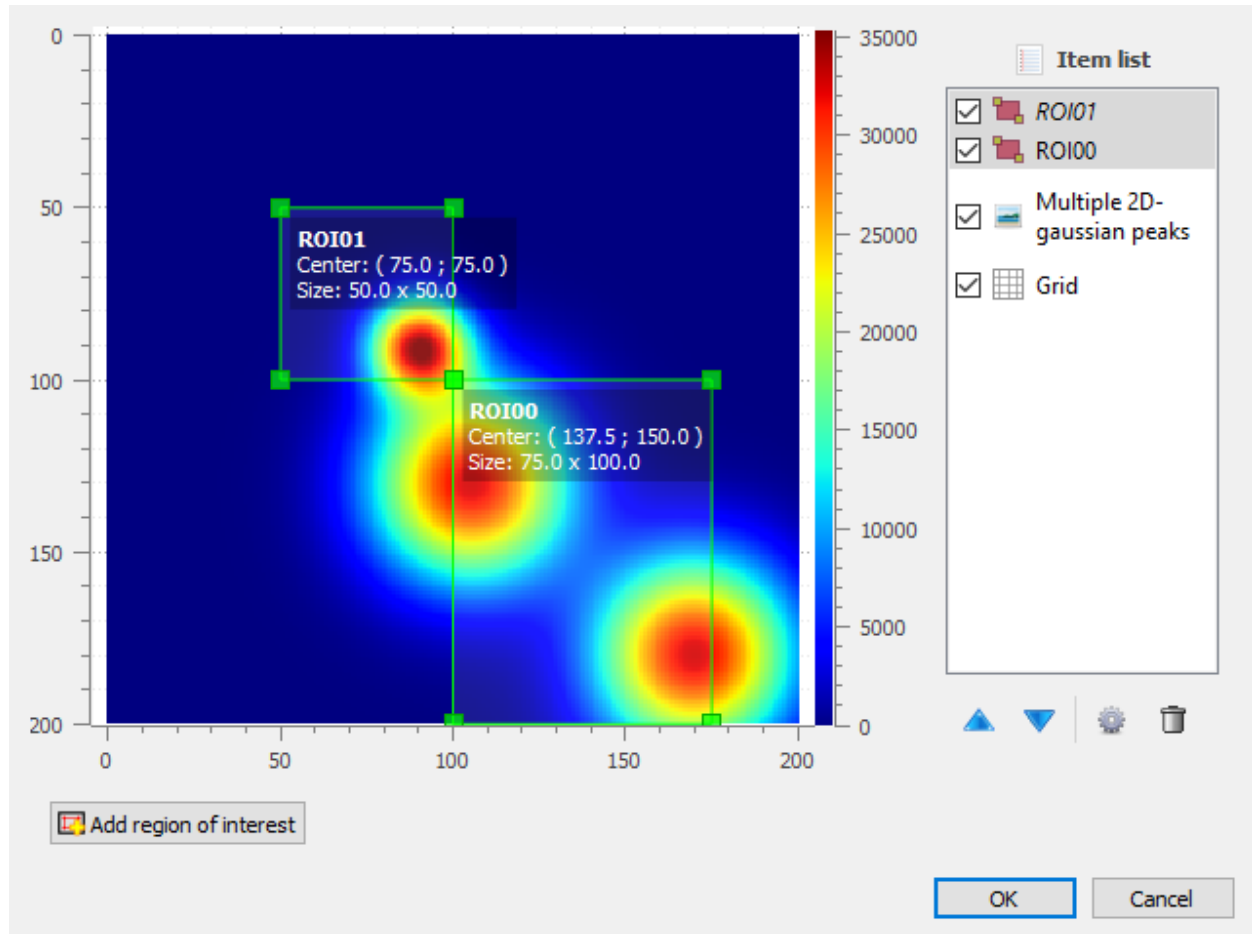
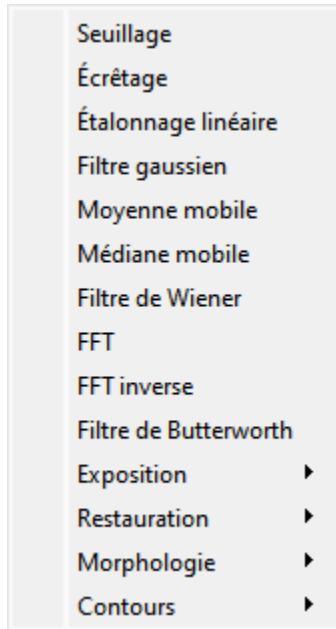


FIG. 2 – Boîte de dialogue d'extraction de ROI : la région d'intérêt (ROI) est définie en ajustant la position et la taille du rectangle de sélection.

4.4 Menu « Traitement »



Étalonnage linéaire

Crée une image à partir de l'étalonnage linéaire (par rapport à l'axe des Z) de chaque image sélectionnée.

Paramètre	Étalonnage linéaire
Axe des Z	$z_1 = a.z_0 + b$

Seuillage

Applique un seuillage sur chaque image sélectionnée.

Ecrêtage

Applique un écrêtage sur chaque image sélectionnée.

Moyenne mobile

Calcule le résultat de la moyenne mobile de chaque image sélectionnée (implémentation basée sur `scipy.ndimage.uniform_filter`).

Médiane mobile

Calcule le résultat de la médiane mobile de chaque image sélectionnée (implémentation basée sur `scipy.signal.medfilt`).

Filtre de Wiener

Calcule le résultat du filtre de Wiener sur chaque image sélectionnée (implémentation basée sur `scipy.signal.wiener`).

FFT

Crée une image à partir de la transformée de Fourier rapide (FFT) de chaque image sélectionnée.

FFT inverse

Crée une image à partir de la transformée de Fourier rapide inverse (FFT inverse) de chaque image sélectionnée.

Filtre de Butterworth

Calcule le résultat d'un filtre de Butterworth sur l'image (implémentation basée sur `skimage.filters.butterworth`)

Exposition

Correction gamma

Applique une correction gamma sur chaque image sélectionnée (implémentation basée sur `skimage.exposure.adjust_gamma`)

Correction logarithmique

Applique une correction logarithmique sur chaque image sélectionnée (implémentation basée sur `skimage.exposure.adjust_log`)

Correction sigmoïde

Applique une correction sigmoïde sur chaque image sélectionnée (implémentation basée sur `skimage.exposure.adjust_sigmoid`)

Egalisation d'histogramme

Egalise les niveaux de l'histogramme de l'image (implémentation basée sur `skimage.exposure.equalize_hist`)

Egalisation d'histogramme adaptative

Egalise les niveaux de l'histogramme de l'image, en utilisant l'algorithme Contrast Limited Adaptive Histogram Equalization (CLAHE) (implémentation basée sur `skimage.exposure.equalize_adapthist`)

Ajustement des niveaux

Réduit ou étend la plage de répartition des niveaux de l'image (implémentation basée sur `skimage.exposure.rescale_intensity`)

Restauration**Filtrage variationnel (débruitage)**

Calcule le résultat d'un débruitage par filtrage variationnel selon l'algorithme de Chambolle (implémentation basée sur `skimage.restoration.denoise_tv_chambolle`)

Filtrage bilatéral (débruitage)

Calcule le résultat d'un débruitage par filtrage bilatéral (implémentation basée sur `skimage.restoration.denoise_bilateral`)

Débruitage par ondelettes

Calcule le résultat d'un débruitage par ondelettes (implémentation basée sur `skimage.restoration.denoise_wavelet`)

Débruitage par Top-Hat

Débruitage de l'image par soustraction de la transformée Top-Hat avec ouverture circulaire de rayon paramétrable

Tous les débruitages

Applique tous les débruitages à l'image. Combiné avec l'option « distribuer sur une grille », cela permet de comparer les différents débruitages sur la même image.

Morphologie**Top-Hat (disque)**

Calcule le résultat d'un Top-Hat de l'image, avec une ouverture circulaire (disque) de rayon paramétrable (implémentation basée sur `skimage.restoration.denoise_wavelet`)

Top-Hat dual (disque)

Calcule le résultat d'un Top-Hat dual de l'image, avec une ouverture circulaire (disque) de rayon paramétrable (implémentation basée sur `skimage.restoration.denoise_wavelet`)

Erosion (disque)

Calcule le résultat d'une érosion de l'image, avec une ouverture circulaire (disque) de rayon paramétrable (implémentation basée sur `skimage.morphology.erosion`)

Dilatation (disque)

Calcule le résultat d'une dilatation de l'image, avec une ouverture circulaire (disque) de rayon paramétrable (implémentation basée sur `skimage.morphology.dilation`)

Ouverture (disque)

Calcule le résultat d'une ouverture morphologique de l'image, avec une forme circulaire (disque) de rayon paramétrable (implémentation basée sur `skimage.morphology.opening`)

Fermeture (disque)

Calcule le résultat d'une fermeture morphologique de l'image, avec une forme circulaire (disque) de rayon paramétrable (implémentation basée sur `skimage.morphology.closing`)

Toutes les opérations morphologiques

Applique toutes les opérations morphologiques à une image. Combiné avec l'option « distribuer sur une grille », cela permet de comparer les différentes opérations morphologiques sur la même image.

Contours

Filtre de Roberts

Calcule le résultat d'une détection de contours à l'aide l'algorithme de Roberts (implémentation basée sur `skimage.filters.roberts`)

Filtre de Prewitt

Calcule le résultat d'une détection de contours à l'aide l'algorithme de Prewitt (implémentation basée sur `skimage.filters.prewitt`)

Filtre de Prewitt (horizontal)

Recherche les contours horizontaux d'une image, à l'aide de l'algorithme de Prewitt (implémentation basée sur `skimage.filters.prewitt_h`)

Filtre de Prewitt (vertical)

Recherche les contours verticaux d'une image, à l'aide de l'algorithme de Prewitt (implémentation basée sur `skimage.filters.prewitt_v`)

Filtre de Sobel

Calcule le résultat d'une détection de contours à l'aide l'algorithme de Sobel (implémentation basée sur `skimage.filters.sobel`)

Filtre de Sobel (horizontal)

Recherche les contours horizontaux d'une image, à l'aide de l'algorithme de Sobel (implémentation basée sur `skimage.filters.sobel_h`)

Filtre de Sobel (vertical)

Recherche les contours verticaux d'une image, à l'aide de l'algorithme de Sobel (implémentation basée sur `skimage.filters.sobel_v`)

Filtre de Scharr

Calcule le résultat d'une détection de contours à l'aide l'algorithme de Scharr (implémentation basée sur `skimage.filters.scharr`)

Filtre de Scharr (horizontal)

Recherche les contours horizontaux d'une image, à l'aide de l'algorithme de Scharr (implémentation basée sur `skimage.filters.scharr_h`)

Filtre de Scharr (vertical)

Recherche les contours verticaux d'une image, à l'aide de l'algorithme de Scharr (implémentation basée sur `skimage.filters.scharr_v`)

Filtre de Farid

Calcule le résultat d'une détection de contours à l'aide l'algorithme de Farid (implémentation basée sur `skimage.filters.farid`)

Filtre de Farid (horizontal)

Recherche les contours horizontaux d'une image, à l'aide de l'algorithme de Farid (implémentation basée sur `skimage.filters.farid_h`)

Filtre de Farid (vertical)

Recherche les contours verticaux d'une image, à l'aide de l'algorithme de Farid (implémentation basée sur `skimage.filters.farid_v`)

Filtre de Laplace

Calcule le résultat d'une détection de contours à l'aide l'algorithme de Laplace (implémentation basée sur `skimage.filters.laplace`)

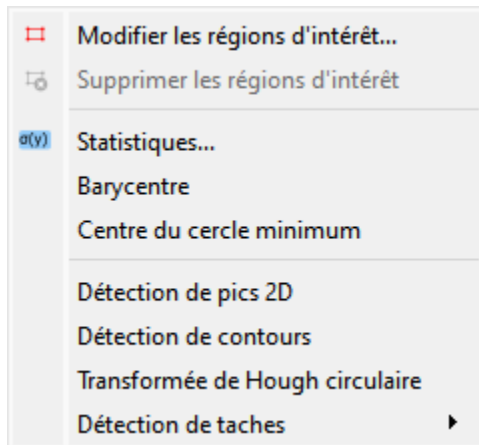
Tous les filtres de contours

Applique tous les algorithmes de détection de contours (voir ci-dessus) à une image. Combiné avec l'option « distribuer sur une grille », cela permet de comparer les différents filtres de contours sur la même image.

Filtre de Canny

Calcule le résultat d'une détection de contours à l'aide l'algorithme de Canny (implémentation basée sur `skimage.feature.canny`)

4.5 Menu « Calculs »



Modifier les régions d'intérêt

Ouvre une boîte de dialogue pour définir des régions d'intérêt (ROI) multiples. Les ROI sont stockées sous la forme de métadonnées ; elles sont donc attachées à l'image.

La boîte de dialogue de définition de ROI est identique à celle utilisée pour l'extraction de ROI (voir plus haut).

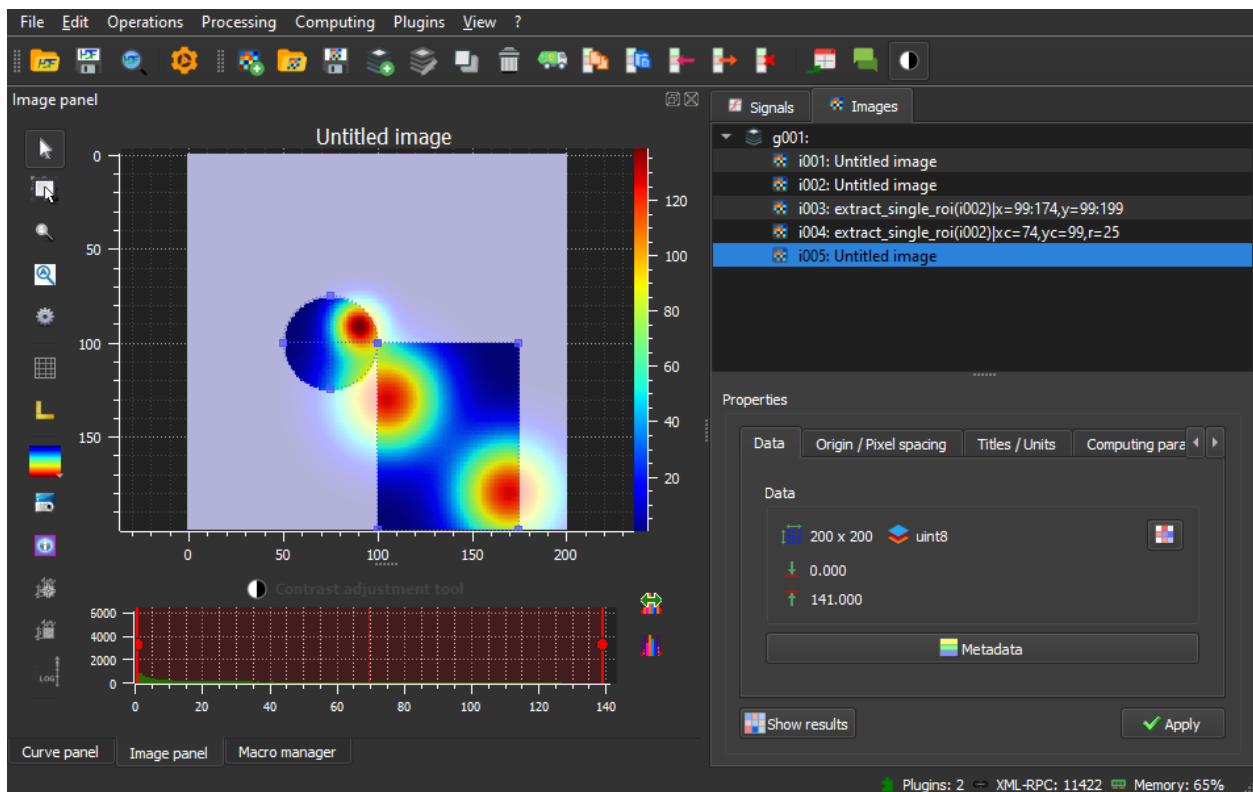


FIG. 3 – Une image avec une ROI.

Supprimer les régions d'intérêt

Supprimer toutes les ROI définies pour l'objet ou les objets sélectionné(s).

Statistiques

Calcule des statistiques sur les images sélectionnées et affiche un tableau récapitulatif.

	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 – Exemple de tableau récapitulatif de statistiques : chaque ligne est associée à une ROI (à l'exception de la première qui correspond aux statistiques calculées sur la totalité des données).

Barycentre

Calcule le barycentre en utilisant une méthode basée sur la transformée de Fourier (telle que décrite dans [Weisshaar et al.](#)). Cette méthode présente l'avantage d'être peu sensible au bruit de fond.

Centre du cercle minimum

Calcule le contour circulaire entourant les valeurs de l'image au-delà d'un seuil (moitié du maximum de l'image).

Détection de pics 2D

Détecte automatiquement des pics sur une image en utilisant un algorithme basé sur des filtres minimum-maximum.

Voir aussi :

Voir [Détection de pics 2D](#) pour plus de détails sur l'algorithme et les paramètres associés.

Détection de contours

Détecte automatiquement les contours et ajuste ces derniers par des cercles ou des ellipses, ou les représente directement par des polygones.

Voir aussi :

Voir [Détection de contours](#) pour plus de détails sur l'algorithme et les paramètres associés.

Note : Les résultats de calcul scalaires sont systématiquement stockés dans les métadonnées. Les métadonnées sont attachées à l'image et sérialisées avec cette dernière par exemple lors de l'export d'une session de DataLab vers un fichier HDF5.

Transformée de Hough circulaire

Détection de formes circulaires à partir d'une transformée de Hough (implémentation basée sur [skimage.transform.hough_circle_peaks](#))

Détection de taches

Détection de taches (DOG)

Détection de taches basée sur la méthode de différence de gaussienne (DOG) (implémentation basée sur [skimage.feature.blob_dog](#)).

Détection de taches (hessien)

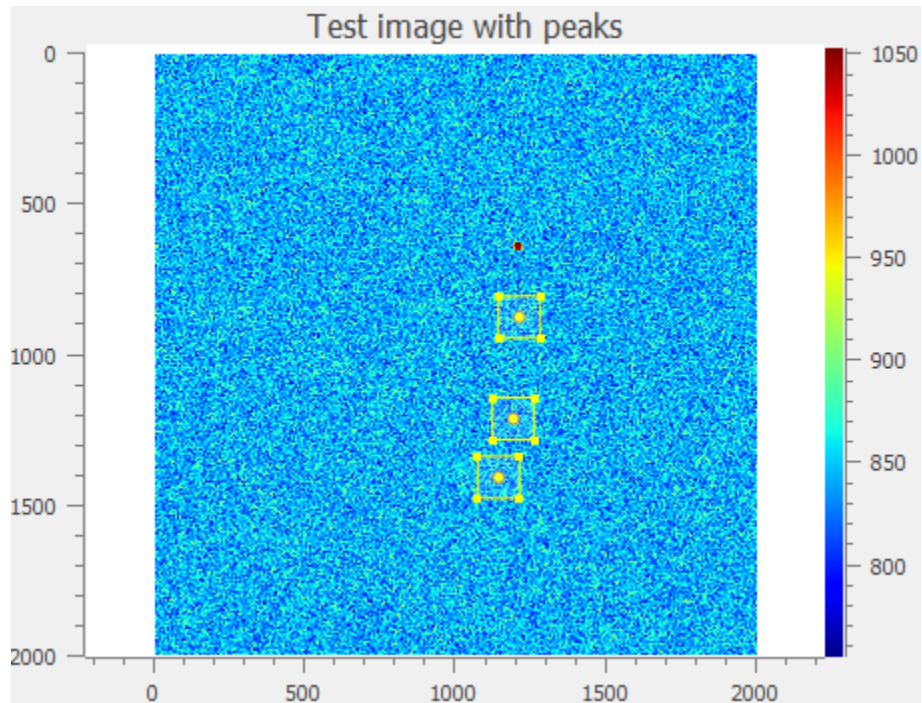


FIG. 5 – Exemple de détection de pics 2D.

Détection de taches basée sur la méthode du discriminant hessien (implementation basée sur `skimage.feature.blob_doh`).

Détection de taches (LOG)

Détection de taches basée sur la méthode du laplacien de gaussienne (LOG) (implementation basée sur `skimage.feature.blob_log`).

Détection de taches (OpenCV)

Détection de taches basée sur l'implémentation OpenCV de `SimpleBlobDetector`.

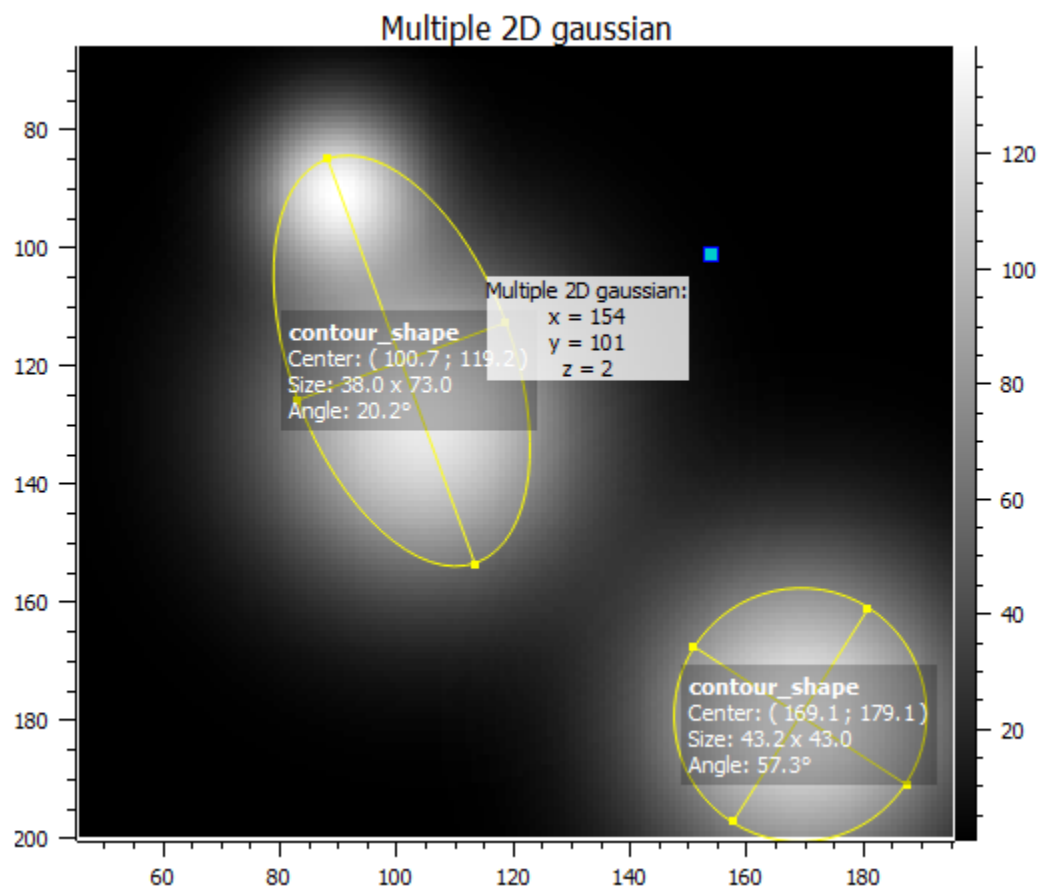
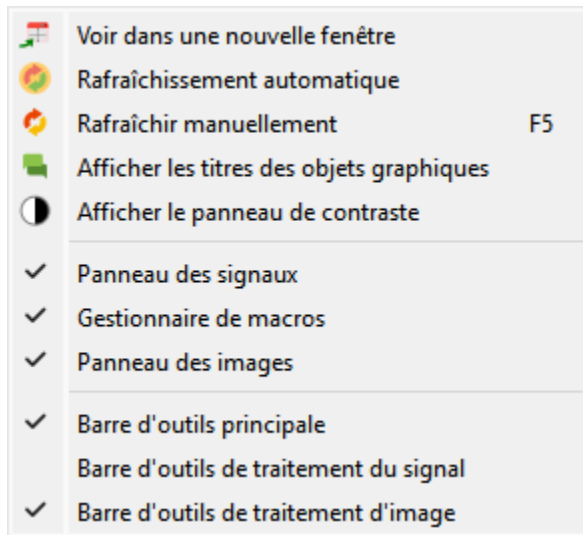


FIG. 6 – Exemple de détection de contours.

4.6 Menu « Affichage »



Voir dans une nouvelle fenêtre

Ouvre une nouvelle fenêtre pour visualiser les images sélectionnées.

Dans cette nouvelle fenêtre, la visualisation des données est plus aisée (p.ex. en maximisant la fenêtre) et des annotations peuvent être ajoutées aux données.

Voir aussi :

Voir *Annotations (Images)* pour plus de détails sur les annotations.

Afficher les titres des objets graphiques

Affiche/cache les titres des objets graphiques liés aux résultats de calculs et aux annotations.

Rafraîchissement automatique

Rafraîchit automatiquement la visualisation quand les données changent. Quand ce réglage est activé (par défaut), la visualisation est automatiquement rafraîchie quand les données changent. Quand il est désactivé, la visualisation n'est pas rafraîchie tant que vous ne cliquez pas sur le bouton « Rafraîchir manuellement » dans la barre d'outils. Même si l'algorithme de rafraîchissement est optimisé, il peut prendre du temps pour rafraîchir la visualisation quand les données changent, surtout quand le jeu de données est grand. Par conséquent, vous pouvez désactiver le rafraîchissement automatique quand vous travaillez avec des données volumineuses, et l'activer à nouveau quand vous avez terminé. Cela évitera des rafraîchissements inutiles.

Rafraîchir manuellement

Rafraîchit la visualisation manuellement. Cela déclenche un rafraîchissement de la visualisation, même si le rafraîchissement automatique est désactivé.

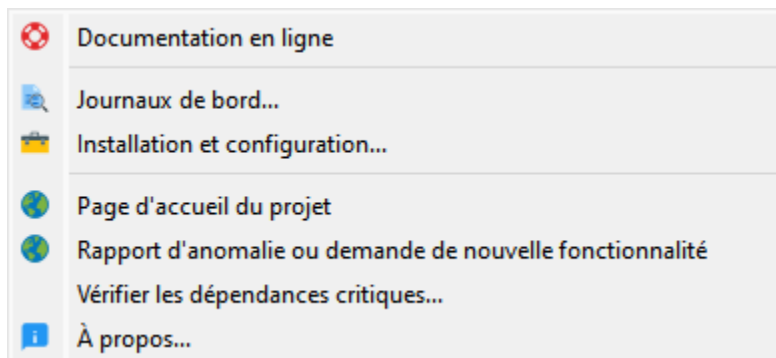
Afficher le panneau de contraste

Affiche/cache le panneau de réglage du contraste.

Autres entrées du menu

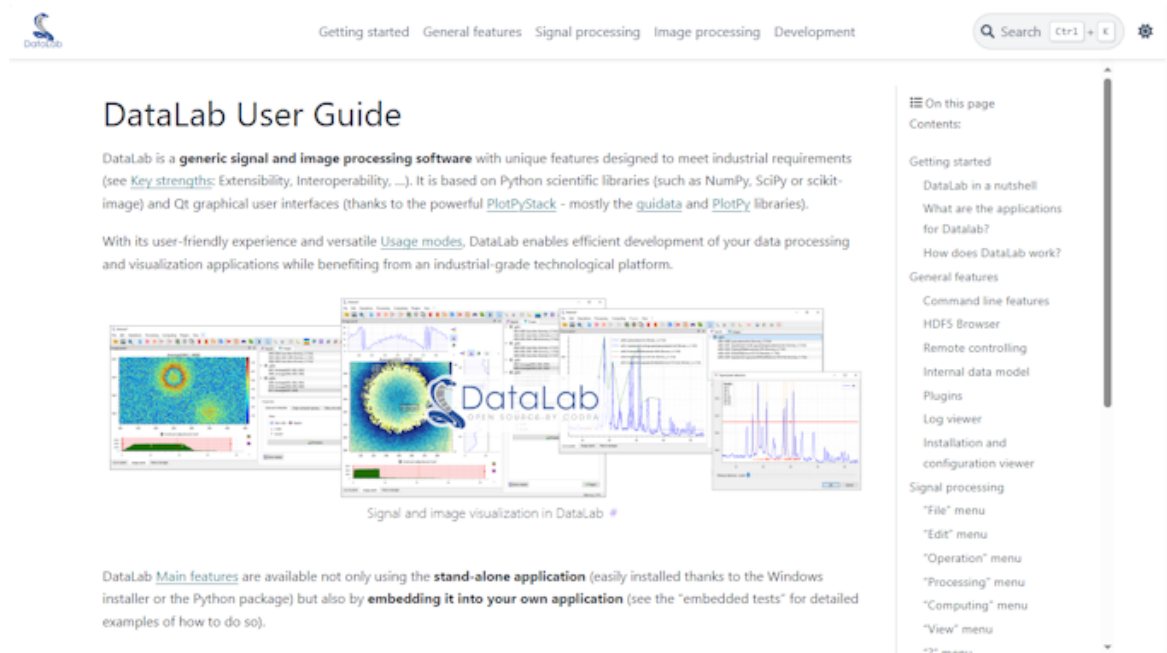
Affiche/cache les panneaux et barres d'outils.

4.7 Menu « ? »



Documentation en ligne ou locale

Affiche la documentation en ligne ou locale (disponible uniquement en anglais pour la version en ligne) :



Afficher les journaux de bords

Affiche l'explorateur de journaux de bord de DataLab

Voir aussi :

Voir la page [Journaux de bord](#) pour plus de détails sur les journaux de bord.

Configuration d'installation de DataLab

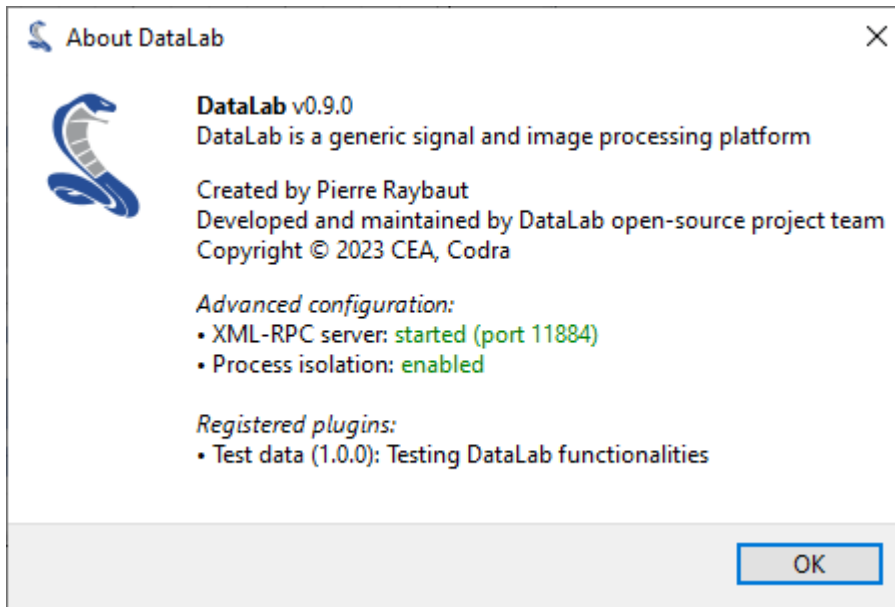
Affiche des informations sur votre configuration d'installation de DataLab (particulièrement utile pour signaler des anomalies de manière efficace).

Voir aussi :

Voir la page [Installation et configuration](#) pour plus de détails sur cette boîte de dialogue.

À propos

Affiche la boîte de dialogue « A propos de DataLab »



4.8 Détection de pics 2D

DataLab fournit une fonctionnalité de « Détection de pics 2D » qui est basée sur un algorithme de filtrage minimum-maximum.

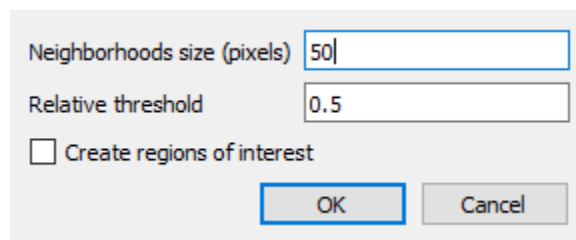


FIG. 7 – Paramètres de détection de pics 2D.

La fonctionnalité s'utilise de la façon suivante :

- Créer ou ouvrir une image dans l'espace de travail de DataLab
- Sélectionner « Détection de pics 2D » dans le menu « Calculs »
- Saisir les paramètres « Taille de voisinage » et « Seuil relatif »
- Cocher « Créer des régions d'intérêt » si vous souhaitez que des ROI soient définies pour chaque pic détecté (ce qui peut s'avérer utile en cas de calcul ultérieur sur chaque pic détecté, tel que par exemple une détection de contour).

Les résultats sont affichés dans un tableau :

- Chaque ligne est associée à un pic détecté
- La première colonne contient l'indice de la ROI (0 si aucune ROI n'est définie)
- Les deuxième et troisième colonnes contiennent les coordonnées des pics

L'algorithme de détection de pics 2D fonctionne de la manière suivante :

- Tout d'abord, des images filtrées minimum-maximum sont calculées en utilisant un algorithme de fenêtre glissante dont la taille est définie par l'utilisateur (implémentation basée sur `scipy.ndimage.minimum_filter` et `scipy.ndimage.maximum_filter`)

Results - NumPy array (read only)			
	ROI	x	y
Peaks(i000)	0	1366	638
Peaks(i000)	0	1416	1069
Peaks(i000)	0	1018	1135
Peaks(i000)	0	828	1229

Format Resize ☒ Background color

Close

FIG. 8 – Résultats d'une détection de pics 2D (voir le test « peak2d_app.py »)

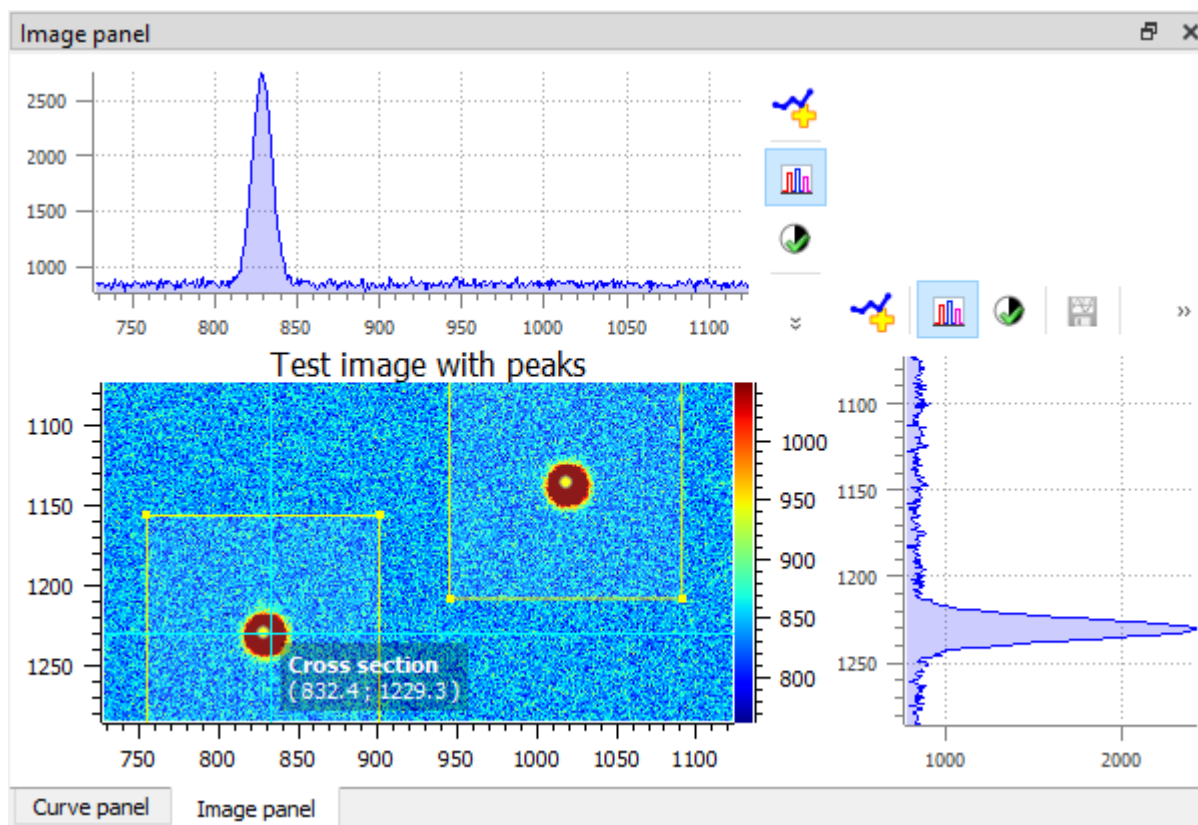


FIG. 9 – Exemple de détection de pics 2D.

- Ensuite, la différence entre ces deux images filtrées est ecrêtée à une valeur correspondant à un seuil défini par l'utilisateur
- L'image résultante est étiquetée en utilisant `scipy.ndimage.label`
- Les coordonnées des pics sont ensuite obtenues en calculant le centre de chaque étiquette
- Les doublons sont éventuellement supprimés

Les paramètres de la détection de pics 2D sont les suivants :

- « Taille de voisinage » : taille de la fenêtre glissante (cf. plus haut)
- « Seuil relatif » : seuil de détection

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

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

    If neighborhoods size is None, default value is the highest value
    between 50 pixels and the 1/40th of the smallest image dimension.

    Detection threshold level is relative to difference
    between data maximum and minimum values.

    Args:
        data (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 Détection de contours

DataLab fournit une fonctionnalité de « Détection de contours » qui est basée sur l'algorithme des [marching cubes](#)

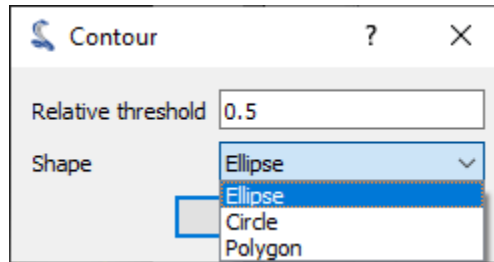


FIG. 10 – Paramètres de détection de contours.

La fonctionnalité s'utilise de la façon suivante :

- Créer ou ouvrir une image dans l'espace de travail de DataLab
- Créer éventuellement une ROI autour de la zone cible de l'image
- Sélectionner « Détection de contours » dans le menu « Calculs »
- Saisir le paramètre « Forme » (« Ellipse », « Cercle » ou « Polygone »)

	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

FIG. 11 – Résultats de la détection de contours (cf. test « contour_app.py »)

Les résultats sont affichés dans un tableau :

- Chaque ligne est associée à un contour
- La première colonne contient l'indice de la ROI (0 si aucune ROI n'est définie)
- Les colonnes suivantes présentent les coordonnées des contours : 4 colonnes pour les cercles (coordonnées du diamètre) et 8 colonnes pour les ellipses (coordonnées des diamètres)

L'algorithme de détection de contours fonctionne de la manière suivante :

- Tout d'abord, les isocontours sont calculés (l'implémentation est basée sur [skimage.measure.find_contours.find_contours](#))
- Ensuite, chaque contour est ajusté à une ellipse (ou à un cercle)

La fonctionnalité est basée sur la fonction `get_contour_shapes` du module `cdl.core.computation` :

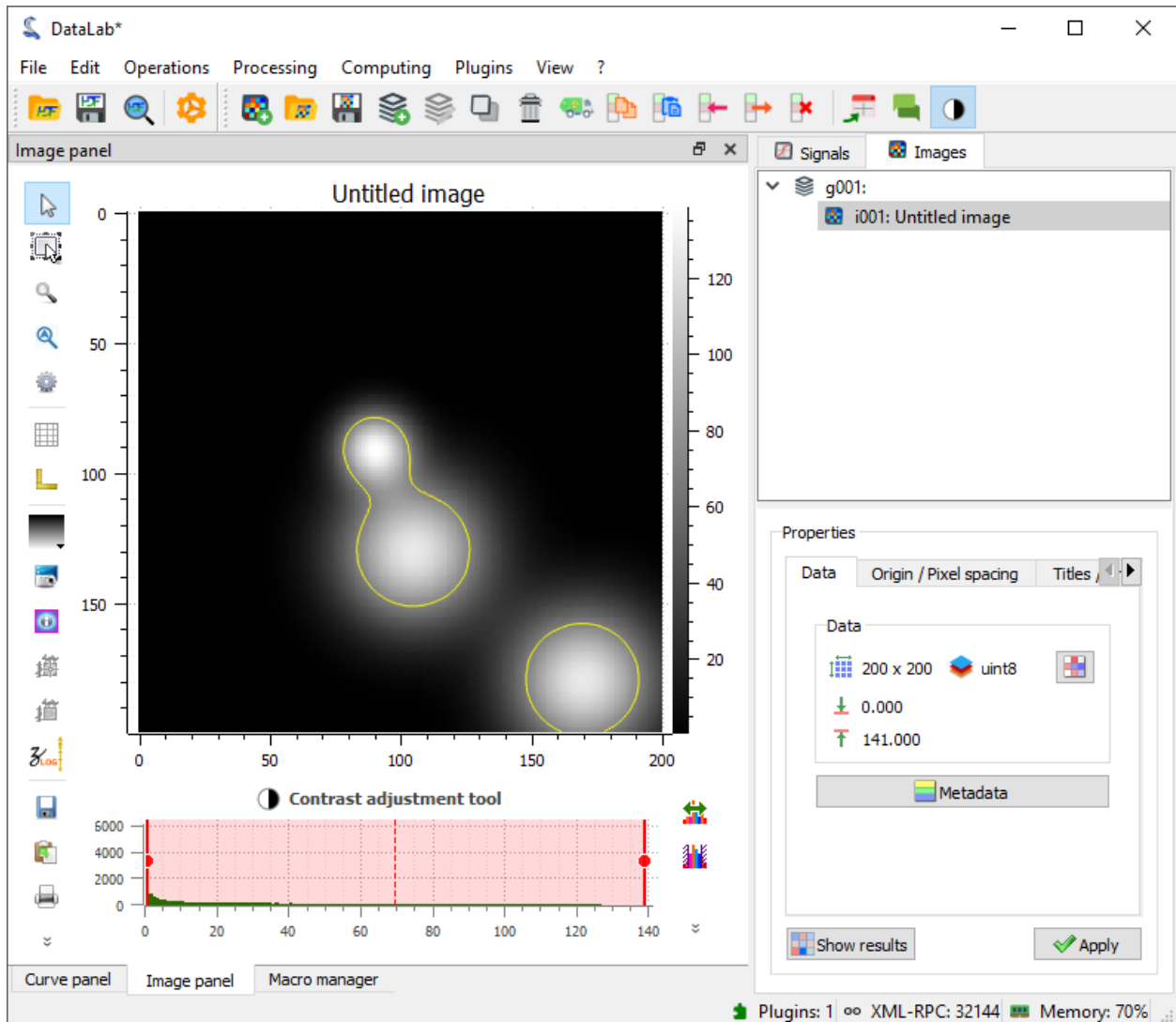


FIG. 12 – Exemple de détection de contours.

```

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')
        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)

```

(suite sur la page suivante)

(suite de la page précédente)

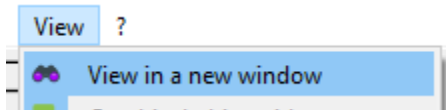
```

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 dispose d'une fonctionnalité d'annotation pour les images (ainsi que pour les signaux).



La fonctionnalité s'utilise de la façon suivante :

- Créer ou ouvrir une image dans l'espace de travail de DataLab
- Double-cliquer sur l'image ou sélectionner « Voir dans une nouvelle fenêtre » dans le menu « Affichage »
- Ajouter des annotations (étiquettes, rectangles, cercles, etc.)
- Personnaliser éventuellement les annotations (clic-droit, « Paramètres »)
- Valider vos changements en cliquant sur le bouton « OK »
- A présent, les annotations sont attachées à l'image et seront ainsi sauvegardées avec l'espace de travail DataLab.

Une fois que les annotations ont été ajoutées dans la fenêtre séparée (cf. ci-dessus), elles sont intégrées aux métadonnées de l'objet image (cf. ci-dessous).

Note : Les annotations peuvent être copiées d'une image à l'autre en utilisant la fonctionnalité de « Copier/coller » des métadonnées.

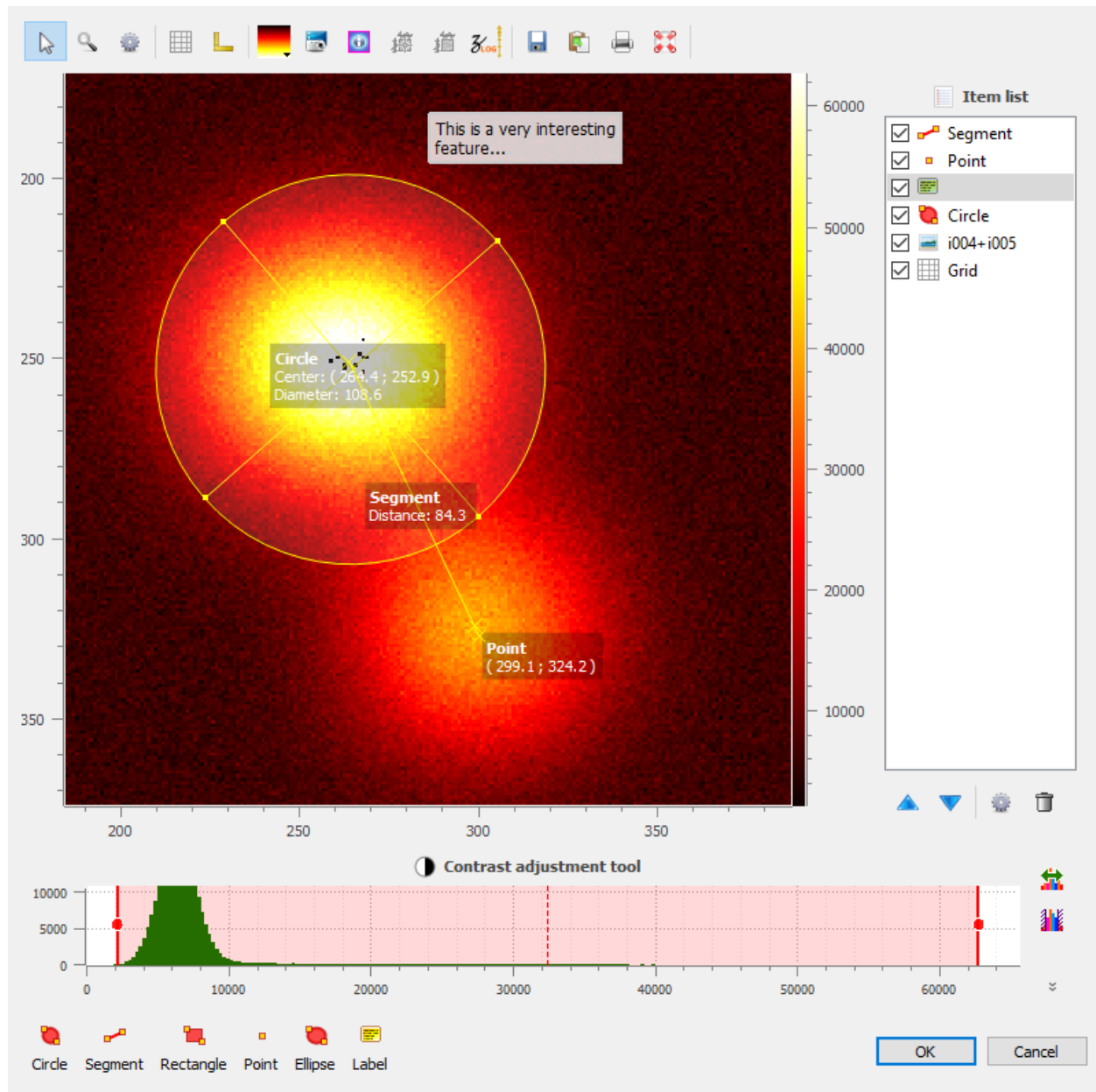
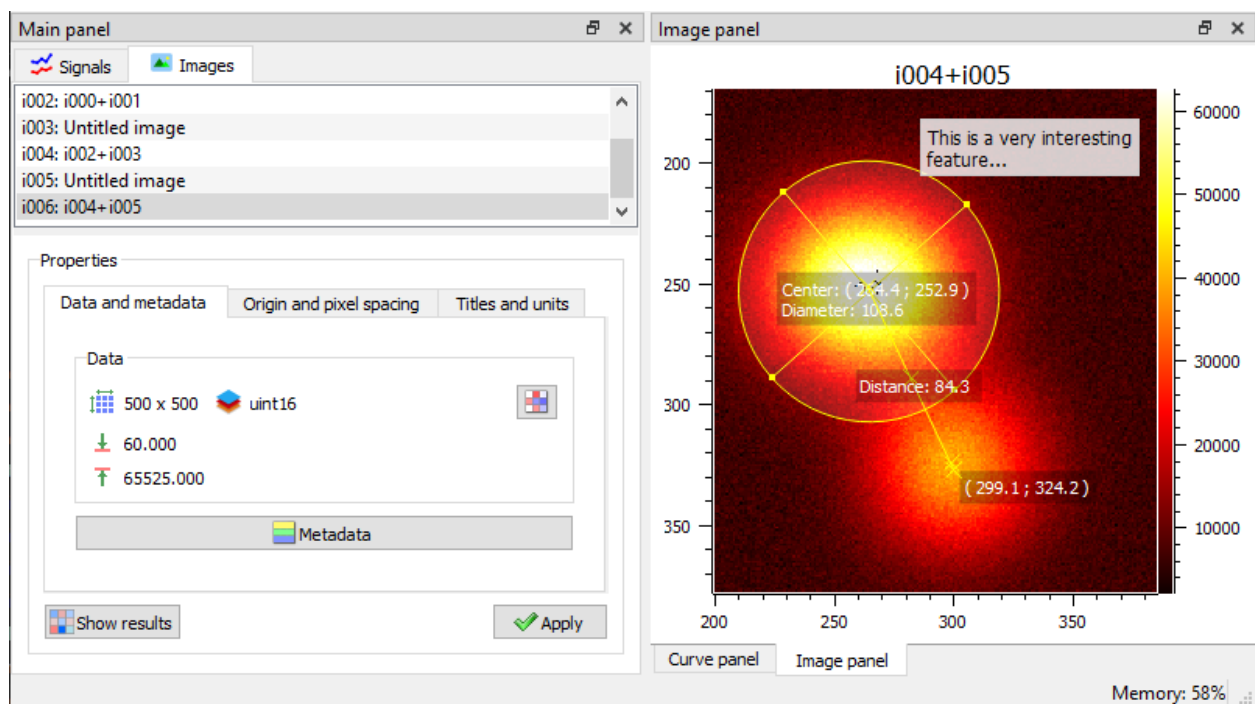


FIG. 13 – Exemple d'annotations.



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 Comment contribuer

5.2.1 Forker le projet

La première étape est de forker le projet sur GitHub. Vous pouvez le faire en visitant le projet DataLab et en cliquant sur le bouton « Fork » en haut à droite de la page.

Une fois que vous avez forké le projet, vous aurez une copie du projet dans votre propre compte GitHub. Ensuite, vous pouvez cloner le projet sur votre ordinateur et commencer à travailler dessus.

5.2.2 Soumettre une pull request

Dès que vous avez apporté des modifications, vous pouvez soumettre une pull request au projet original. Pour ce faire, allez sur votre projet forké sur GitHub et cliquez sur le bouton « Pull request » en haut à droite de la page.

Ensuite vous devrez remplir un formulaire pour décrire votre pull request. Une fois que vous avez soumis la pull request, les mainteneurs du projet examineront vos modifications et les fusionneront s'ils sont satisfaits.

Lors du processus de revue, les mainteneurs du projet vérifieront que votre code suit les règles de codage et qu'il ne casse pas les tests existants. Si votre code ne suit pas les directives de codage, vous devrez le corriger avant que votre pull request puisse être fusionnée.

Voir aussi :

Les règles de codage sont présentées dans la page [Règles de codage](#).

5.3 Règles de codage

5.3.1 Règles de codage génériques

Nous suivons le style de codage [PEP 8](#).

En particulier, nous sommes particulièrement stricts sur les directives suivantes :

- Limiter toutes les lignes à un maximum de 79 caractères.
- Respecter les conventions de nommage (classes, fonctions, variables, etc.).
- Utiliser des exceptions spécifiques au lieu de [Exception](#).

Pour faire respecter ces directives, les outils suivants sont obligatoires :

- [black](#) pour le formatage du code.
- [isort](#) pour le tri des importations.
- [pylint](#) pour l'analyse statique du code.

black

Si vous utilisez [Visual Studio Code](#), les paramètres du projet formateront automatiquement votre code lors de l'enregistrement.

Ou vous pouvez utiliser *black* manuellement. Pour formater votre code, exécutez la commande suivante :

```
black .
```

isort

Encore une fois, si vous utilisez [Visual Studio Code](#), les paramètres du projet trieront automatiquement vos importations lors de l'enregistrement.

Ou vous pouvez utiliser *isort* manuellement. Pour trier vos importations, exécutez la commande suivante :

```
isort .
```

pylint

Pour exécuter *pylint*, exécutez la commande suivante :

```
pylint datalab
```

Si vous utilisez [Visual Studio Code](#) sur Windows, vous pouvez exécuter la tâche « Run Pylint » pour exécuter *pylint* sur le projet.

Note : Une note *pylint* supérieure à 9/10 est requise pour fusionner une demande d'extraction.

5.3.2 Règles de codage spécifiques

En plus des directives de codage génériques, nous avons les directives spécifiques suivantes :

- Écrire des docstrings pour toutes les classes, méthodes et fonctions. Les docstrings doivent suivre le [style Google](#).
- Ajouter des annotations de typage pour toutes les fonctions et méthodes. Les annotations doivent utiliser la syntaxe future (`from __future__ import annotations`)
- Essayez de garder le code aussi simple que possible. Si vous devez écrire un morceau de code complexe, essayez de le diviser en plusieurs fonctions ou classes.
- Ajouter autant de commentaires que possible. Le code doit être auto-explicatif, mais il est toujours utile d'ajouter des commentaires pour expliquer l'idée générale du code, ou pour expliquer certaines parties délicates.
- N'utilisez pas d'instructions `from module import *`, même dans le module `__init__` d'un package.
- Évitez d'utiliser des mixins (héritage multiple) si possible. Il est souvent possible d'utiliser la composition au lieu de l'héritage.
- Évitez d'utiliser les méthodes `__getattr__` et `__setattr__`. Ils sont souvent utilisés pour implémenter une initialisation paresseuse, mais cela peut être fait de manière plus explicite.

5.4 Mise en place de l'environnement de développement

Démarrer le développement dans le cadre du projet DataLab est facile.

Voici ce dont vous aurez besoin :

1. Un environnement de développement intégré (IDE) pour Python. Nous recommandons [Spyder](#) ou [Visual Studio Code](#), mais tout IDE fera l'affaire.
2. Une distribution Python. Nous recommandons [WinPython](#), sur Windows, ou [Anaconda](#), sur Linux ou Mac. Mais, encore une fois, n'importe quelle distribution Python fera l'affaire.
3. Une structure de projet propre (voir ci-dessous).
4. Des données de test (voir ci-dessous).
5. Des variables d'environnement (voir ci-dessous).
6. Des logiciels tiers (voir ci-dessous).

5.4.1 Environnement de développement

Si vous utilisez [Spyder](#), merci de soutenir la communauté scientifique Python open-source !

Si vous utilisez Visual Studio Code, c'est aussi un excellent choix (pour d'autres raisons). Nous recommandons d'installer les extensions suivantes :

Extension	Description
Black Formatter	Formateur de code Python
gettext	Coloration syntaxique Gettext
isort	Classeur d'importations Python
Pylance	Serveur de langage Python
Python	Extension Python
reStructuredText Syntax highlighting	Coloration syntaxique reStructuredText
Ruff	Linter et formateur de code Python extrêmement rapide
Todo Tree	Arbre de tâches

5.4.2 Environnement Python

DataLab nécessite les éléments suivants :

- Python (p.ex. WinPython)
- Paquets Python supplémentaires

Installation de tous les paquets requis :

```
pip install --upgrade -r dev\requirements.txt
```

Voir [Installation](#) pour plus de détails sur les versions de référence de Python et Qt.

Si vous utilisez [WinPython](#), merci de soutenir la communauté scientifique Python open-source !

Le tableau suivant liste les distributions Python actuellement utilisées officiellement :

Version de Python	Statut	Version de WinPython
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

Nous recommandons fortement d'utiliser les versions `.dot` de WinPython qui sont légères et peuvent être personnalisées selon vos besoins (en utilisant `pip install -r requirements.txt`).

Nous recommandons également d'utiliser une instance WinPython dédiée pour DataLab.

5.4.3 Données de test

Les données de test de DataLab sont situées dans différents dossiers, selon leur nature ou leur origine.

Les données requises pour les tests unitaires sont situées dans « `cdl\data\tests` » (données publiques)

Un second dossier `%CDL_DATA%` (facultatif) peut être défini pour des tests supplémentaires qui sont encore en cours de développement (ou pour des données confidentielles).

5.4.4 Variables d'environnement spécifiques

Activer le mode « debug » (pas de redirection stdin/stdout vers la console interne) :

```
@REM Mode DEBUG
set DEBUG=1
```

Générer la documentation PDF nécessite LaTeX. Sur Windows, l'environnement suivant :

```
@REM LaTeX executable must be in Windows PATH, for mathematical equations rendering
@REM Example with MiKTeX :
set PATH=C:\\Apps\\miktex-portable\\texmf\\install\\miktex\\bin\\x64;%PATH%
```

La configuration de Visual Studio Code utilisée dans `launch.json` et `tasks.json` (exemples) :

```
@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
```

Fichier `.env` de Visual Studio Code :

- Ce fichier est utilisé pour définir les variables d’environnement de l’application.
- Il est utilisé pour définir la variable d’environnement `PYTHONPATH` à la racine du projet.
- Cela est nécessaire pour pouvoir importer les modules du projet depuis VS Code.
- Pour créer ce fichier, copiez le fichier `.env.template` en `.env` (et ajoutez éventuellement vos propres chemins).

5.4.5 Logiciels tiers

Les logiciels suivants peuvent être nécessaires pour maintenir le projet :

Logiciel	Description
gettext	Traductions
Git	Système de contrôle de version
ImageMagick	Utilitaires de manipulation d’images
Inkscape	Editeur de graphiques vectoriels
MikTeX	Distribution LaTeX sur Windows

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

6.1 DataLab Version 0.10.0

New features :

- Features common to signals and images :
 - Added « Real part » and « Imaginary part » features to « Operation » menu
 - Added « Convert data type » feature to « Operation » menu
- Features added following user requests (12/18/2023 meetup @ CEA) :
 - Curve and image styles are now saved in the HDF5 file :
 - Curve style covers the following properties : color, line style, line width, marker style, marker size, marker edge color, marker face color, etc.
 - Image style covers the following properties : colormap, interpolation, etc.
 - Those properties were already persistent during the working session, but were lost when saving and reloading the HDF5 file
 - Now, those properties are saved in the HDF5 file and are restored when reloading the HDF5 file
 - New profile extraction features for images :
 - Added « Extract profile » to « Operations » menu, to extract a profile from an image along a row or a column
 - Added « Extract average profile » to « Operations » menu, to extract the average profile on a rectangular area of an image, along a row or a column
 - Image LUT range (contrast/brightness settings) is now saved in the HDF5 file :
 - As for curve and image styles, the LUT range was already persistent during the working session, but was lost when saving and reloading the HDF5 file
 - Now, the LUT range is saved in the HDF5 file and is restored when reloading it
 - Added « Auto-refresh » and « Refresh manually » actions in « View » menu (and main toolbar) :
 - When « Auto-refresh » is enabled (default), the plot view is automatically refreshed when a signal/image is modified, added or removed. Even though the refresh is optimized, this may lead to performance issues when working with large datasets.

- When disabled, the plot view is not automatically refreshed. The user must manually refresh the plot view by clicking on the « Refresh manually » button in the main toolbar or by pressing the standard refresh key (e.g. « F5 »).
- Added `toggle_auto_refresh` method to DataLab proxy object :
 - This method allows to toggle the « Auto-refresh » feature from a macro-command, a plugin or a remote control client.
 - A context manager `context_no_refresh` is also available to temporarily disable the « Auto-refresh » feature from a macro-command, a plugin or a remote control client. Typical usage :

```
with proxy.context_no_refresh():
    # Do something without refreshing the plot view
    proxy.compute_fft() # (...)
```

- Improved curve readability :
 - Until this release, the curve style was automatically set by cycling through **PlotPy** predefined styles
 - However, some styles are not suitable for curve readability (e.g. « cyan » and « yellow » colors are not readable on a white background, especially when combined with a « dashed » line style)
 - This release introduces a new curve style management with colors which are distinguishable and accessible, even to color vision deficiency people
- Added « Curve anti-aliasing » feature to « View » menu (and toolbar) :
 - This feature allows to enable/disable curve anti-aliasing (default : enabled)
 - When enabled, the curve rendering is smoother but may lead to performance issues when working with large datasets (that's why it can be disabled)
- Added `toggle_show_titles` method to DataLab proxy object. This method allows to toggle the « Show graphical object titles » feature from a macro-command, a plugin or a remote control client.
- Remote client is now checking the server version and shows a warning message if the server version may not be fully compatible with the client version.

Bug fixes :

- Image contour detection feature (« Computing » menu) :
 - The contour detection feature was not taking into account the « shape » parameter (circle, ellipse, polygon) when computing the contours. The parameter was stored but really used only when calling the feature a second time.
 - This unintentional behavior led to an `AssertionError` when choosing « polygon » as the contour shape and trying to compute the contours for the first time.
 - This is now fixed (see [Issue #9](#) - Image contour detection : `AssertionError` when choosing « polygon » as the contour shape)
- Keyboard shortcuts :
 - The keyboard shortcuts for « New », « Open », « Save », « Duplicate », « Remove », « Delete all » and « Refresh manually » actions were not working properly.
 - Those shortcuts were specific to each signal/image panel, and were working only when the panel on which the shortcut was pressed for the first time was active (when activated from another panel, the shortcut was not working and a warning message was displayed in the console, e.g. `QAction::event: Ambiguous shortcut overload: Ctrl+C`)
 - Besides, the shortcuts were not working at startup (when no panel had focus).
 - This is now fixed : the shortcuts are now working whatever the active panel is, and even at startup (see [Issue #10](#) - Keyboard shortcuts not working properly : `QAction::event: Ambiguous shortcut overload: Ctrl+C`)
- « Show graphical object titles » and « Auto-refresh » actions were not working properly :
 - The « Show graphical object titles » and « Auto-refresh » actions were only working on the active signal/image panel, and not on all panels.
 - This is now fixed (see [Issue #11](#) - « Show graphical object titles » and « Auto-refresh » actions were working only on current signal/image panel)
- Fixed [Issue #14](#) - Saving/Reopening HDF5 project without cleaning-up leads to `ValueError`
- Fixed [Issue #15](#) - MacOS : 1. `pip install cdl` error - 2. Missing menus :
 - Part 1 : `pip install cdl` error on MacOS was actually a **PlotPy** issue (<https://github.com/plotpy/plotpy/issues/15>)

thub.com/PlotPyStack/PlotPy/issues/9), and was fixed in PlotPy v2.0.3 with an additional compilation flag indicating to use C++11 standard

- Part 2 : Missing menus on MacOS was due to a PyQt/MacOS bug regarding dynamic menus
- HDF5 file format : when importing an HDF5 dataset as a signal or an image, the dataset attributes were systematically copied to signal/image metadata : we now only copy the attributes which match standard data types (integers, floats, strings) to avoid errors when serializing/deserializing the signal/image object
- Installation/configuration viewer : improved readability (removed syntax highlighting)
- PyInstaller specification file : added missing skimage data files manually in order to continue supporting Python 3.8 (see [Issue #12](#) - Stand-alone version on Windows 7 : missing api-ms-win-core-path-l1-1-0.dll)

6.2 DataLab Version 0.9.2

Bug fixes :

- Region of interest (ROI) extraction feature for images :
 - ROI extraction was not working properly when the « Extract all regions of interest into a single image object » option was enabled if there was only one defined ROI. The result was an image positioned at the origin (0, 0) instead of the expected position (x0, y0) and the ROI rectangle itself was not removed as expected. This is now fixed (see [Issue #6](#) - “Extract multiple ROI” feature : unexpected result for a single ROI)
 - ROI rectangles with negative coordinates were not properly handled : ROI extraction was raising a `ValueError` exception, and the image mask was not displayed properly. This is now fixed (see [Issue #7](#) - Image ROI extraction : `ValueError: zero-size array to reduction operation minimum which has no identity`)
 - ROI extraction was not taking into account the pixel size (dx, dy) and the origin (x0, y0) of the image. This is now fixed (see [Issue #8](#) - Image ROI extraction : take into account pixel size)
- Macro-command console is now read-only :
 - The macro-command panel Python console is currently not supporting standard input stream (`stdin`) and this is intended (at least for now)
 - Set Python console read-only to avoid confusion

6.3 DataLab Version 0.9.1

Bug fixes :

- French translation is not available on Windows/Stand alone version :
 - Locale was not properly detected on Windows for stand-alone version (frozen with `pyinstaller`) due to an issue with `locale.getlocale()` (function returning `None` instead of the expected locale on frozen applications)
 - This is ultimately a `pyinstaller` issue, but a workaround has been implemented in `guidata V3.2.2` (see [guidata issue #68](#) - Windows : gettext translation is not working on frozen applications)
 - [Issue #2](#) - French translation is not available on Windows Stand alone version
- Saving image to JPEG2000 fails for non integer data :
 - JPEG2000 encoder does not support non integer data or signed integer data
 - Before, DataLab was showing an error message when trying to save incompatible data to JPEG2000 : this was not a consistent behavior with other standard image formats (e.g. PNG, JPG, etc.) for which DataLab was automatically converting data to the appropriate format (8-bit unsigned integer)
 - Current behavior is now consistent with other standard image formats : when saving to JPEG2000, DataLab automatically converts data to 8-bit unsigned integer or 16-bit unsigned integer (depending on the original data type)
 - [Issue #3](#) - Save image to JPEG2000 : “`OSError : encoder error -2 when writing image file`”
- Windows stand-alone version shortcuts not showing in current user start menu :

- When installing DataLab on Windows from a non-administrator account, the shortcuts were not showing in the current user start menu but in the administrator start menu instead (due to the elevated privileges of the installer and the fact that the installer does not support installing shortcuts for all users)
- Now, the installer *does not* ask for elevated privileges anymore, and shortcuts are installed in the current user start menu (this also means that the current user must have write access to the installation directory)
- In future releases, the installer will support installing shortcuts for all users if there is a demand for it (see [Issue #5](#))
- [Issue #4](#) - Windows : stand-alone version shortcuts not showing in current user start menu
- Installation and configuration window for stand-alone version :
 - Do not show ambiguous error message “Invalid dependencies” anymore
 - Dependencies are supposed to be checked when building the stand-alone version
- Added PDF documentation to stand-alone version :
 - The PDF documentation was missing in previous release
 - Now, the PDF documentation (in English and French) is included in the stand-alone version

6.4 DataLab Version 0.9.0

New dependencies :

- DataLab is now powered by [PlotPyStack](#) :
 - [PythonQwt](#)
 - [guidata](#)
 - [PlotPy](#)
- [opencv-python](#) (algorithms for image processing)

New reference platform :

- DataLab is validated on Windows 11 with Python 3.11 and PyQt 5.15
- DataLab is also compatible with other OS (Linux, MacOS) and other Python-Qt bindings and versions (Python 3.8-3.12, PyQt6, PySide6)

New features :

- DataLab is a platform :
 - Added support for plugins
 - Custom processing features available in the « Plugins » menu
 - Custom I/O features : new file formats can be added to the standard I/O features for signals and images
 - Custom HDF5 features : new HDF5 file formats can be added to the standard HDF5 import feature
 - More features to come...
 - Added remote control feature : DataLab can be controlled remotely via a TCP/IP connection (see [Remote control](#))
 - Added macro commands : DataLab can be controlled via a macro file (see [Macro commands](#))
- General features :
 - Added settings dialog box (see « Settings » entry in « File » menu) :
 - General settings
 - Visualization settings
 - Processing settings
 - Etc.
 - New default layout : signal/image panels are on the right side of the main window, visualization panels are on the left side with a vertical toolbar
- Signal/Image features :
 - Added process isolation : each signal/image is processed in a separate process, so that DataLab does not freeze anymore when processing large signals/images
 - Added support for groups : signals and images can be grouped together, and operations can be applied to all objects in a group, or between groups
 - Added warning and error dialogs with detailed traceback links to the source code (warnings may be optionally ignored)

- Drastically improved performance when selecting objects
- Optimized performance when showing large images
- Added support for dropping files on signal/image panel
- Added « Computing parameters » group box to show last result input parameters
- Added « Copy titles to clipboard » feature in « Edit » menu
- For every single processing feature (operation, processing and computing menus), the entered parameters (dialog boxes) are stored in cache to be used as defaults the next time the feature is used
- Signal processing :
 - Added support for optional FFT shift (see Settings dialog box)
- Image processing :
 - Added pixel binning operation (X/Y binning factors, operation : sum, mean, ...)
 - Added « Distribute on a grid » and « Reset image positions » in operation menu
 - Added Butterworth filter
 - Added exposure processing features :
 - Gamma correction
 - Logarithmic correction
 - 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 adaptive 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

CHAPITRE 7

Licence et copyrights

- Copyright © 2023 Codra, Pierre Raybaut
- Distribué sous licence BSD 3-Clause

C

- `cdl.core.gui.processor.image`, 30
- `cdl.core.gui.processor.signal`, 28
- `cdl.core.model.image`, 39
- `cdl.core.model.signal`, 34
- `cdl.obj`, 34
- `cdl.param`, 34
- `cdl.plugins`, 53