DataLab

Version 0.14.1

mars 18, 2024

Table des matières

1 Premiers pas **3**
- 1.1 Installation . . . 3
- 1.2 Cas d'usage, principales fonctionnalités et points forts . . . 9
- 1.3 Fonctionnalités principales . . . 11
- 1.4 Tutoriels . . . 13

2 Fonctionnalités **81**
- 2.1 Fonctionnalités générales . . . 81
- 2.2 Traitement du signal . . . 120
- 2.3 Traitement d'image . . . 147

3 API **185**
- 3.1 Algorithmes (`cdl.algorithms`) . . . 185
- 3.2 Paramètres (`cdl.param`) . . . 195
- 3.3 Modèle de données (`cdl.obj`) . . . 200
- 3.4 Calcul (`cdl.core.computation`) . . . 219
- 3.5 Objets proxy (`cdl.proxy`) . . . 245

4 Contribuer **259**
- 4.1 Partagez vos idées et vos expériences . . . 259
- 4.2 Partagez vos connaissances scientifiques/techniques . . . 259
- 4.3 Contribuer aux nouvelles fonctionnalités . . . 260
- 4.4 Développer de nouvelles fonctionnalités . . . 260

Index des modules Python **279**

DataLab est une **plateforme open-source de traitement et de visualisation de données scientifiques et techniques** avec des fonctionnalités uniques conçues pour répondre à des niveaux d'exigence industriels. Exploitant la richesse de l'écosystème scientifique Python [1] et des interfaces graphiques Qt, DataLab est un outil polyvalent, extensible avec des *Plugins* et fonctionnant parfaitement avec *votre IDE* ou *vos notebooks Jupyter*.

Pour voir DataLab en action immédiatement, vous avez deux options :

— Lisez ou regardez nos *Tutoriels*,
— Essayez DataLab en ligne, sans installation, en utilisant notre environnement Binder.

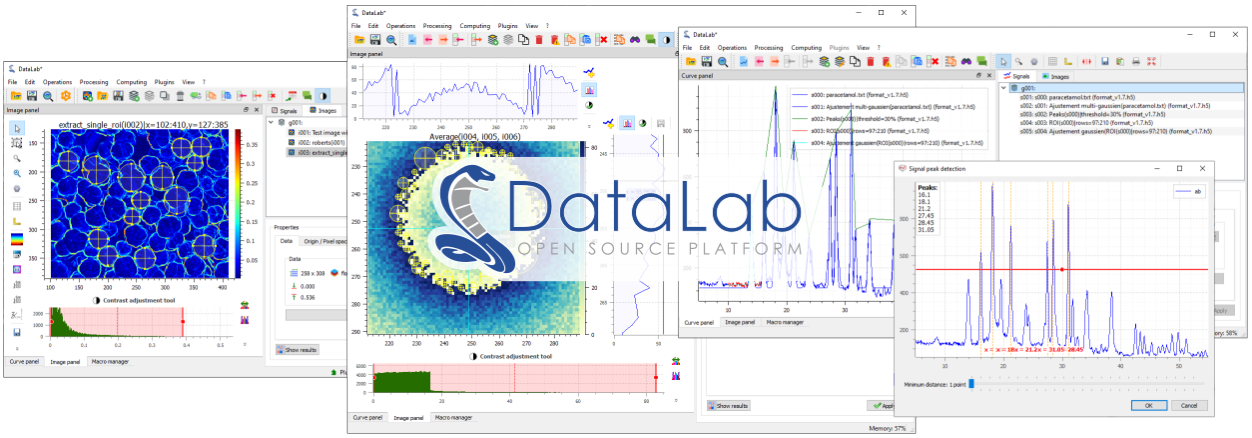


FIG. 1 – Visualisation de signaux et d'images dans DataLab

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.

Powered by PlotPyStack

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

1. Les fonctionnalités de traitement de DataLab sont principalement basées sur les bibliothèques NumPy, SciPy, scikit-image, OpenCV et PyWavelets. Les capacités de visualisation de DataLab sont basées sur la boîte à outils PlotPyStack, un ensemble de bibliothèques Python pour la réalisation d'applications scientifiques avec des interfaces graphiques Qt.

CHAPITRE 1

Premiers pas

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.

DataLab s'appuie sur la puissance de Python et de son écosystème scientifique, grâce à l'utilisation des bibliothèques suivantes :

- NumPy pour le calcul numérique (tableaux, algèbre linéaire, etc.)
- SciPy pour le calcul scientifique (interpolation, fonctions spéciales, etc.)
- scikit-image et *OpenCV <https ://opencv.org/>* pour le traitement d'images
- PyWavelets pour la transformée en ondelettes
- PlotPyStack pour la visualisation interactive de données basée sur Qt

1.1 Installation

Cette section fournit des informations sur l'installation de DataLab sur votre système. Une fois installé, vous pouvez démarrer DataLab en exécutant la commande `cdl` dans un terminal, ou en cliquant sur le raccourci DataLab dans le menu Démarrer (sous Windows).

Voir aussi :

Pour plus de détails sur l'exécution de DataLab et ses options en ligne de commande, voir *Ligne de commande*.

1.1.1 Modes d'installation

DataLab est disponible sous plusieurs formes :

- Un paquet Python qui peut être installé à l'aide du *Gestionnaire de paquets pip*.
- Windows En tant qu'application autonome, qui ne nécessite pas d'installation de Python. Il suffit d'exécuter l'*Installeur tout-en-un* et le tour est joué !
- Windows Dans une distribution prête à l'emploi *Distribution Python*, basée sur WinPython.
- En tant que *Paquet Wheel* précompilé, qui peut être installé à l'aide de `pip`.
- En tant que *Paquet source*, qui peut être installé à l'aide de `pip` ou manuellement.

Voir aussi :

Impatient d'essayer la prochaine version de DataLab ? Vous pouvez également installer la dernière version de développement de DataLab à partir de la branche principale du dépôt Git. Voir *Version de développement* pour plus d'informations.

Gestionnaire de paquets `pip`

GNU/Linux Windows macOS

Le paquet `cdl` de DataLab est disponible sur l'index des paquets Python (PyPI) à l'adresse suivante : https://pypi.python.org/pypi/cdl.

L'installation de DataLab depuis PyPI avec Qt est aussi simple que d'exécuter cette commande (vous devrez peut-être utiliser `pip3` au lieu de `pip` sur certains systèmes) :

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

Ou, si vous préférez, vous pouvez installer DataLab sans la bibliothèque Qt (non recommandé) :

```
$ pip install cdl
```

Note : Si vous avez déjà une version antérieure de DataLab installée, vous pouvez la mettre à jour en exécutant la même commande avec l'option `--upgrade` :

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

Installeur tout-en-un

Windows

DataLab est disponible sous la forme d'une application autonome pour Windows qui ne nécessite pas d'installation de Python. Il suffit d'exécuter l'installeur et vous êtes prêt à partir !

Le paquet d'installation est disponible dans la section Releases. Il prend en charge la désinstallation et la mise à jour automatiques (pas besoin de désinstaller DataLab avant d'exécuter l'installeur d'une autre version de l'application).

Avertissement : L'installeur Windows de DataLab est disponible pour Windows 8, 10 et 11 (version principale, basée sur Python 3.11) et également pour Windows 7 SP1 (version basée sur Python 3.8, voir le fichier se terminant par `-Win7.exe`).

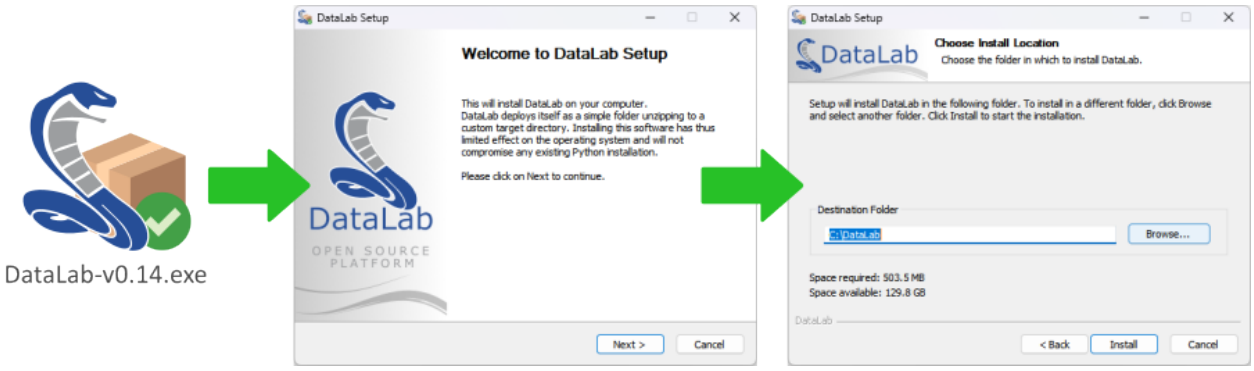


FIG. 1 – Installeur tout-en-un de DataLab pour Windows

Sur Windows 7 SP1, avant d'exécuter DataLab (ou toute autre application Python 3), vous devez installer la mise à jour Microsoft *KB2533623* (*Windows6.1-KB2533623-x64.msu*) et vous devrez peut-être également installer le package redistribuable Microsoft Visual C++ 2015-2022.

Distribution Python

Windows

DataLab est également disponible dans une distribution Python prête à l'emploi, basée sur WinPython. Cette distribution s'appelle DataLab-WinPython et est disponible dans la section DataLab-WinPython Releases.

FIG. 2 – DataLab-WinPython est une distribution Python prête à l'emploi incluant la plateforme DataLab.

La principale différence avec l'installeur tout-en-un est que vous pouvez utiliser la distribution Python à d'autres fins que l'exécution de DataLab, et vous pouvez également l'étendre avec des paquets supplémentaires. En revanche, elle est également *beaucoup plus volumineuse* que l'installeur tout-en-un car elle inclut une distribution Python complète.

Avertissement : Alors que l'installeur tout-en-un fournit un paquet monolithique qui garantit la compatibilité de tous ses composants car il ne peut pas être modifié par l'utilisateur, la distribution WinPython est plus flexible et peut donc être endommagée par une mauvaise manipulation de la distribution Python par l'utilisateur. Cela doit être pris en compte lors du choix de la méthode d'installation.

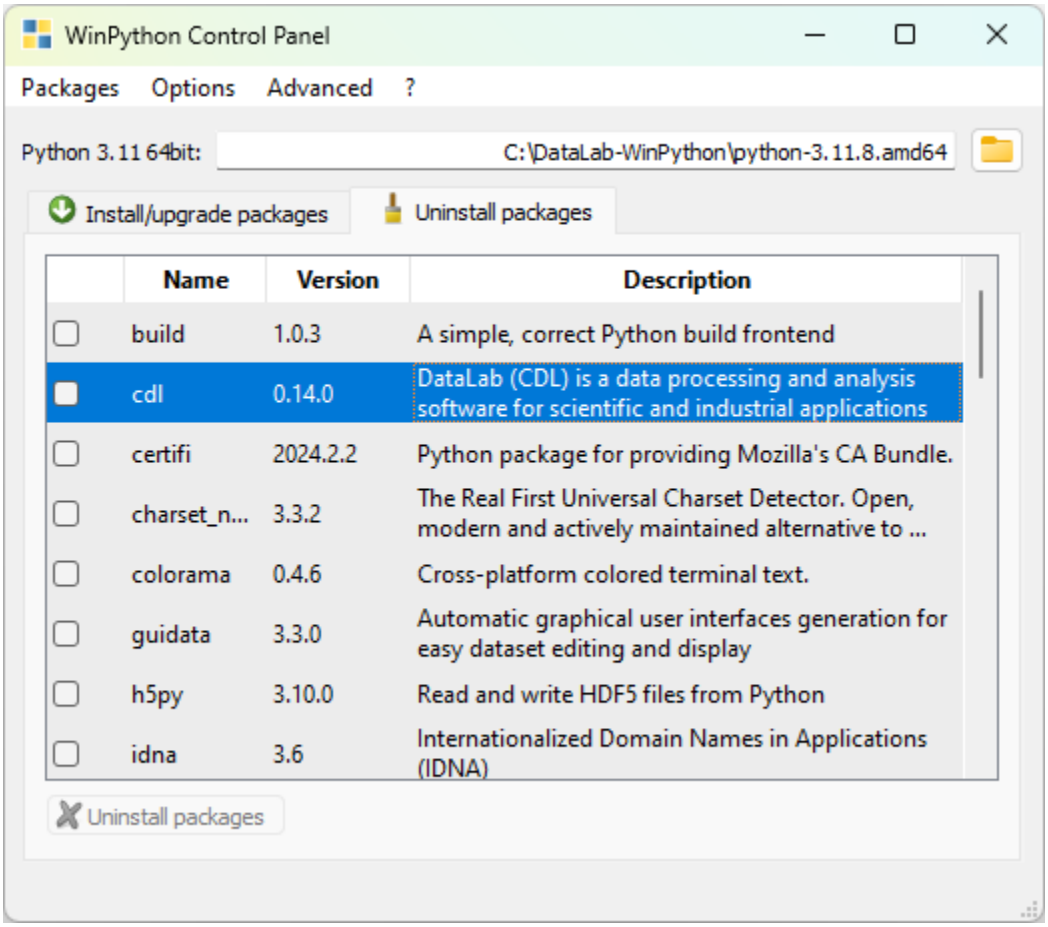


FIG. 3 – Panneau de contrôle DataLab-WinPython

Paquet Wheel

GNU/Linux Windows macOS

Sur n'importe quel système d'exploitation, l'utilisation de pip et du paquet Wheel est le moyen le plus simple d'installer DataLab sur une distribution Python existante :

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

Paquet source

GNU/Linux Windows macOS

L'installation de DataLab directement depuis le paquet source peut être effectuée à l'aide de `pip` :

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

Ou, si vous préférez, vous pouvez l'installer manuellement en exécutant la commande suivante depuis le répertoire racine du paquet source :

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

Enfin, vous pouvez également créer votre propre paquet Wheel et l'installer à l'aide de `pip`, en exécutant la commande suivante depuis le répertoire racine du paquet source (cela nécessite que les paquets `build` et `wheel` soient installés) :

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

Version de développement

GNU/Linux Windows macOS

Si vous souhaitez essayer la dernière version de développement de DataLab, vous pouvez l'installer directement depuis la branche principale du dépôt Git.

La première fois que vous installez DataLab depuis le dépôt Git, entrez la commande suivante :

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

Ensuite, si vous souhaitez à un moment donné passer à la dernière version de DataLab, exécutez simplement la même commande avec les options pour forcer la réinstallation du paquet sans gérer les dépendances (car cela réinstallerait toutes les dépendances) :

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

Note : Si les dépendances ont changé, vous devrez peut-être exécuter la même commande que ci-dessus, mais sans l'option `--no-deps`.

1.1.2 Dépendances

Note : L'installeur tout-en-un de DataLab inclut déjà toutes ces dépendances ainsi que Python lui-même.

Le paquet `cdl` 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-headless	>= 4.5	
PyWavelets	>= 1.1	
psutil	>= 5.5	
guidata	>= 3.4	
PlotPy	>= 2.3	
QtPy	>= 1.9	
PyQt5	>=5.11	Bibliothèque d'interfaces graphiques Qt pour Python

Modules optionnels pour le développement :

Nom	Version	Description
black		Le formateur de code sans compromis.
isort		Un utilitaire Python pour trier les imports Python.
pylint		Analyseur statique de code Python
Coverage		Mesure de la couverture de code pour Python
pyinstaller	>=6.0	PyInstaller permet de créer un exécutable unique à partir d'une application Python et de ses dépendances.

Modules optionnels pour la génération de la documentation :

Nom	Version	Description
PyQt5		Bibliothèque d'interfaces graphiques Qt pour Python
sphinx		Générateur de documentation Python
sphinx_intl		Utilitaire pour la traduction de la documentation générée par Sphinx.
sphinx-sitemap		Sitemap generator for Sphinx
myst_parser		Un parseur étendu compatible avec [CommonMark](https://spec.commonmark.org/)
sphinx_design		Extension sphinx pour la conception de composants web réactifs.
sphinx-copybutton		Extension sphinx ajoutant un bouton de copie à chaque cellule de code.
pydata-sphinx-theme		Thème Bootstrap pour Sphinx de la communauté PyData

Optional modules for running test suite :

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

Note : Python 3.11 et PyQt5 sont les références pour la version de production

1.2 Cas d'usage, principales fonctionnalités et points forts

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.

1.2.1 Quelles sont les applications de DataLab ?

Exemples concrets

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

- Traitement de données expérimentales (signaux et images) acquises sur une installation scientifique dans le domaine nucléaire
- Traitement de données acquises par un capteur dans un contexte industriel
- Traitement d'images acquises par une caméra dans un contexte médical
- Détection automatique de défauts sur une surface, dans le contexte du contrôle qualité
- Détection automatique des positions des tâches laser dans une scène, dans le contexte de l'alignement laser
- Alignement d'instrument par traitement d'image
- Détection automatique de motifs et corrections géométriques d'images, dans le contexte du contrôle non destructif

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.

Cas d'usage

Voir aussi :

Pour des exemples pratiques de cas d'usage, voir la section *Tutoriels* :

- La plupart des tutoriels décrivent des exemples concrets d'utilisation de DataLab dans un contexte scientifique ou technique.
- Concernant l'utilisation de DataLab avec un IDE (Integrated Development Environment) tel que Visual Studio Code ou Spyder, voir le tutoriel *DataLab et Spyder : un mariage parfait*.
- Quant à l'utilisation de DataLab avec des notebooks Jupyter, c'est l'un des sujets abordés dans le tutoriel *Ajouter vos propres fonctionnalités*.

DataLab est un outil polyvalent qui peut être utilisé dans différents contextes :

Traitement de données

DataLab est un outil puissant pour le traitement de signaux et d'images. Il peut être utilisé pour développer des algorithmes complexes, ou pour prototyper rapidement une chaîne de traitement.

Voir nos *Tutoriels* pour des exemples pratiques d'utilisation dans le traitement de données.

Outil de soutien aux travaux scientifiques/techniques

DataLab peut être utilisé comme un outil de soutien aux travaux scientifiques/techniques. Il vous permet de visualiser et de traiter des données, et de partager vos résultats avec vos collègues. Il peut facilement être adapté à vos besoins par l'ajout de plugins, et peut même être utilisé avec vos outils quotidiens (par exemple Visual Studio Code, Spyder, … ou des notebooks Jupyter).

Voir nos *Tutoriels* pour des exemples pratiques d'utilisation dans un contexte scientifique/technique.

Prototypage d'une application de traitement de données

DataLab peut être utilisé pour prototyper rapidement une application de traitement de données. Il peut ensuite être utilisé comme base pour le développement d'une application à part entière.

Voir le tutoriel *Prototypage d'une chaîne de traitement personnalisée* pour un exemple concret.

Débogage d'une application de traitement de données

DataLab peut être utilisé comme un outil de débogage avancé pour vos applications de traitement de données, indépendamment de l'environnement de développement ou du langage utilisé (Python, C#, C++, …). Tout ce dont vous avez besoin est de pouvoir communiquer avec DataLab via son interface de pilotage à distance (protocole XML-RPC standard). Cela vous permet d'envoyer des données à DataLab (signaux, images ou même des formes géométriques), de visualiser les données à chaque étape de la chaîne de traitement, de les manipuler pour mieux comprendre le comportement de vos algorithmes, et même de les modifier pour tester la robustesse de votre code.

Voir le tutoriel *Déboguer votre algorithme avec DataLab* pour un aperçu rapide de cette fonctionnalité.

Note : DataLab peut également être piloté depuis votre environnement de développement familier (par exemple Visual Studio Code, Spyder, …) ou depuis un notebook Jupyter, afin d'effectuer des calculs en utilisant vos fonctions de traitement tout en bénéficiant des fonctionnalités avancées de DataLab. Voir les tutoriels *Prototypage d'une chaîne de traitement personnalisée* ou *DataLab et Spyder : un mariage parfait* pour des exemples d'utilisation.

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.

1.2.2 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

1.2.3 Points forts

Pour résumer, les quatre points forts de DataLab sont les suivants :

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.

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.

Automatisation
: Une API publique de haut niveau permet de piloter à distance DataLab pour ouvrir et traiter des données.

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.

1.3 Fonctionnalités principales

Cette page présente brièvement les principales fonctionnalités de DataLab.

1.3.1 Visualisation de données

Signal	Image	Fonctionnalité
✓	✓	Captures d'écran (enregistrer, copier)
✓	Axe Z	Échelles linéaire/logarithmique
✓	✓	Édition de table de données
✓	✓	Statistiques sur ROI défini par l'utilisateur
✓	✓	Marqueurs
	✓	Ratio d'aspect (1 :1, personnalisé)
	✓	Plus de 50 échelles de couleurs disponibles
	✓	Profils X/Y bruts/moyennés
✓	✓	Annotations
✓	✓	Persistance des paramètres dans l'espace de travail
	✓	Distribuer les images sur une grille

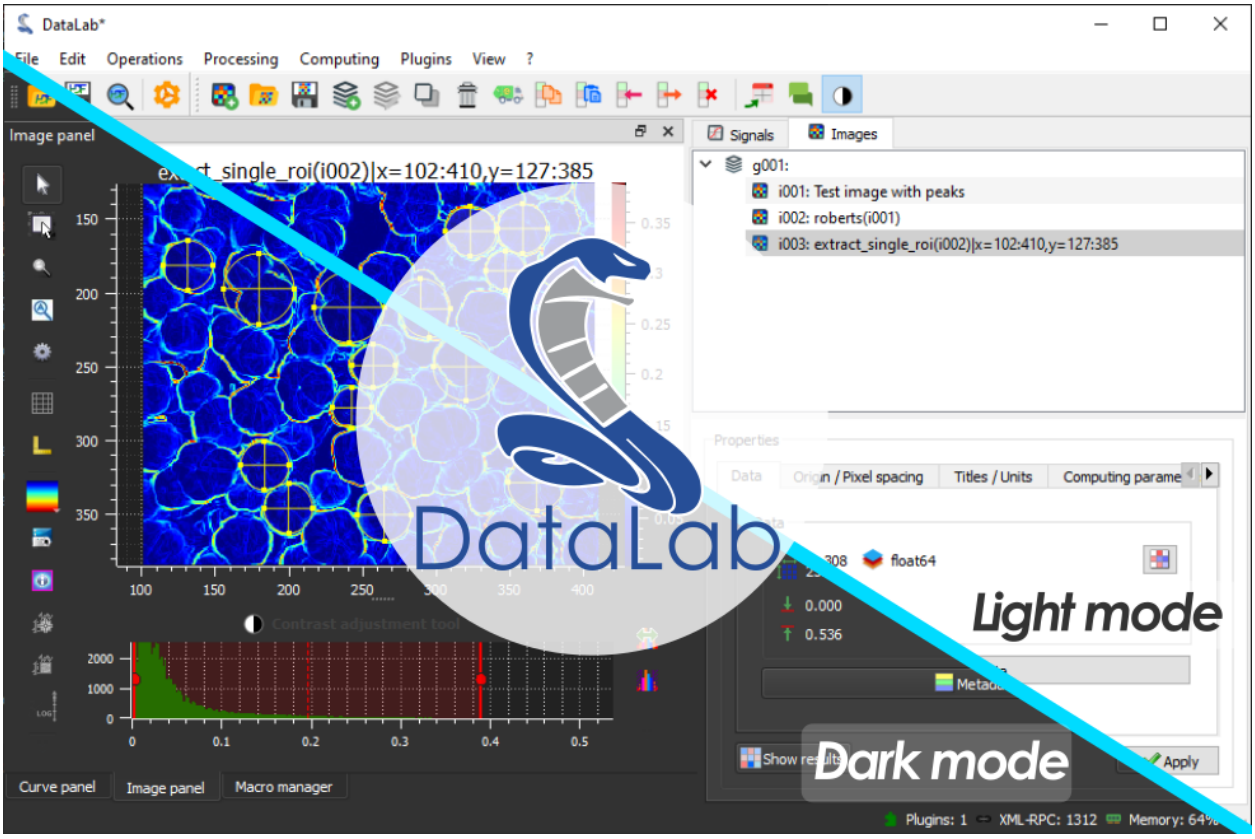


FIG. 4 – DataLab supports dark and light mode depending on your platform settings (this is handled by the guidata package, and may be overridden by setting the *QT_COLOR_MODE* environment variable to *dark* or *light*).

1.3.2 Traitement de données

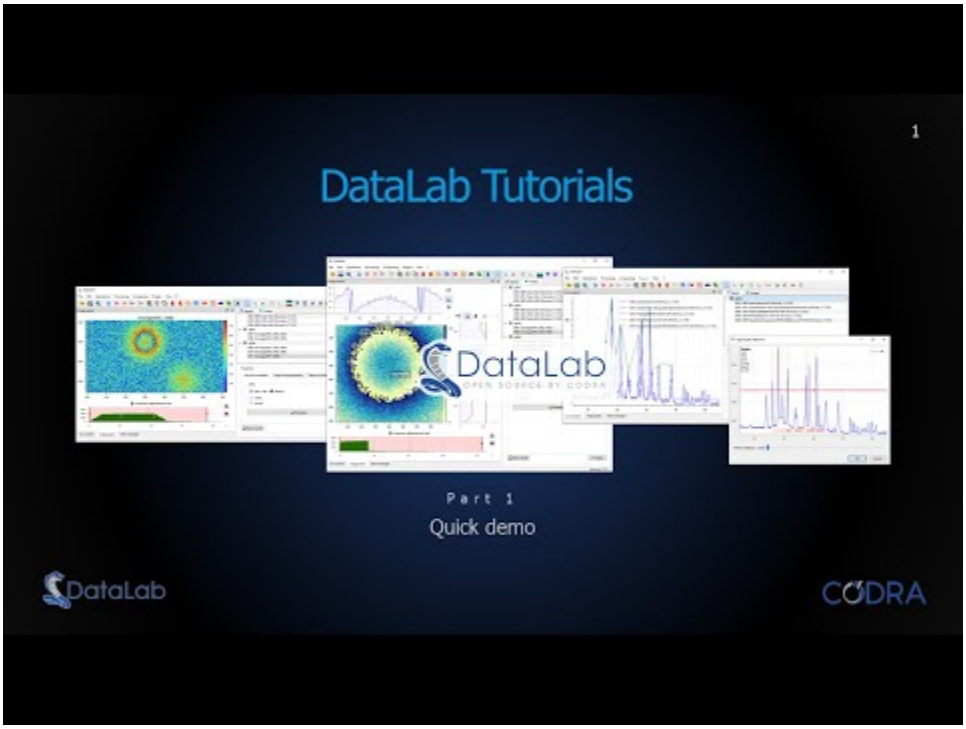
Signal	Image	Fonctionnalité
✓	✓	Processus isolé pour l'exécution des calculs
✓	✓	Contrôle à distance depuis Jupyter, Spyder ou tout autre IDE
✓	✓	Contrôle à distance depuis une application tierce
✓	✓	Somme, moyenne, différence, produit, …
✓	✓	Extraction de ROI, permutation des axes X/Y
✓		Détection de pics semi-automatique
✓		Convolution
	✓	Correction de champ plat
	✓	Rotation (retournement, rotation), redimensionnement, …
	✓	Profils d'intensité (ligne, moyen, radial)
	✓	Binning de pixels
✓		Normalisation, dérivée, intégrale
✓	✓	Calibration linéaire
	✓	Seuillage, rognage
✓	✓	Filtre gaussien, filtre de Wiener
✓	✓	Moyenne mobile, médiane mobile
✓	✓	FFT, FFT inverse
✓		Interpolation, rééchantillonnage
✓		Elimination de tendance
✓		Ajustement interactif : Gauss, Lorenzt, Voigt, polynôme
✓		Ajustement interactif multi-gaussien
	✓	Filtre de Butterworth
	✓	Correction d'exposition (gamma, log, …)
	✓	Restauration (variation totale, bilatérale, …)
	✓	Morphologie (érosion, dilatation, …)
	✓	Détection de contours (Roberts, Sobel, …)
✓	✓	Calcul sur ROI personnalisé
✓		FWHM, FW @ $1/e^2$
	✓	Barycentre (méthode robuste par rapport au bruit)
	✓	Centre du cercle d'encadrement minimal
	✓	Détection de pics 2D
	✓	Détection de contours
	✓	Circle Hough transform
	✓	Détection de taches (OpenCV, Laplacien de Gauss, …)

1.4 Tutoriels

1.4.1 Tutoriels vidéo

Les tutoriels vidéo de DataLab ont pour but de fournir un matériel complémentaire à la documentation et aux autres tutoriels disponibles ici.

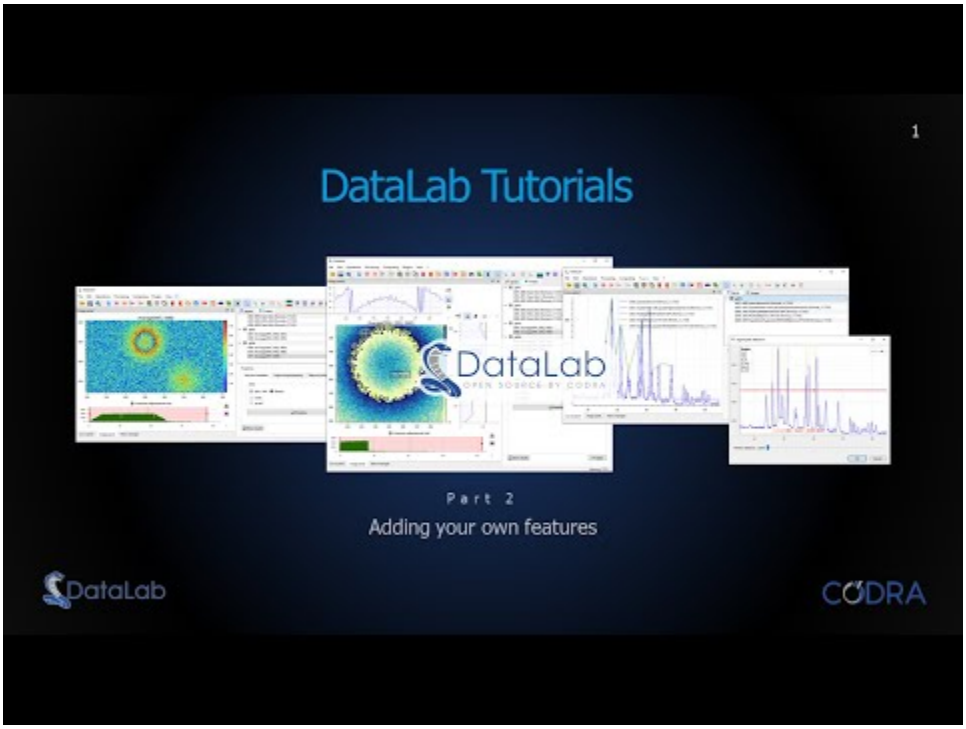
Démo rapide



Dans cette première vidéo, nous allons rapidement passer en revue les principales fonctionnalités de DataLab, et montrer comment effectuer un traitement de signal et d'image de base.

Avertissement : Cette vidéo est volontairement courte et ne couvre pas toutes les fonctionnalités de DataLab. Elle a pour but de vous donner un aperçu rapide du logiciel. Pour une description plus détaillée des fonctionnalités, veuillez regarder les autres vidéos ou lire la documentation.

Ajouter vos propres fonctionnalités



Dans ce tutoriel, nous allons montrer comment ajouter vos propres fonctionnalités à DataLab en utilisant trois approches différentes :

1. Les macro-commandes, en utilisant le gestionnaire de macro intégré
2. Le contrôle à distance de DataLab depuis un IDE externe (par exemple Spyder) ou un notebook Jupyter
3. Les plugins

Le premier point commun entre ces trois approches est qu'elles reposent toutes sur l'API de haut niveau de DataLab, qui permet d'interagir avec presque tous les aspects du logiciel. Cette API est ici accédée à l'aide de scripts Python, mais elle peut également être accédée à l'aide de n'importe quel autre langage lors de l'utilisation de l'approche de contrôle à distance (car elle repose sur un protocole de communication standard, XML-RPC).

Le second point commun est qu'ils utilisent tous du code Python et des objets proxy compatibles, de sorte que le même code peut être au moins partiellement réutilisé dans les trois approches.

1.4.2 Autres tutoriels

Les tutoriels suivants sont des guides détaillés étape par étape pour effectuer des tâches spécifiques avec DataLab, ou pour illustrer les fonctionnalités du logiciel dans le contexte d'un problème scientifique ou technique. Chaque tutoriel se concentre sur un aspect spécifique du logiciel et est destiné à être autonome.

Traitement d'un spectre

Cet exemple montre comment traiter un spectre avec DataLab :

- Lire le spectre à partir d'un fichier
- Appliquer un filtre au spectre
- Extraire une région d'intérêt
- Ajuster un modèle au spectre
- Sauvegarder l'espace de travail dans un fichier

Tout d'abord, nous ouvrons DataLab et lisons le spectre à partir d'un fichier.

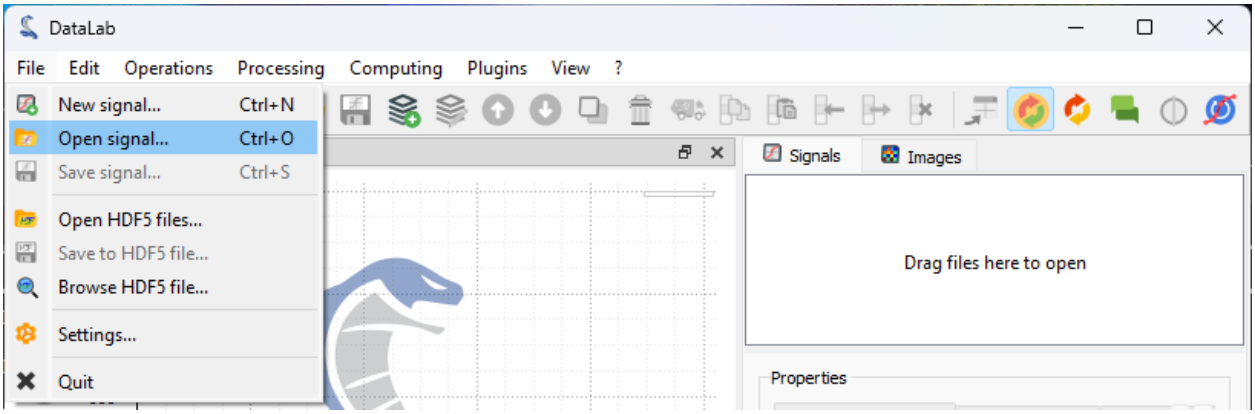


FIG. 5 – Ouvrir le fichier de spectre avec « Fichier > Ouvrir… », ou avec le bouton dans la barre d'outils, ou en faisant glisser et déposer le fichier dans DataLab (sur le panneau de droite).

Ici, nous générons en fait le signal à partir d'un fichier de données de test (en utilisant « Plugins > Test data > Load spectrum of paracetamol »), mais le principe est le même.

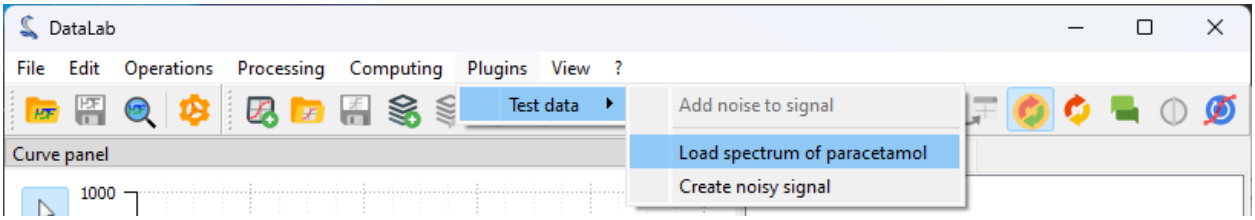


FIG. 6 – Utiliser le plugin « Données de test » est un moyen pratique de générer des données de test pour les tutoriels, mais vous pouvez utiliser n'importe quel fichier contenant un spectre, comme un spectre provenant d'une véritable expérience.

Le spectre est affiché dans la fenêtre principale.

A présent, traitons ce spectre en lui appliquant un filtre. Nous utiliserons un filtre de Wiener, qui est un filtre qui peut être utilisé pour supprimer le bruit d'un signal, même si cela paraît superflu dans le cas présent.

Si nous voulons analyser une région spécifique du spectre, nous pouvons l'extraire du spectre en utilisant la fonction « Extraction de ROI » du menu « Opérations ».

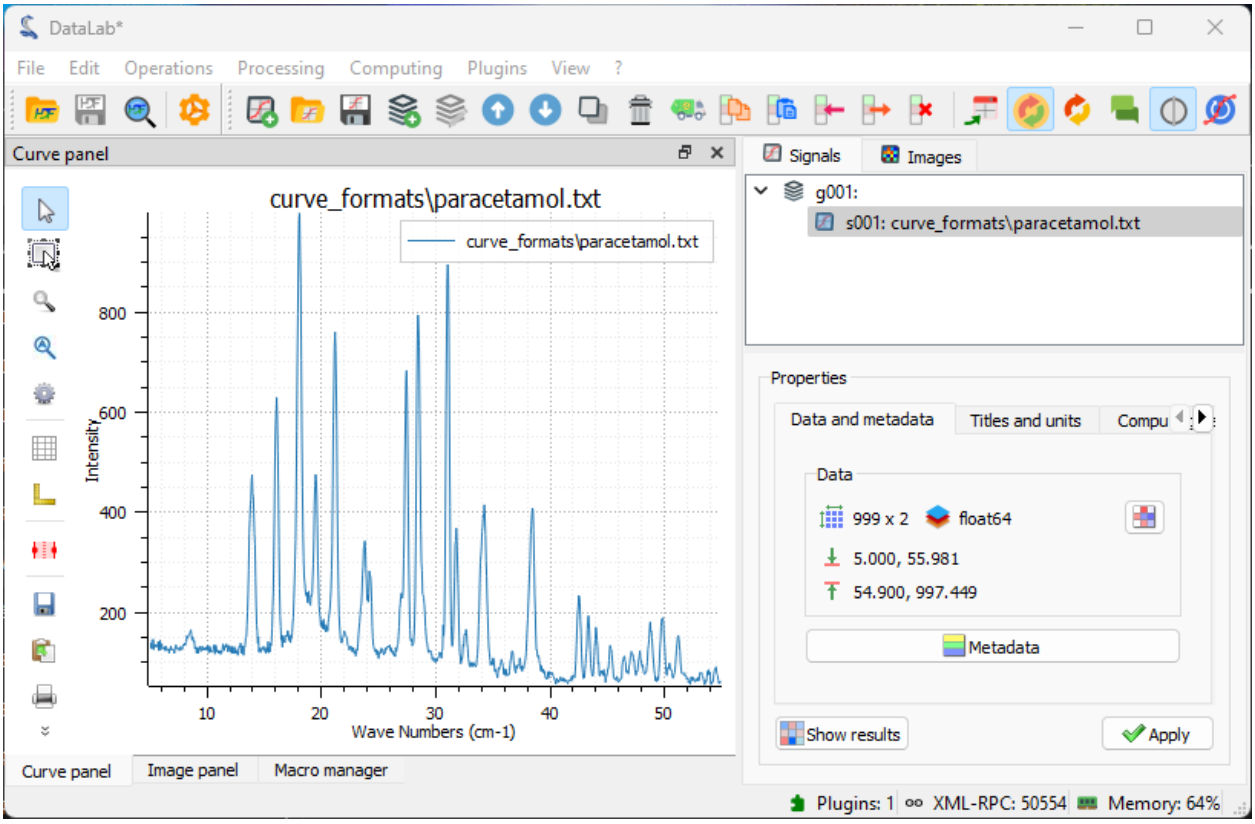


FIG. 7 – Le spectre est un signal 1D, il est donc affiché sous forme de courbe. L'axe horizontal est l'axe de l'énergie et l'axe vertical est l'axe de l'intensité.

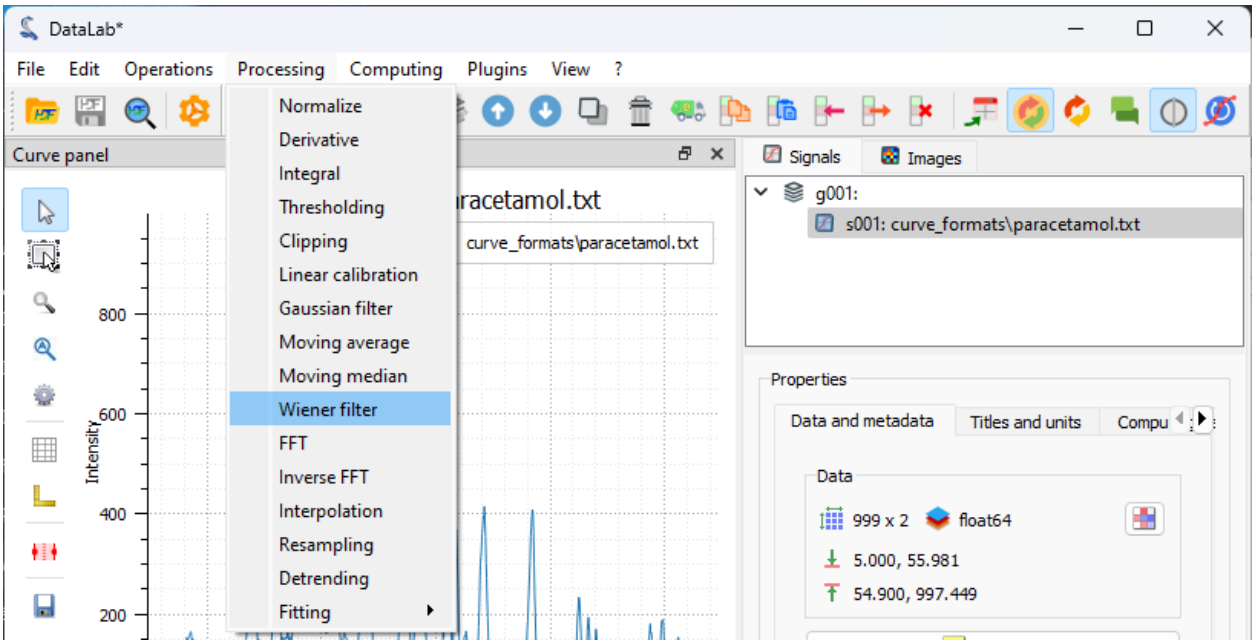


FIG. 8 – Ouvrir la fenêtre de filtre avec « Traitement > Filtre de Wiener ».

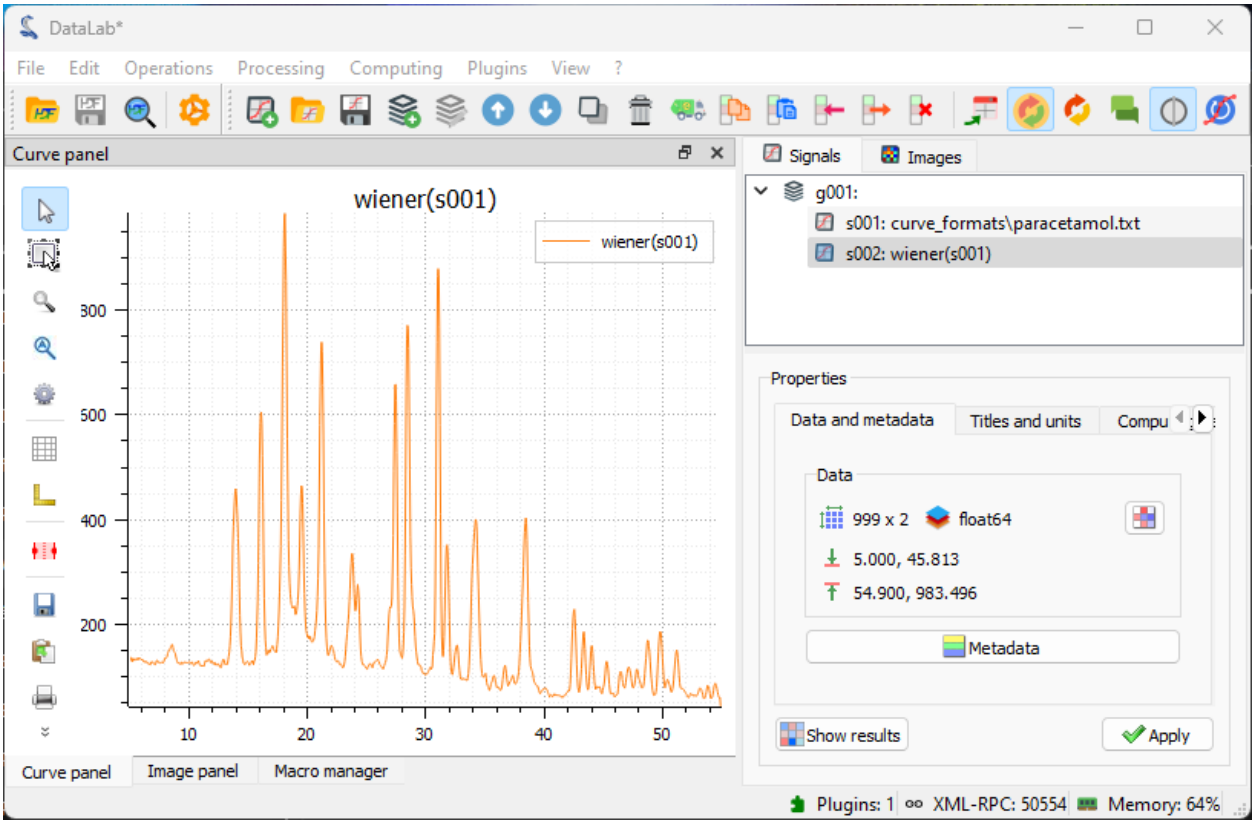


FIG. 9 – Le résultat du filtre est affiché dans la fenêtre principale.

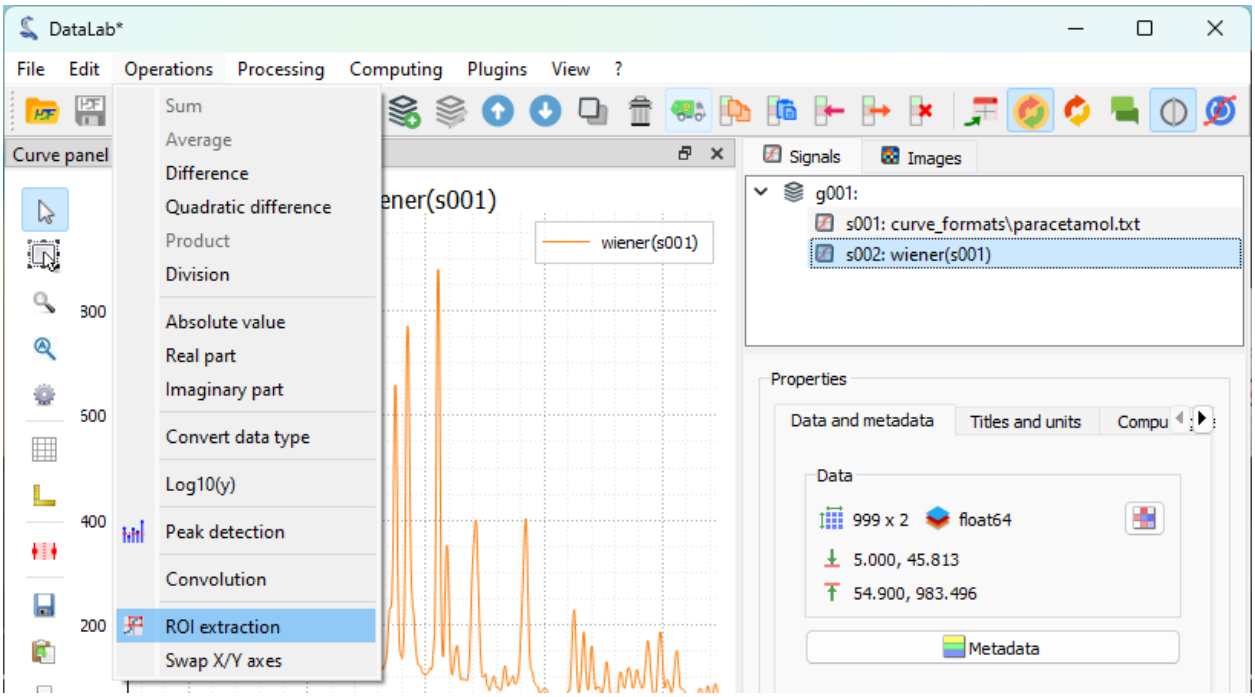


FIG. 10 – Ouvrir la fenêtre d'extraction de ROI avec « Opérations > Extraction de ROI ».

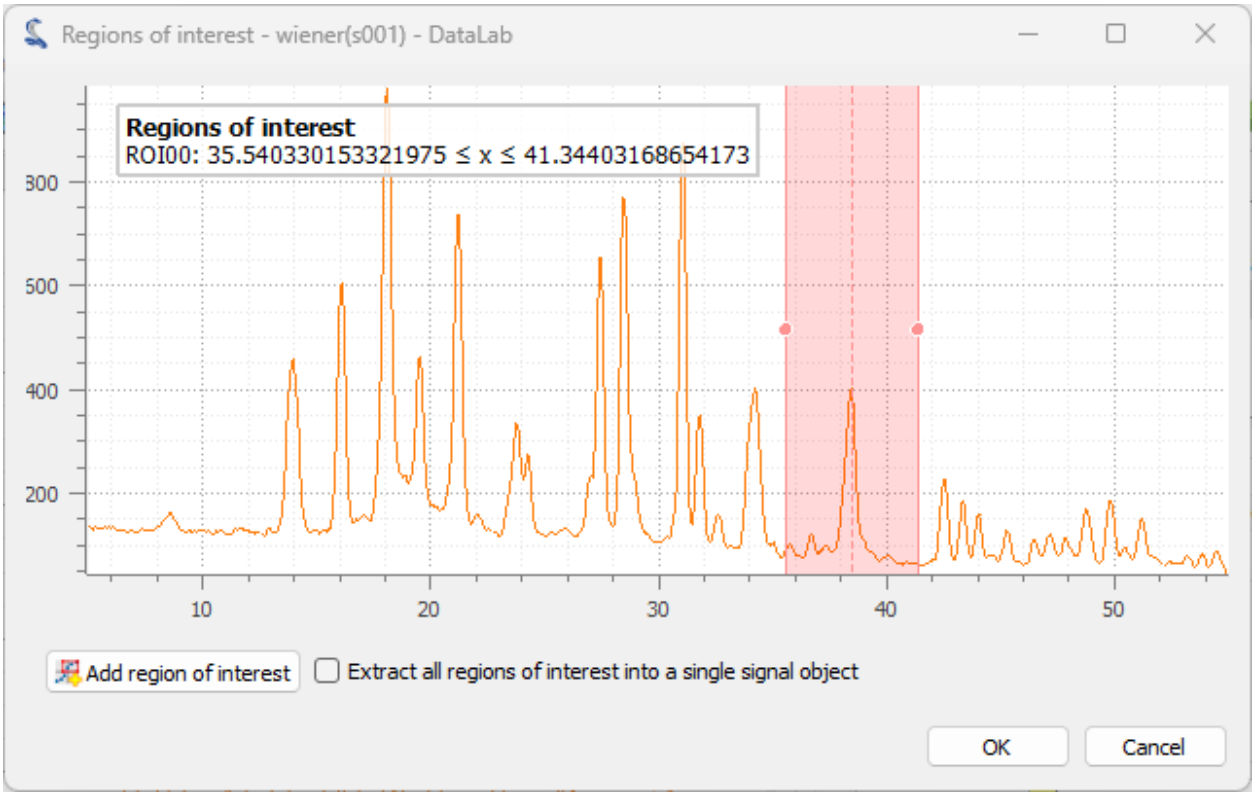


FIG. 11 – La boîte de dialogue « Régions d'intérêt » est affichée. Cliquez sur « Ajouter une région d'intérêt » et redimensionnez la fenêtre horizontale pour sélectionner la zone. Ensuite, cliquez sur « OK ».

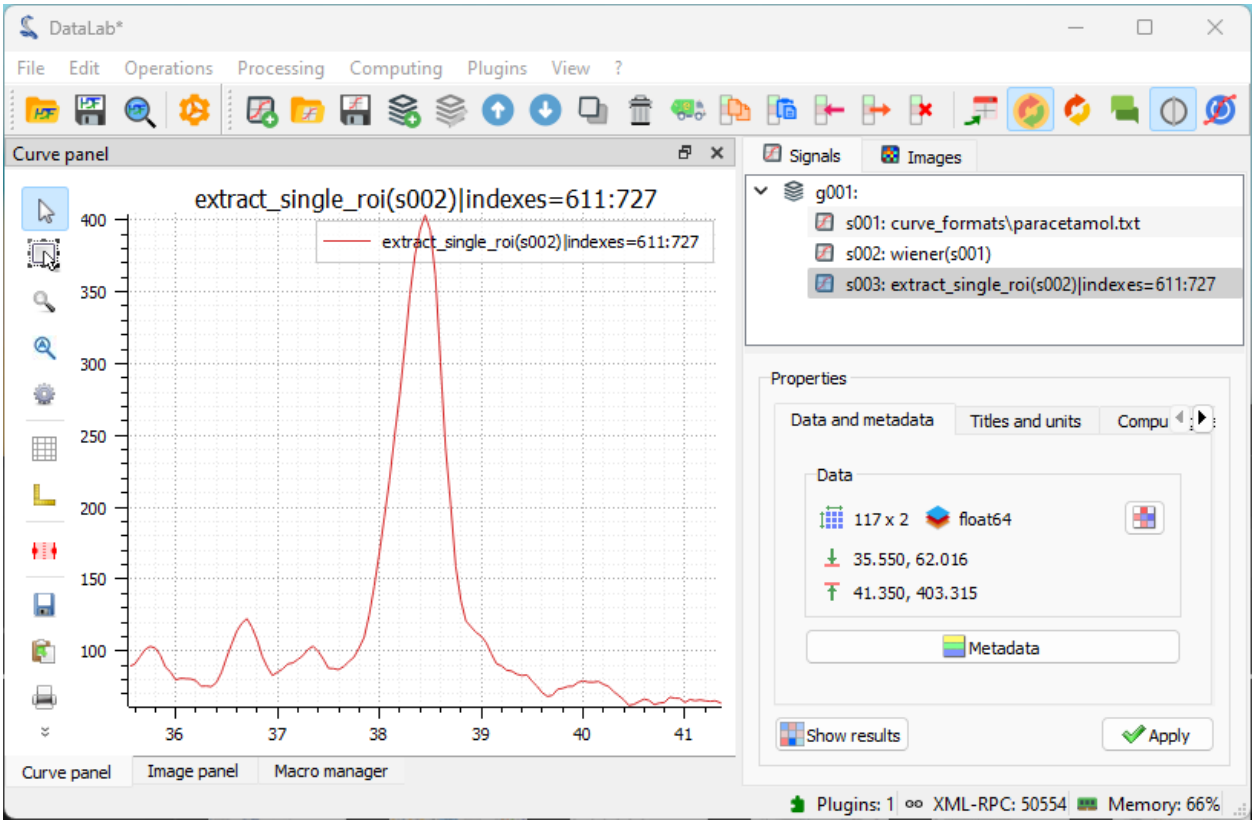


FIG. 12 – La région d'intérêt est affichée dans la fenêtre principale.

Essayons d'ajuster un modèle au spectre. Nous utiliserons un modèle gaussien, qui est un modèle typiquement utilisé pour ajuster un pic dans un spectre.

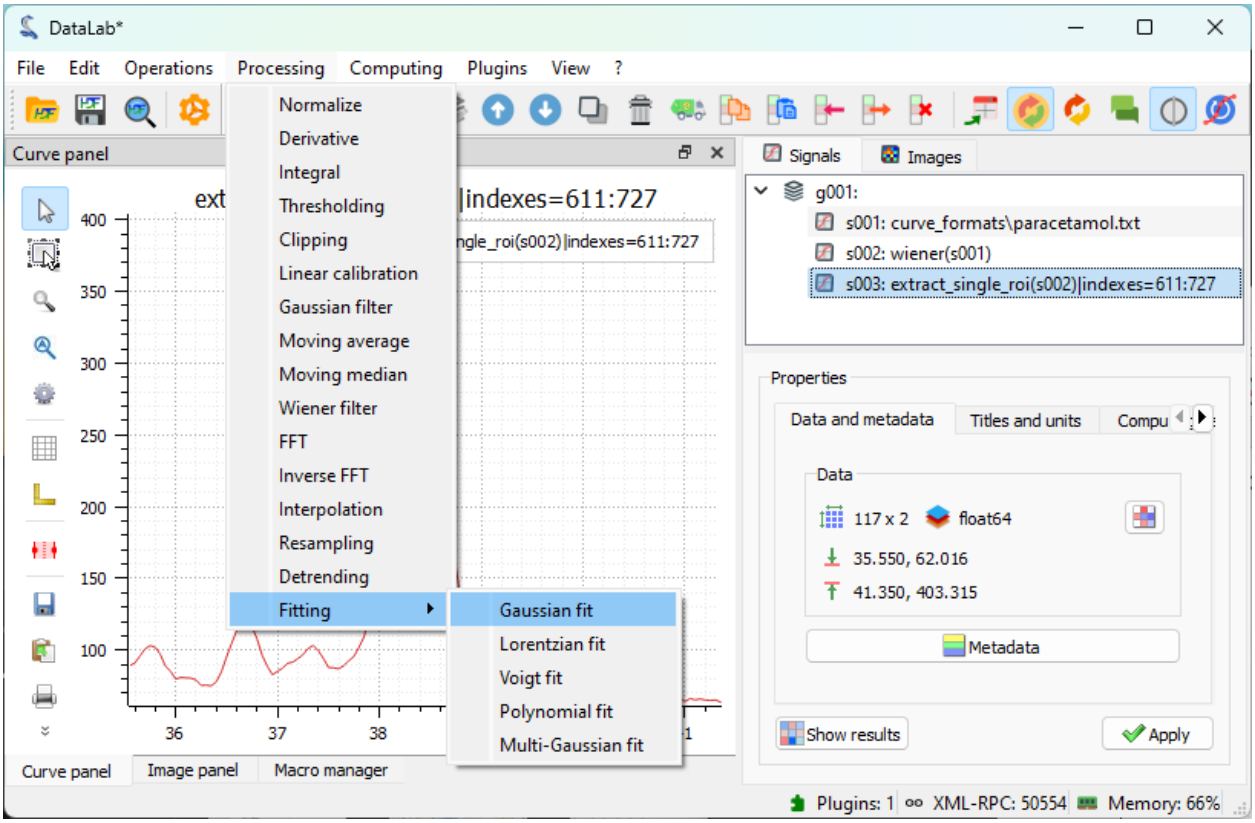


FIG. 13 – Ouvrir la fenêtre d'ajustement du modèle avec « Traitement > Ajustement > Ajustement gaussien ».

Pour faire la démonstration d'une autre fonction de traitement, nous pouvons également essayer la fonctionnalité d'élimination de tendance.

Lors de l'analyse d'un spectre, il peut être utile d'essayer d'identifier les pics dans le spectre. Nous pouvons le faire en ajustant un modèle multi-gaussien au spectre, en utilisant la fonction « Traitement > Ajustement > Ajustement multi-gaussien ».

Nous aurions également pu utiliser la fonction « Détection de pics » du menu « Opérations » pour détecter les pics dans le spectre.

Enfin, nous pouvons sauvegarder l'espace de travail dans un fichier. L'espace de travail contient tous les signaux qui ont été chargés dans DataLab, ainsi que les résultats du traitement. Il contient également les paramètres de visualisation (couleurs de courbe, etc.).

Si vous voulez charger à nouveau l'espace de travail, vous pouvez utiliser le « Fichier > Ouvrir un fichier HDF5… » (ou le bouton dans la barre d'outils) pour charger l'ensemble de l'espace de travail, ou le « Fichier > Parcourir un fichier HDF5… » (ou le bouton dans la barre d'outils) pour charger uniquement une sélection d'ensembles de données de l'espace de travail.

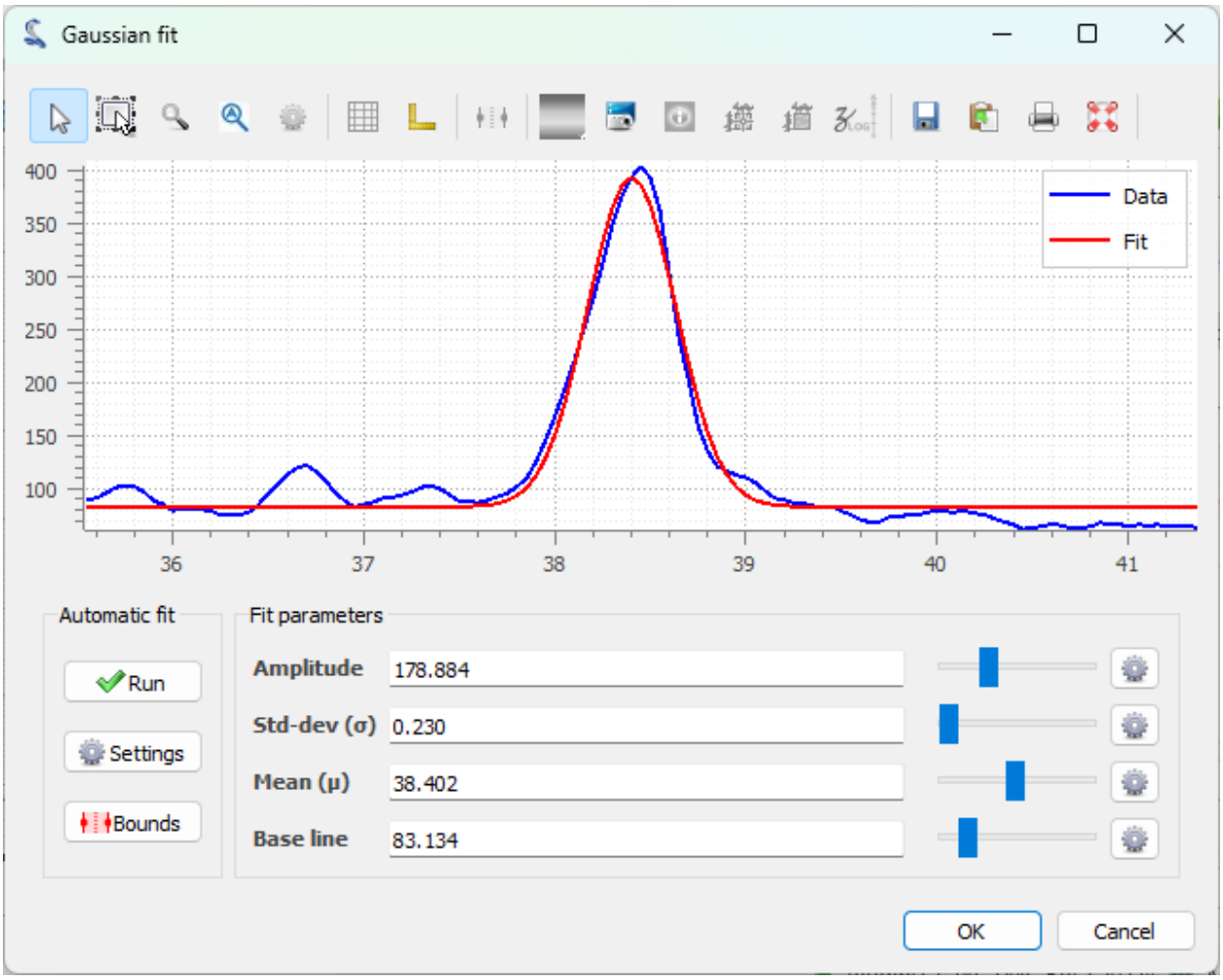


FIG. 14 – La boîte de dialogue « Ajustement gaussien » est affichée. Un ajustement automatique est effectué par défaut. Cliquez sur « OK » (ou essayez éventuellement d'ajuster le modèle manuellement en ajustant les paramètres ou les curseurs, ou essayez de modifier les paramètres d'ajustement automatique).

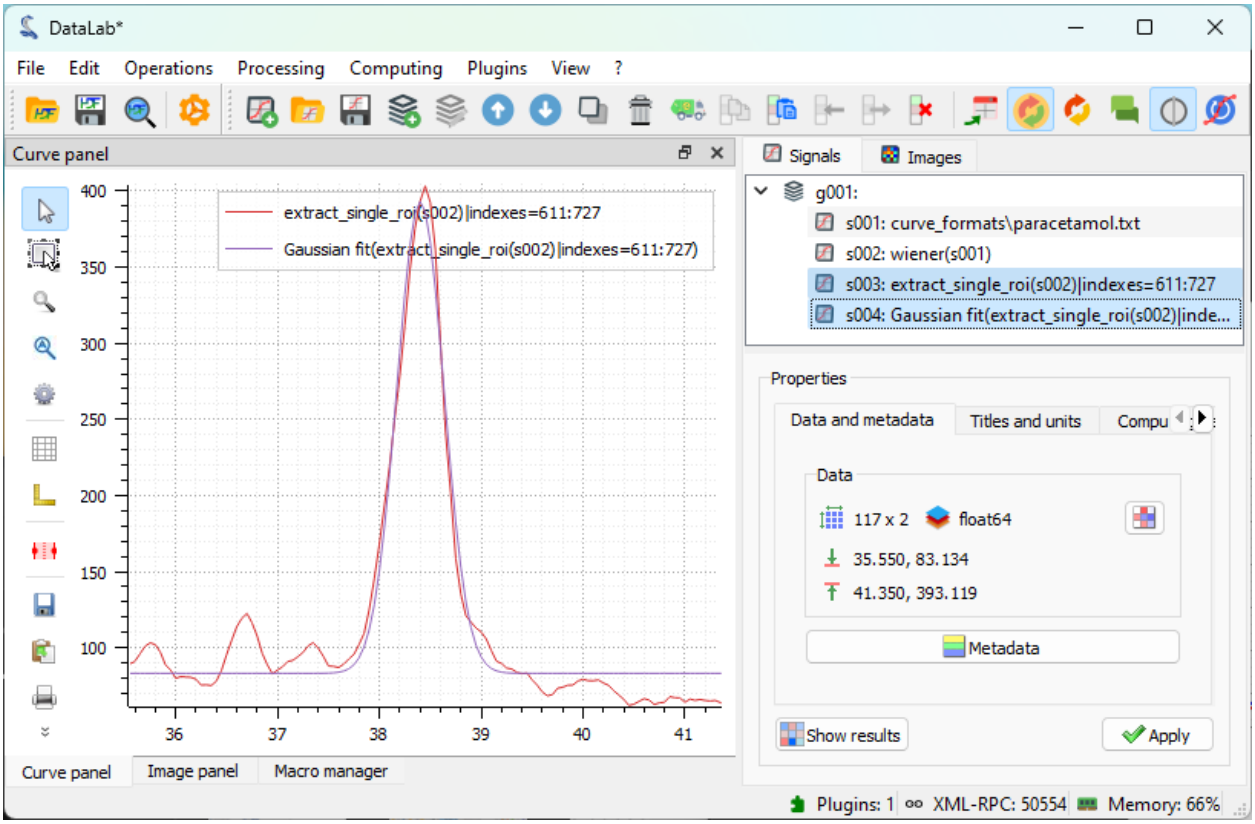


FIG. 15 – Le résultat de l'ajustement est affiché dans la fenêtre principale. Ici, nous avons sélectionné à la fois le spectre et l'ajustement dans le panneau « Signaux » à droite, donc les deux sont affichés dans le panneau de visualisation à gauche.

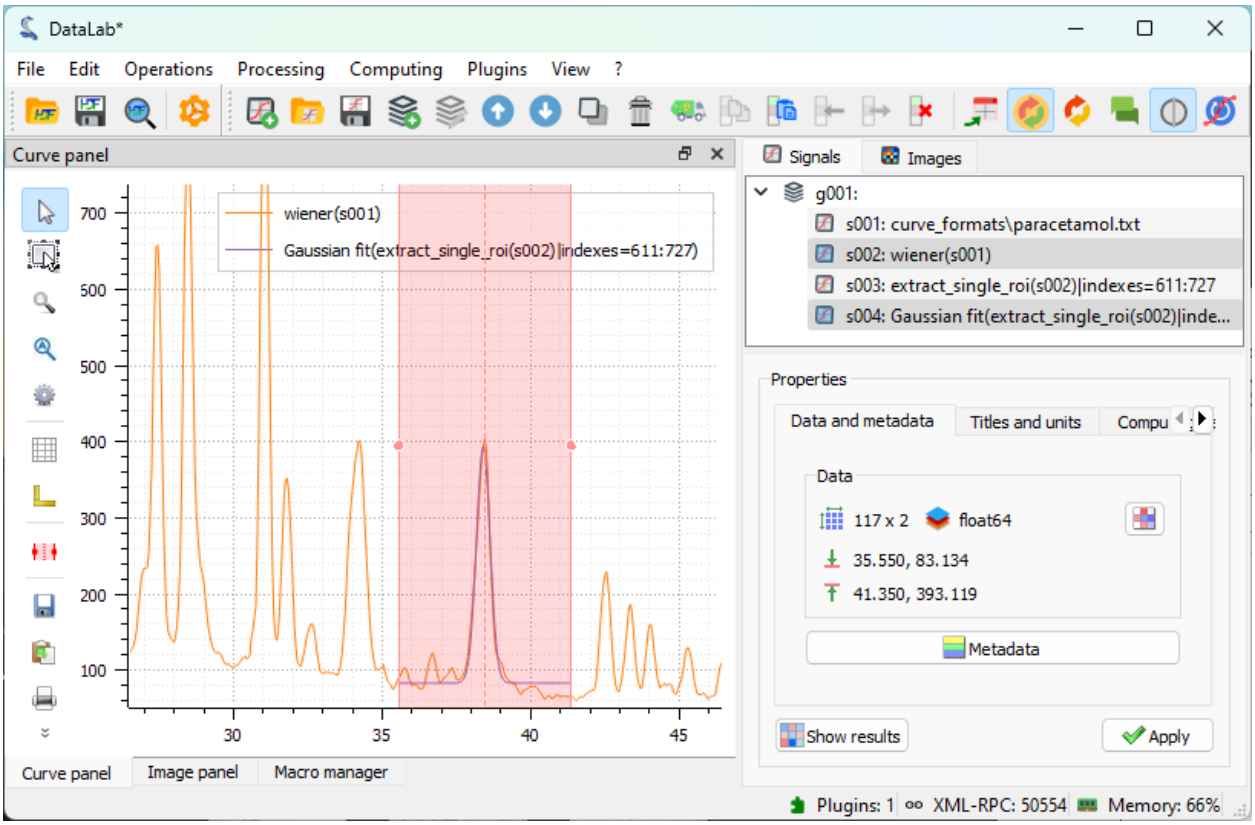


FIG. 16 – Nous pouvons également sélectionner le spectre complet et l'ajustement dans le panneau « Signaux » à droite, de sorte que les deux soient affichés dans le panneau de visualisation à gauche, si cela a un sens pour l'analyse que nous voulons effectuer. Notez que la visualisation du spectre complet contient également la région d'intérêt que nous avons extraite précédemment.

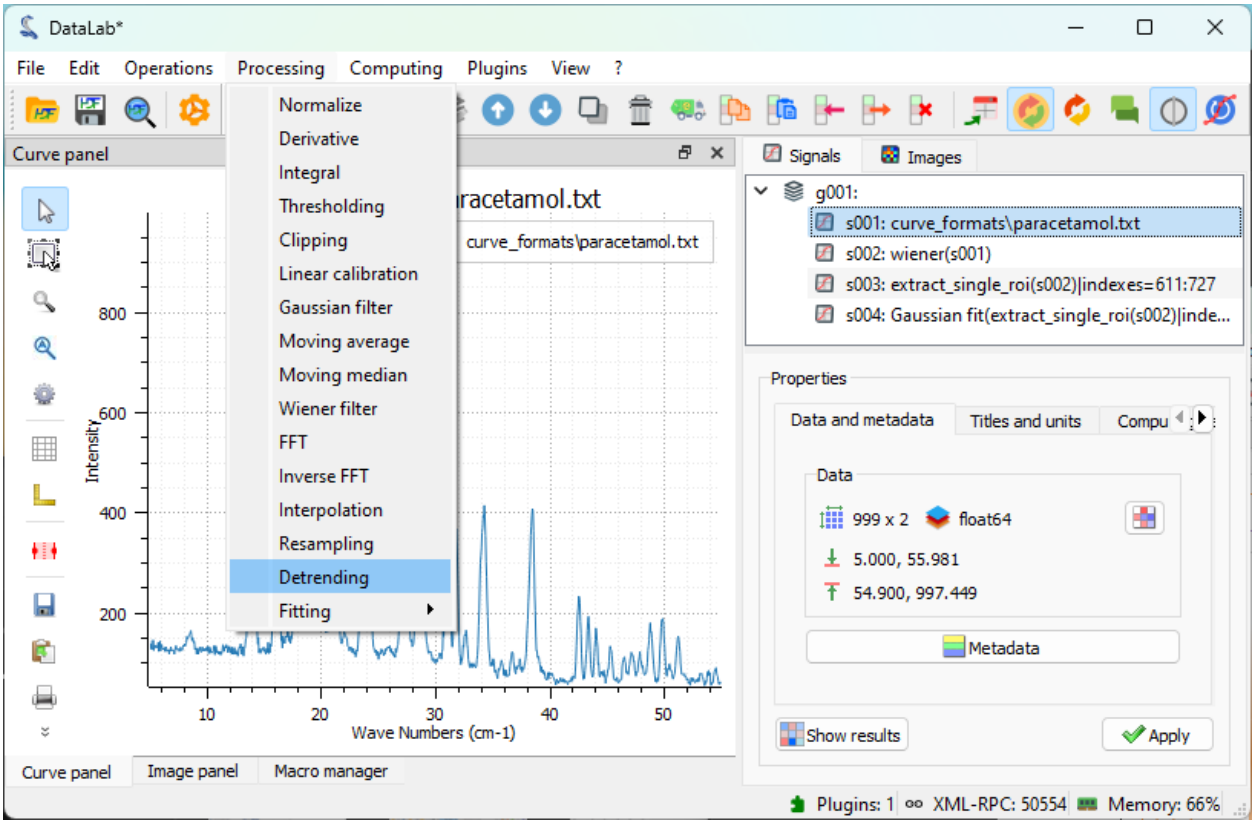


FIG. 17 – Exécutez la fonction « Traitement > Elimination de tendance ».

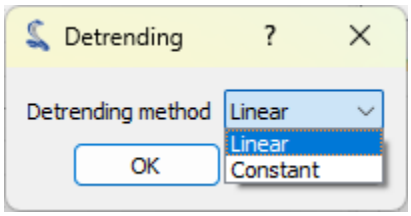


FIG. 18 – Nous choisissons une méthode d'élimination de tendance linéaire, et nous cliquons sur « OK ».

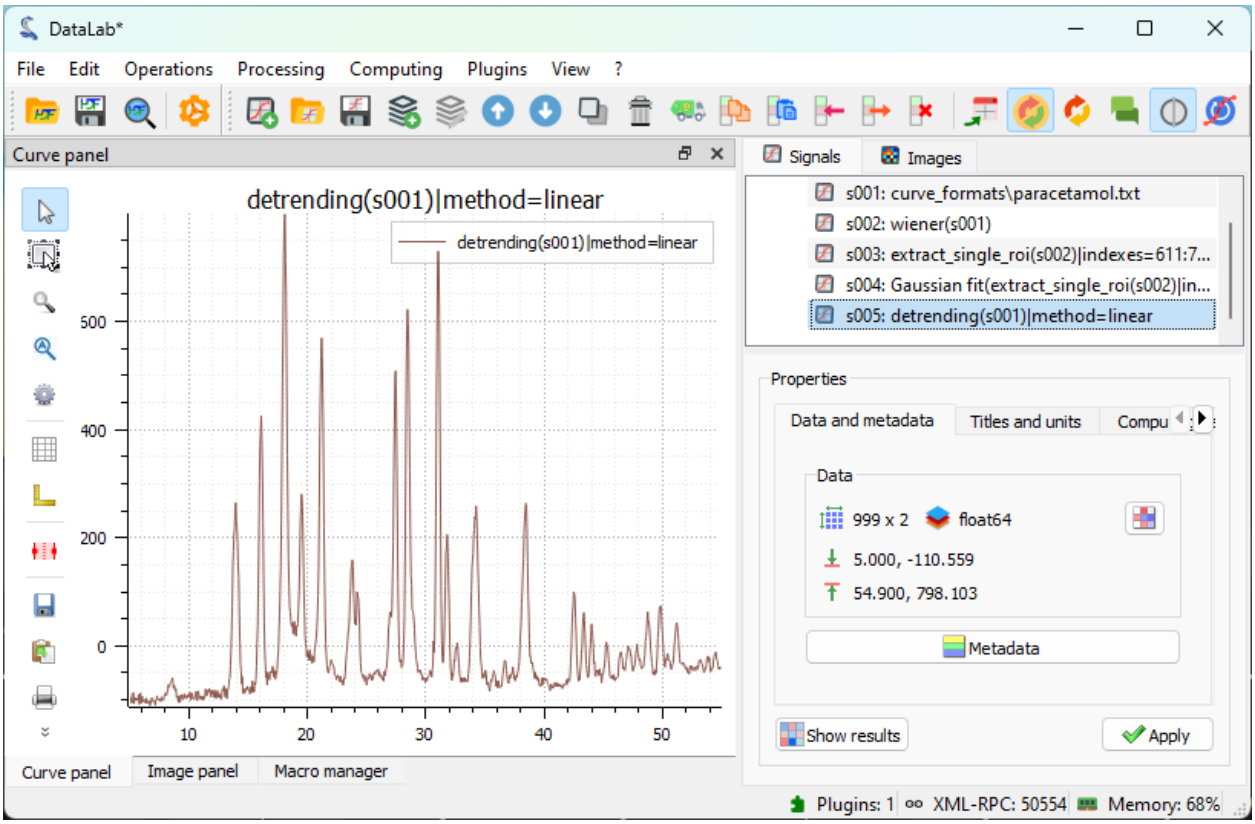


FIG. 19 – Le résultat du traitement est affiché dans la fenêtre principale (dans ce cas précis, l'élimination de tendance n'est pas nécessairement appropriée, mais c'est juste pour démontrer la fonctionnalité).

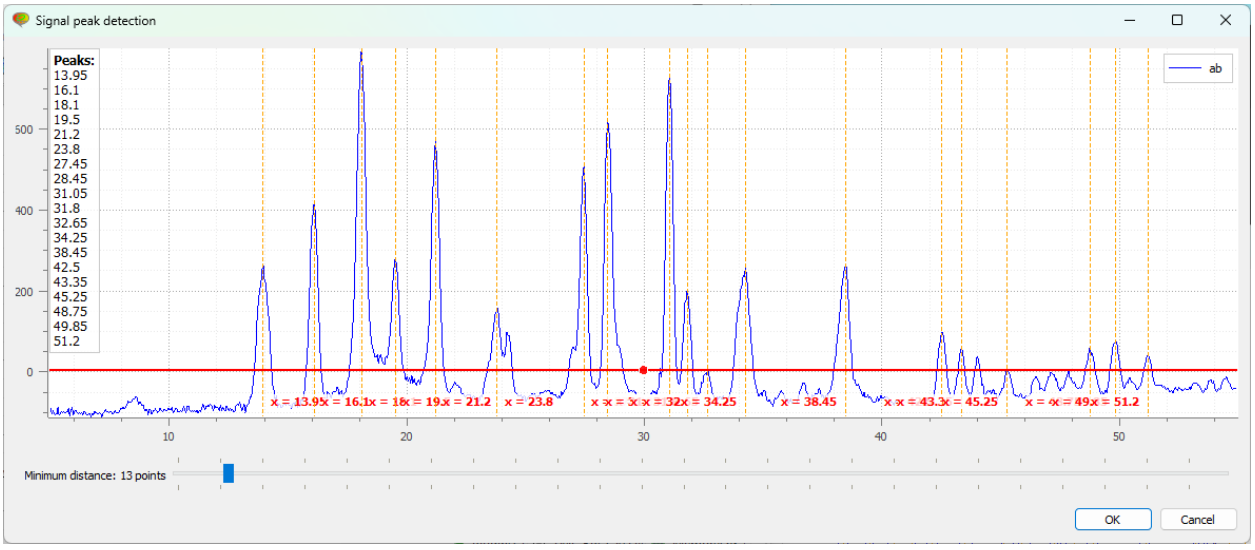


FIG. 20 – Tout d'abord, une boîte de dialogue « Détection de pics de signal » est affichée. Nous pouvons ajuster la position du curseur vertical pour sélectionner le seuil de détection des pics, ainsi que la distance minimale entre deux pics. Ensuite, nous cliquons sur « OK ».

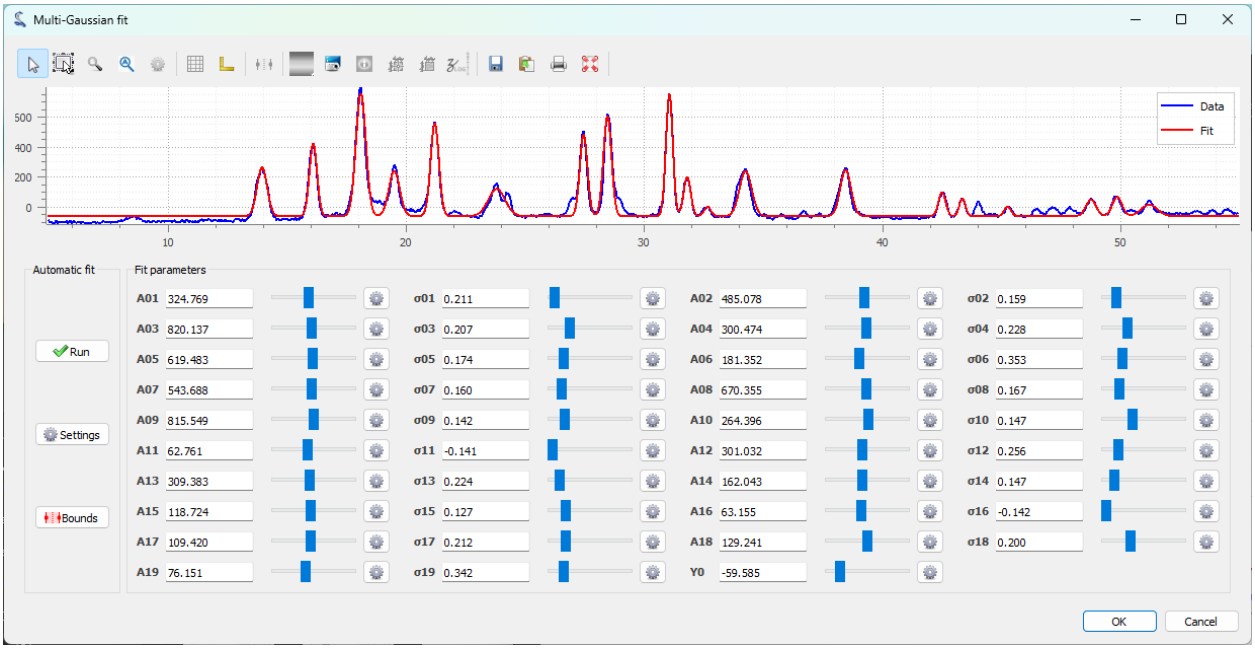


FIG. 21 – La boîte de dialogue « Ajustement multi-gaussien » est affichée. Un ajustement automatique est effectué par défaut. Cliquez sur « OK » (ou essayez éventuellement d'ajuster le modèle manuellement en ajustant les paramètres ou les curseurs, ou essayez de modifier les paramètres d'ajustement automatique).

Détection de taches sur une image

Cet exemple montre comment détecter des taches sur une image avec DataLab, et couvre également d'autres fonctionnalités telles que le système de plugins :

- Ajouter un nouveau plugin à DataLab
- Débruitage d'une image
- Détecter des taches sur une image
- Enregistrer l'espace de travail dans un fichier

Tout d'abord, nous ouvrons DataLab, et ouvrons la boîte de dialogue des paramètres (en utilisant « Fichier > Paramètres… », ou l'icône dans la barre d'outils).

Voir aussi :

Le système de plugins est décrit dans la section *Plugins*.

Ajoutons le plugin *cdl_example_imageproc.py* à DataLab (c'est un plugin d'exemple qui est livré avec le package source de DataLab, ou peut être téléchargé ici sur GitHub).

Si nous fermons et rouvrons DataLab, nous pouvons voir que le plugin est maintenant disponible dans le menu « Plugins » : il y a une nouvelle entrée « Extract blobs (example) ».

Pour information, l'image est générée par le plugin en utilisant le code suivant :

```
def generate_test_image(self) -> None:
    """Generate test image"""
    # Create a NumPy array:
    arr = np.random.normal(10000, 1000, (2048, 2048))
    for _ in range(10):
        row = np.random.randint(0, arr.shape[0])
        col = np.random.randint(0, arr.shape[1])
```

(suite sur la page suivante)

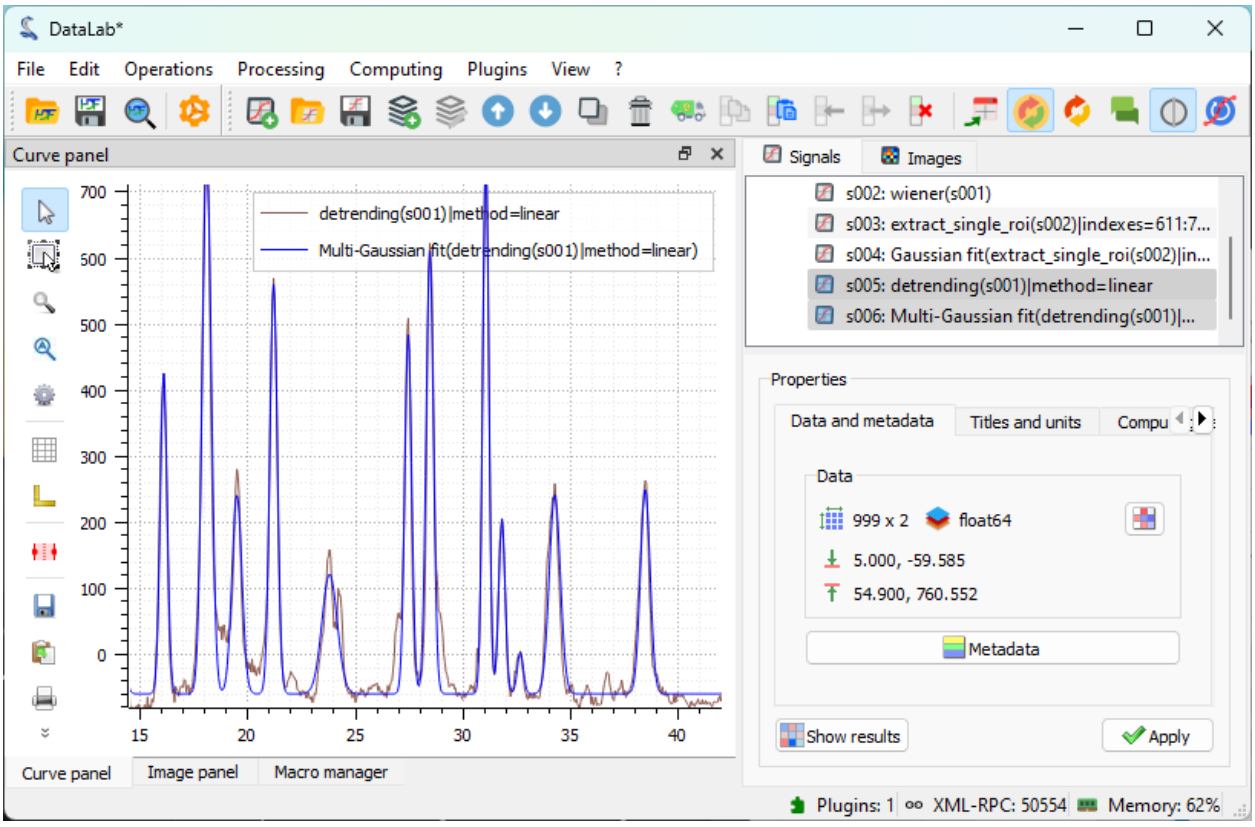


FIG. 22 – Le résultat de l'ajustement est affiché dans la fenêtre principale. Ici, nous avons sélectionné à la fois le spectre et l'ajustement dans le panneau « Signaux » à droite, donc les deux sont affichés dans le panneau de visualisation à gauche.

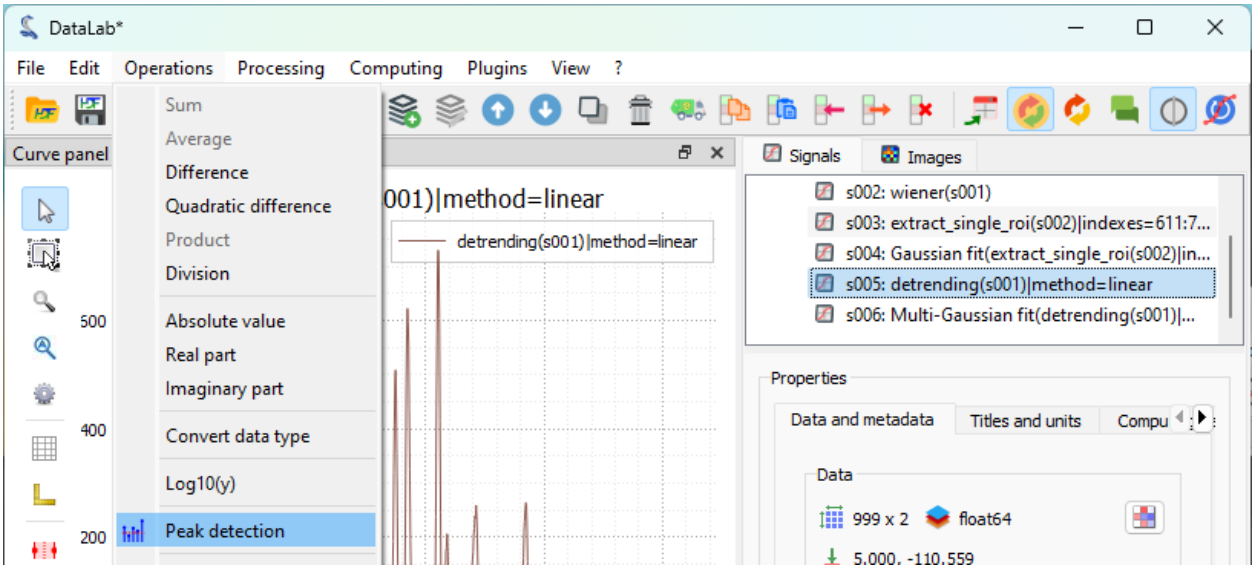


FIG. 23 – Ouvrir la fenêtre « Détection de pics » avec « Opérations > Détection de pics ».

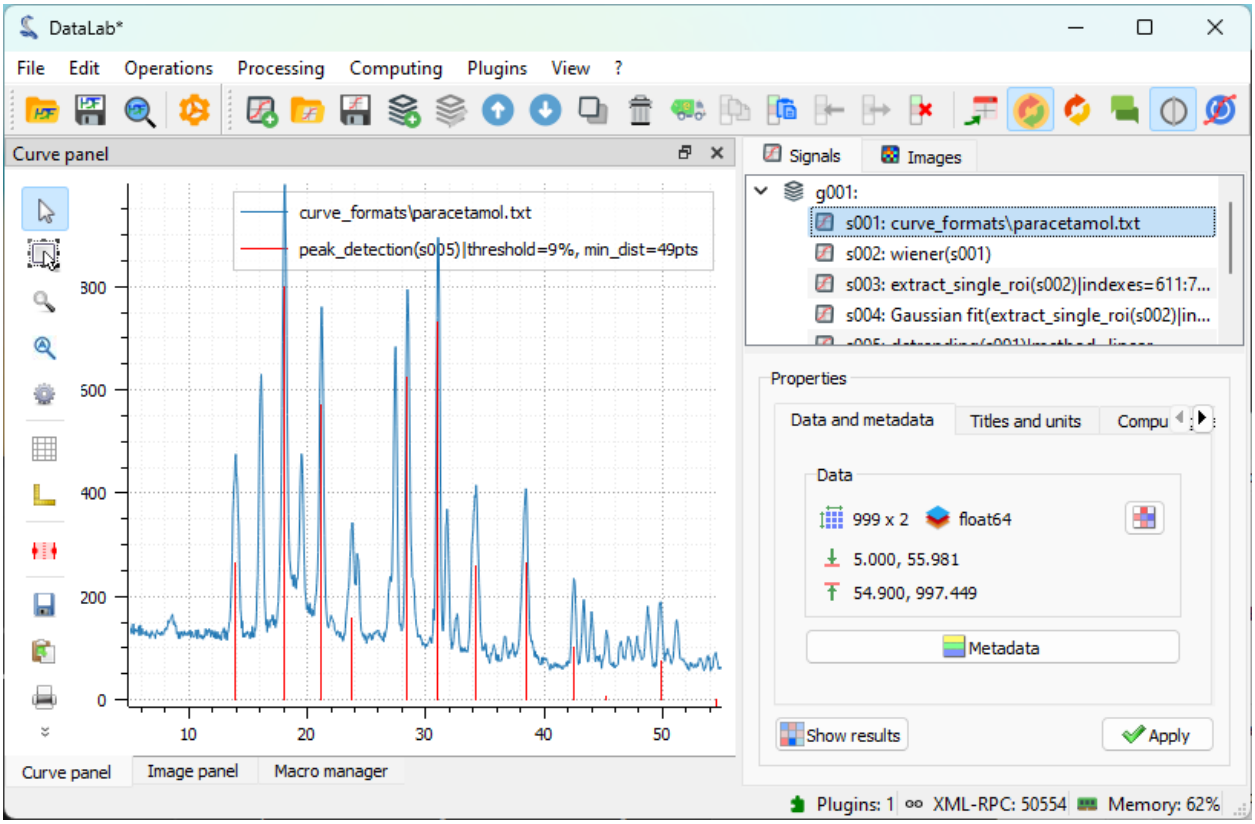


FIG. 24 – Après avoir ajusté les paramètres de la boîte de dialogue de détection de pics (même boîte de dialogue que celle utilisée pour l'ajustement multi-gaussien), cliquez sur « OK ». Ensuite, nous sélectionnons la « détection_de_pics » et le spectre d'origine dans le panneau « Signaux » à droite, de sorte que les deux soient affichés dans le panneau de visualisation à gauche.

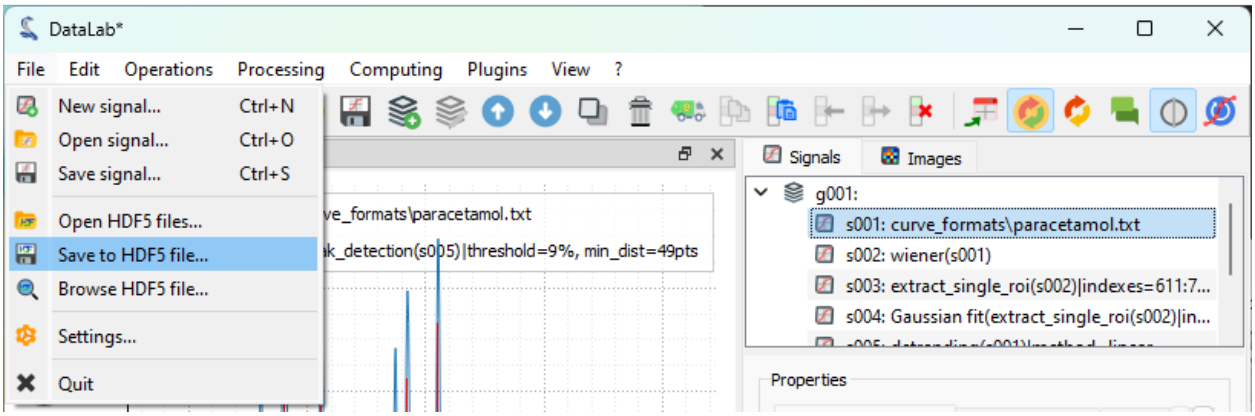


FIG. 25 – Sauvegarder l'espace de travail dans un fichier avec « Fichier > Sauvegarder dans un fichier HDF5… », ou le bouton dans la barre d'outils.

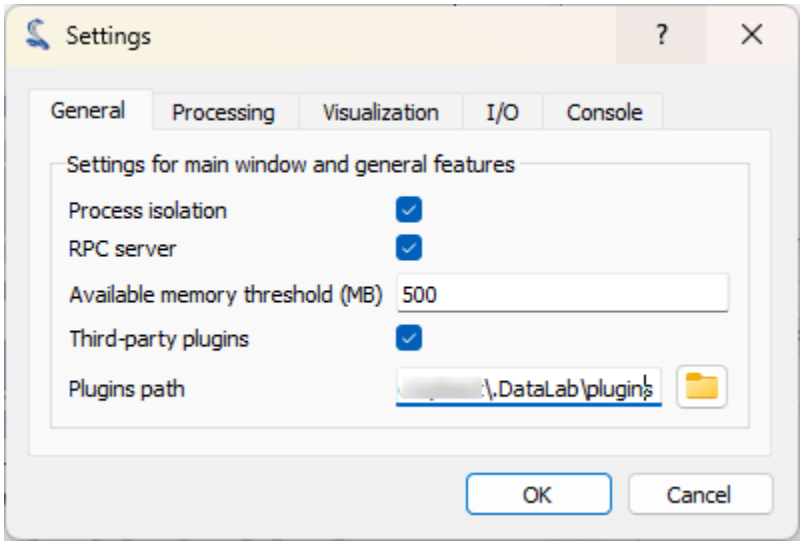


FIG. 26 – Dans l'onglet « Général », nous pouvons voir le champ « Chemin des plugins ». C'est le chemin où DataLab recherchera les plugins. Nous pouvons ajouter un nouveau plugin en copiant/collant le fichier du plugin dans ce répertoire.

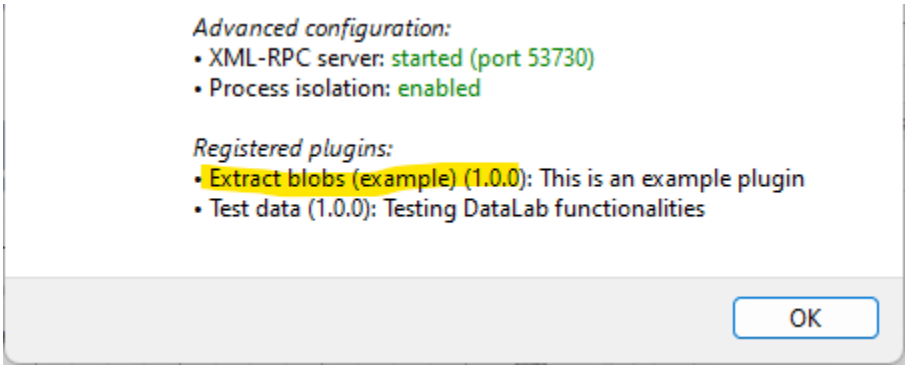


FIG. 27 – La boîte de dialogue « À propos de DataLab » affiche la liste des plugins disponibles.

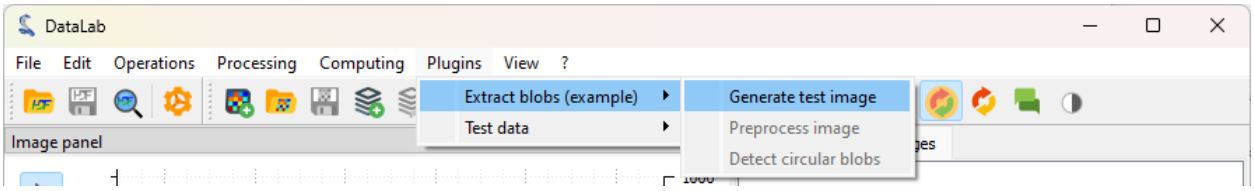


FIG. 28 – Cliquons sur « Extract blobs (example) > Generate test image »

(suite de la page précédente)

```
        rr, cc = skimage.draw.disk((row, col), 40, shape=arr.shape)
        arr[rr, cc] -= np.random.randint(5000, 6000)
    icenter = arr.shape[0] // 2
    rr, cc = skimage.draw.disk((icenter, icenter), 200, shape=arr.shape)
    arr[rr, cc] -= np.random.randint(5000, 8000)
    data = np.clip(arr, 0, 65535).astype(np.uint16)

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

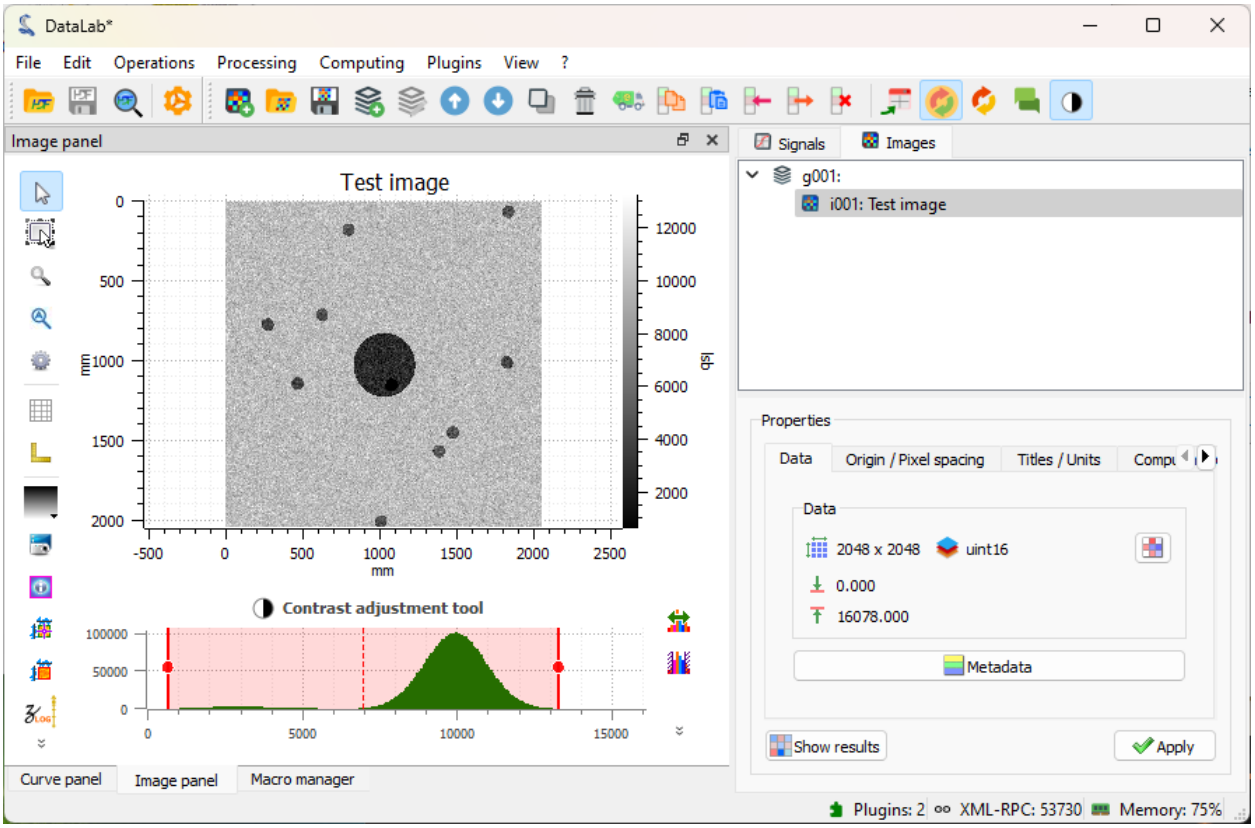


FIG. 29 – Le plugin a généré une image de test, et l'a ajoutée au panneau « Images ». L'image montre quelques taches, avec un disque sombre central, et un fond bruité.

Ce plugin a d'autres fonctionnalités, telles que le débruitage de l'image, et la détection de taches sur l'image, mais nous ne les couvrirons pas ici : nous utiliserons les mêmes fonctionnalités natives de DataLab que le plugin, manuellement.

L'image est un peu bruitée, et aussi assez grande. Réduisons la taille de l'image tout en la débruitant un peu en la binarisant par un facteur de 2.

Appliquons un filtre médian mobile à l'image, pour la débruiter un peu plus.

A présent, détectons les taches sur l'image.

Note : Si vous souhaitez afficher à nouveau les résultats de calcul, vous pouvez sélectionner l'entrée « Afficher les résultats » dans le menu « Calcul », ou le bouton « Afficher les résultats » , en dessous de la liste des images :

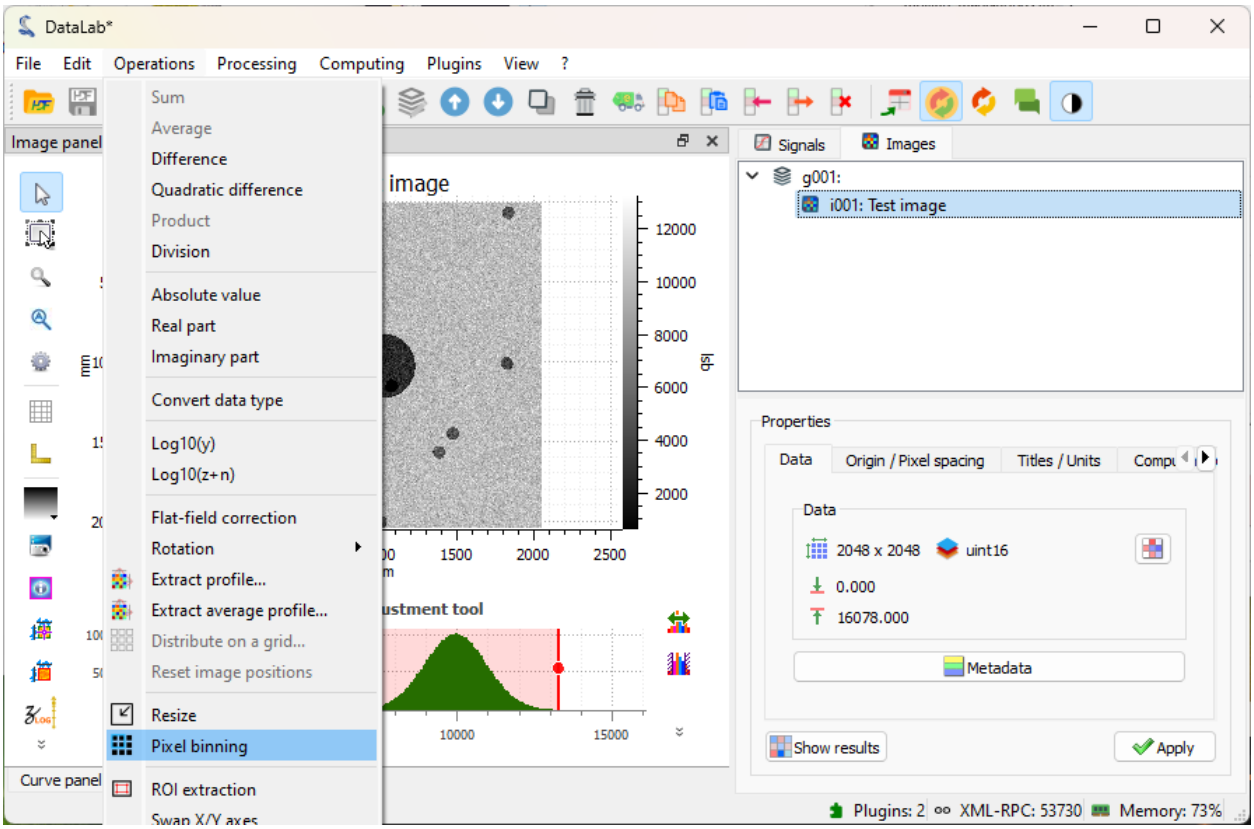


FIG. 30 – Cliquons sur « Opérations > Pixel binning ».

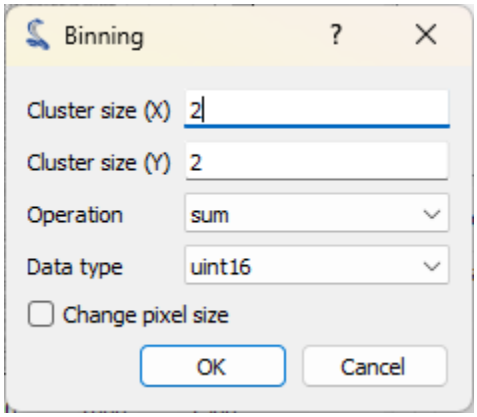


FIG. 31 – La boîte de dialogue « Binning » s'ouvre. Réglez le facteur de binning à 2, et cliquez sur « OK ».

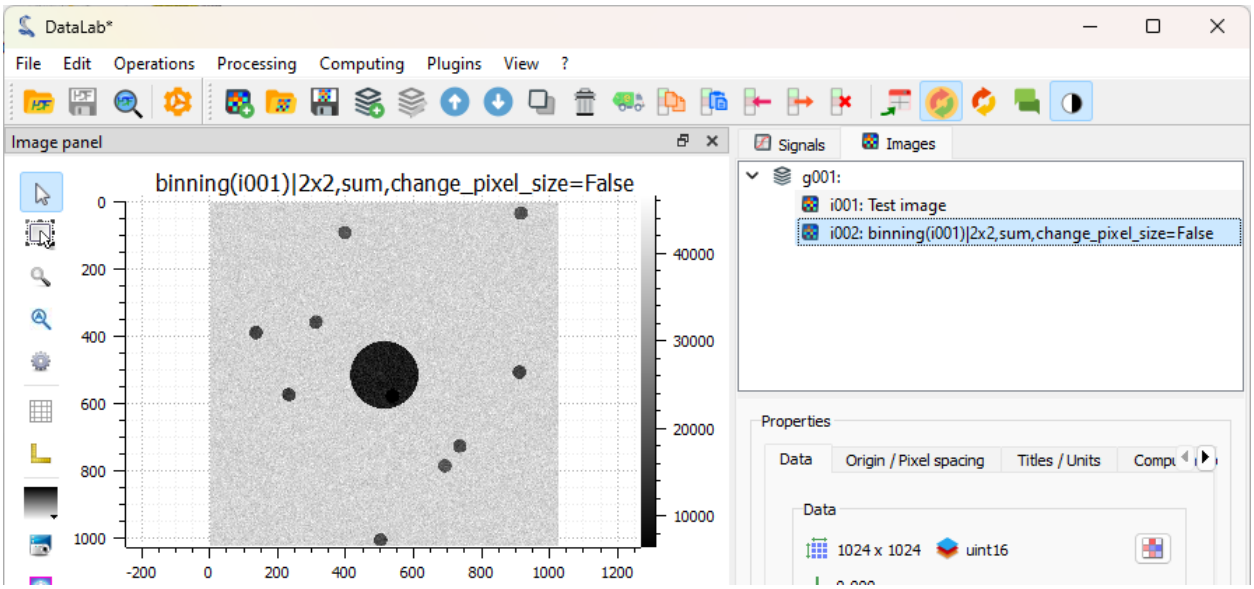


FIG. 32 – L'image binnée est ajoutée au panneau « Images ». Il est maintenant plus facile de voir les taches (même si elles étaient déjà assez visibles sur l'image d'origine : c'est juste un exemple), et l'image sera plus rapide à traiter.

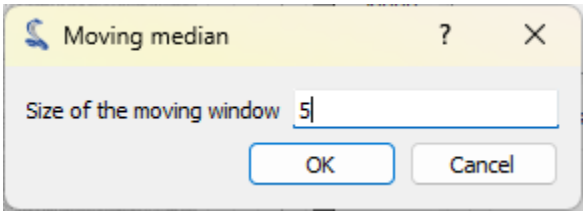


FIG. 33 – Cliquons sur l'entrée « Processing > Moving median », et réglez la taille de la fenêtre sur 5.

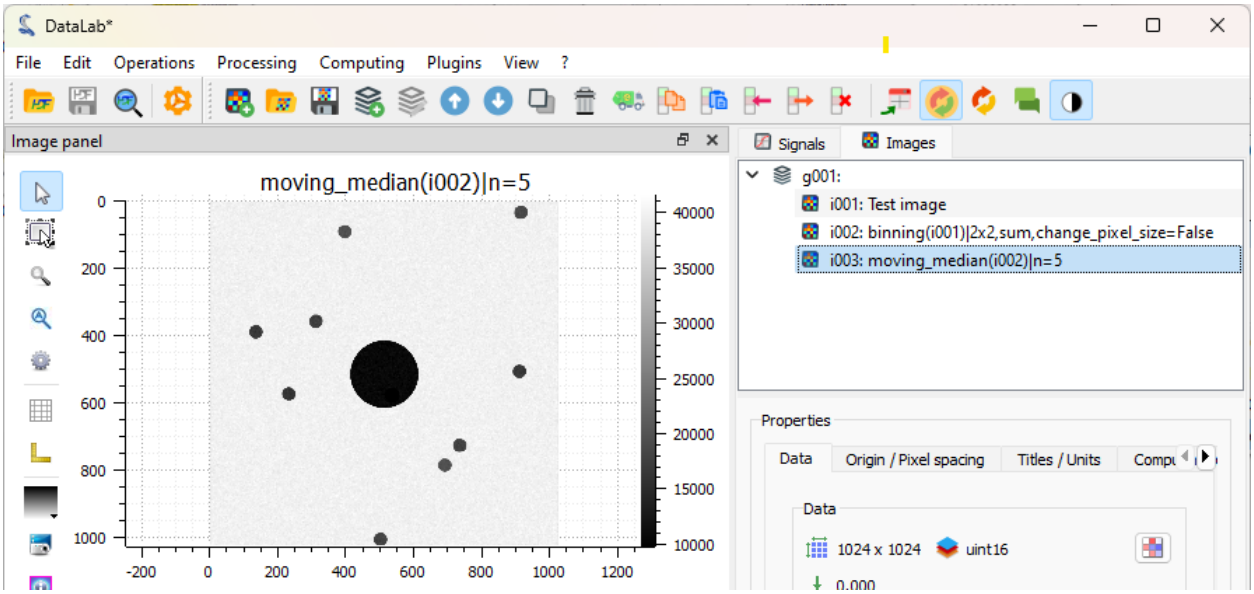


FIG. 34 – L'image filtrée est ajoutée au panneau « Images ». Le débruitage est assez efficace.

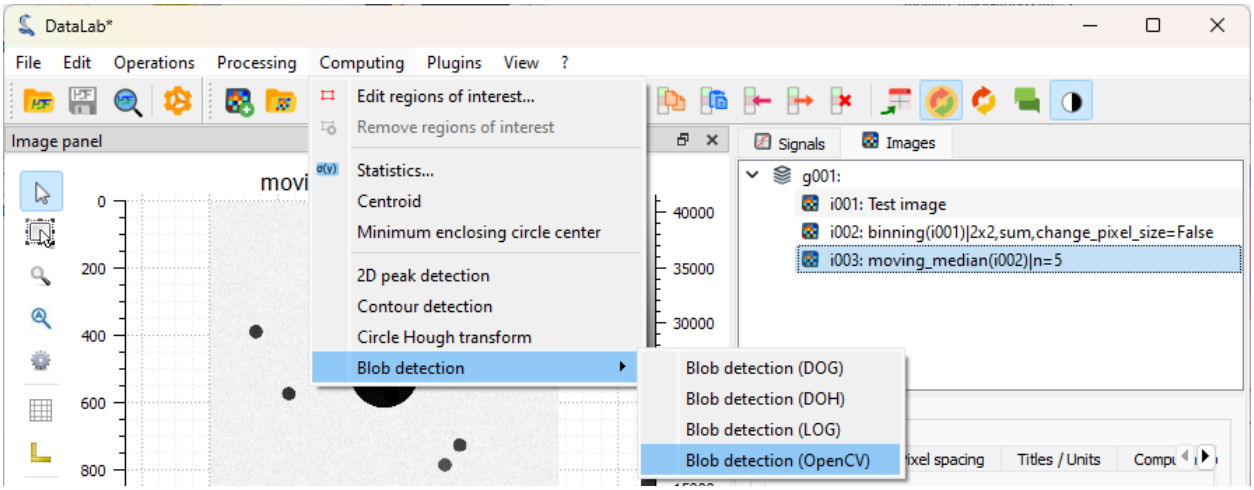


FIG. 35 – Cliquons sur « Computing > Blob detection > Blob detection (OpenCV) ».

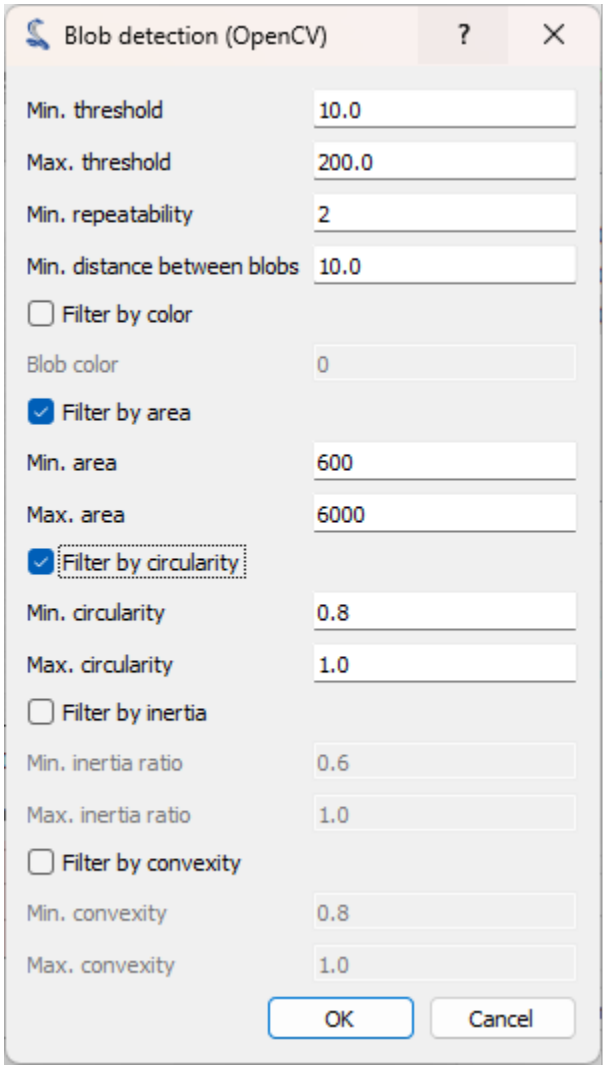


FIG. 36 – La boîte de dialogue « Blob detection (OpenCV) » s'ouvre. Réglez les paramètres comme indiqué sur la capture d'écran, et cliquez sur « OK ».

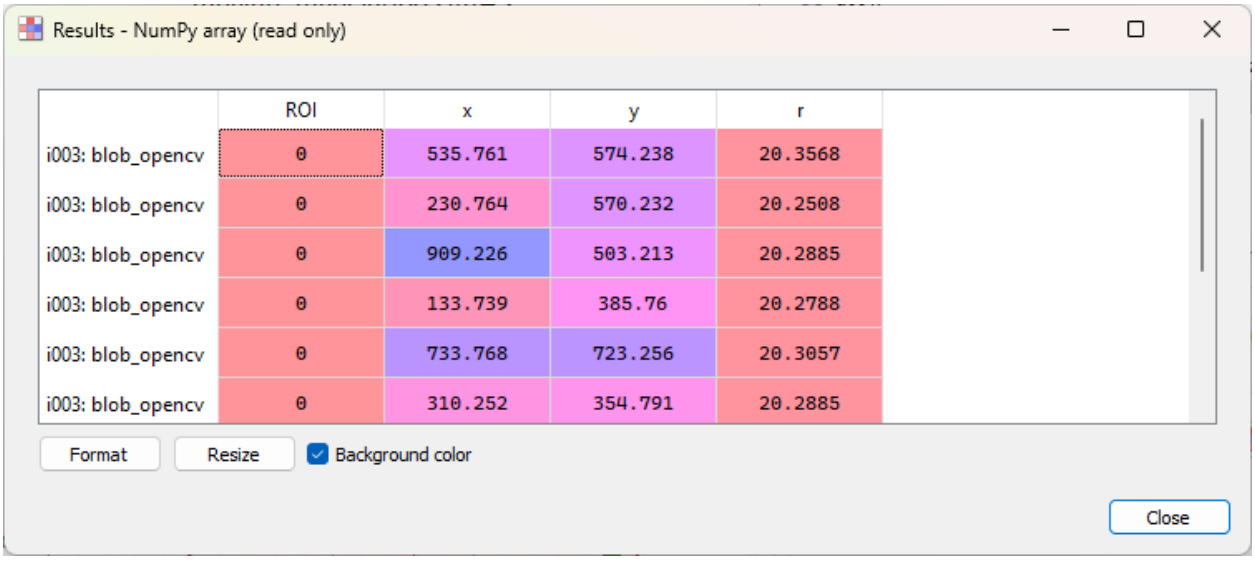
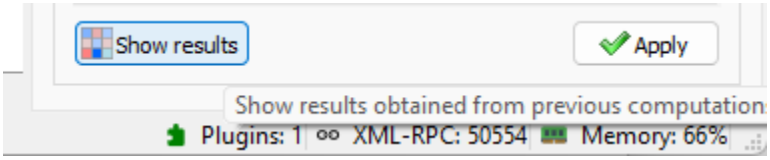


FIG. 37 – La boîte de dialogue « Résultats » s'ouvre, montrant les taches détectées : une ligne par tache, avec les coordonnées et le rayon de la tache.



Enfin, nous pouvons enregistrer l'espace de travail dans un fichier. L'espace de travail contient toutes les images qui ont été chargées dans DataLab, ainsi que les résultats de traitement. Il contient également les paramètres de visualisation (palettes de couleurs, contraste, etc.), les métadonnées et les annotations. Pour enregistrer l'espace de travail, cliquez sur « Fichier > Enregistrer dans un fichier HDF5… », ou sur le bouton dans la barre d'outils.

Si vous souhaitez charger à nouveau l'espace de travail, vous pouvez utiliser « Fichier > Ouvrir un fichier HDF5… » (ou le bouton dans la barre d'outils) pour charger l'ensemble de l'espace de travail, ou « Fichier > Parcourir un fichier HDF5… » (ou le bouton dans la barre d'outils) pour charger uniquement une sélection d'ensembles de données de l'espace de travail.

Mesure de franges de Fabry-Perot

Cet exemple montre comment mesurer des franges de Fabry-Perot à l'aide des fonctionnalités de traitement d'image de DataLab :

- Charger une image d'un interféromètre de Fabry-Perot
- Définir une région d'intérêt circulaire (ROI) autour de la frange centrale
- Détecter les contours dans la ROI et les ajuster à des cercles
- Afficher le rayon des cercles
- Annoter l'image
- Copier/coller la ROI dans une autre image
- Extraire le profil d'intensité le long de l'axe X
- Sauvegarder l'espace de travail

Tout d'abord, nous ouvrons DataLab et chargeons les images :

L'image sélectionnée est affichée dans la fenêtre principale. Nous pouvons zoomer en appuyant sur le bouton droit de la souris et en faisant glisser la souris vers le haut et vers le bas. Nous pouvons également déplacer l'image en appuyant

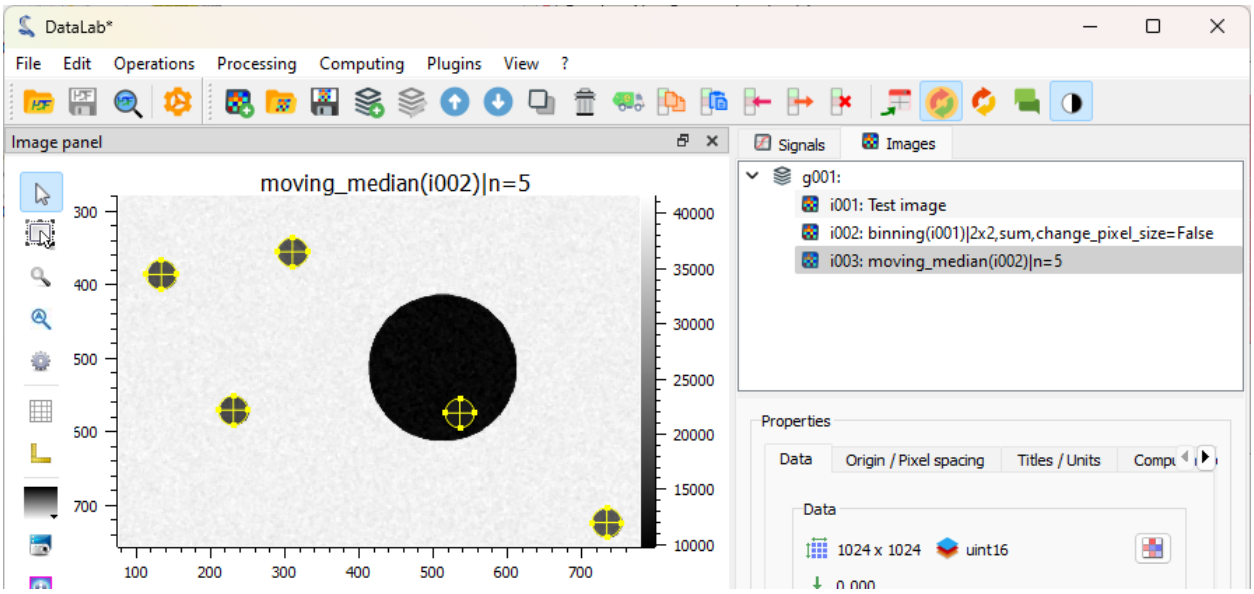


FIG. 38 – Les taches détectées sont également ajoutées aux métadonnées de l'image, et peuvent être vues dans le panneau de visualisation à gauche.

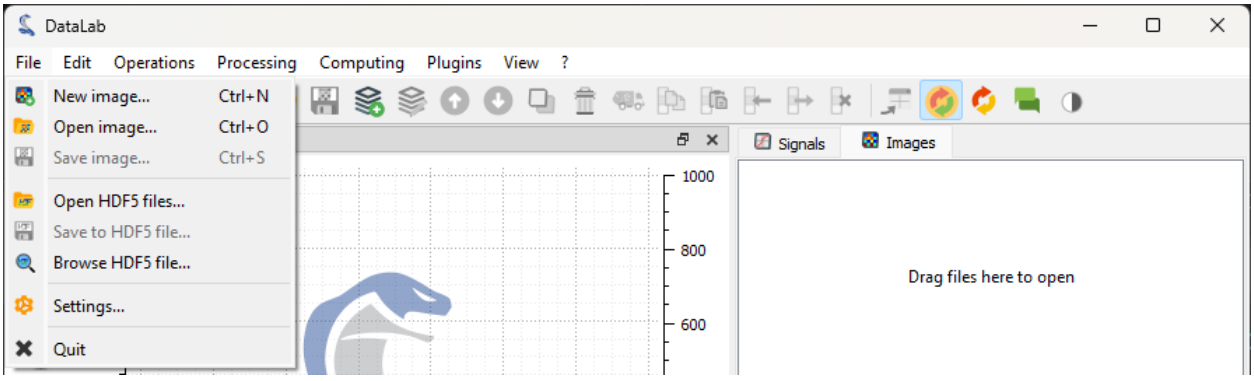


FIG. 39 – Ouvrir les fichiers d'image avec « Fichier > Ouvrir… », ou avec le bouton dans la barre d'outils, ou en faisant glisser-déposer les fichiers dans DataLab (sur le panneau de droite).

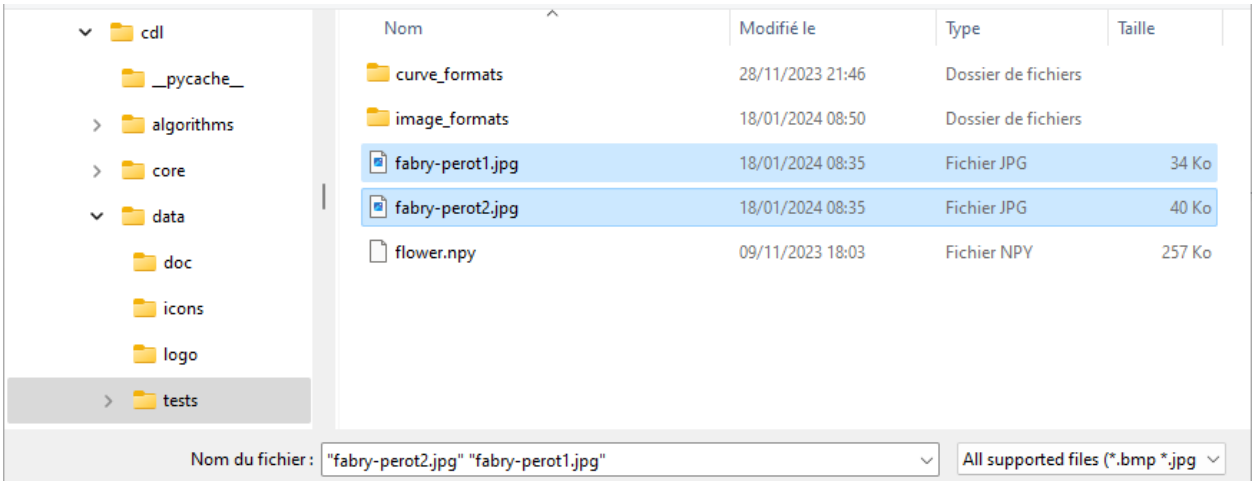


FIG. 40 – Sélectionnez les images de test « fabry_perot1.jpg » et « fabry_perot2.jpg » et cliquez sur « Ouvrir ».

sur le bouton du milieu de la souris et en faisant glisser la souris.

Note : Lorsque vous travaillez sur des images spécifiques à une application (par exemple des images de radiographie X, ou des images de microscopie optique), il est souvent utile de changer la colormap en une colormap en niveaux de gris. Si vous voyez une colormap d'image différente de celle affichée dans l'image, vous pouvez la modifier en sélectionnant l'image dans le panneau de visualisation, puis en sélectionnant la colormap dans la barre d'outils verticale à gauche du panneau de visualisation.

Ou, encore mieux, vous pouvez modifier la colormap par défaut dans les paramètres de DataLab en sélectionnant « Edition > Paramètres… » dans le menu, ou le bouton dans la barre d'outils.

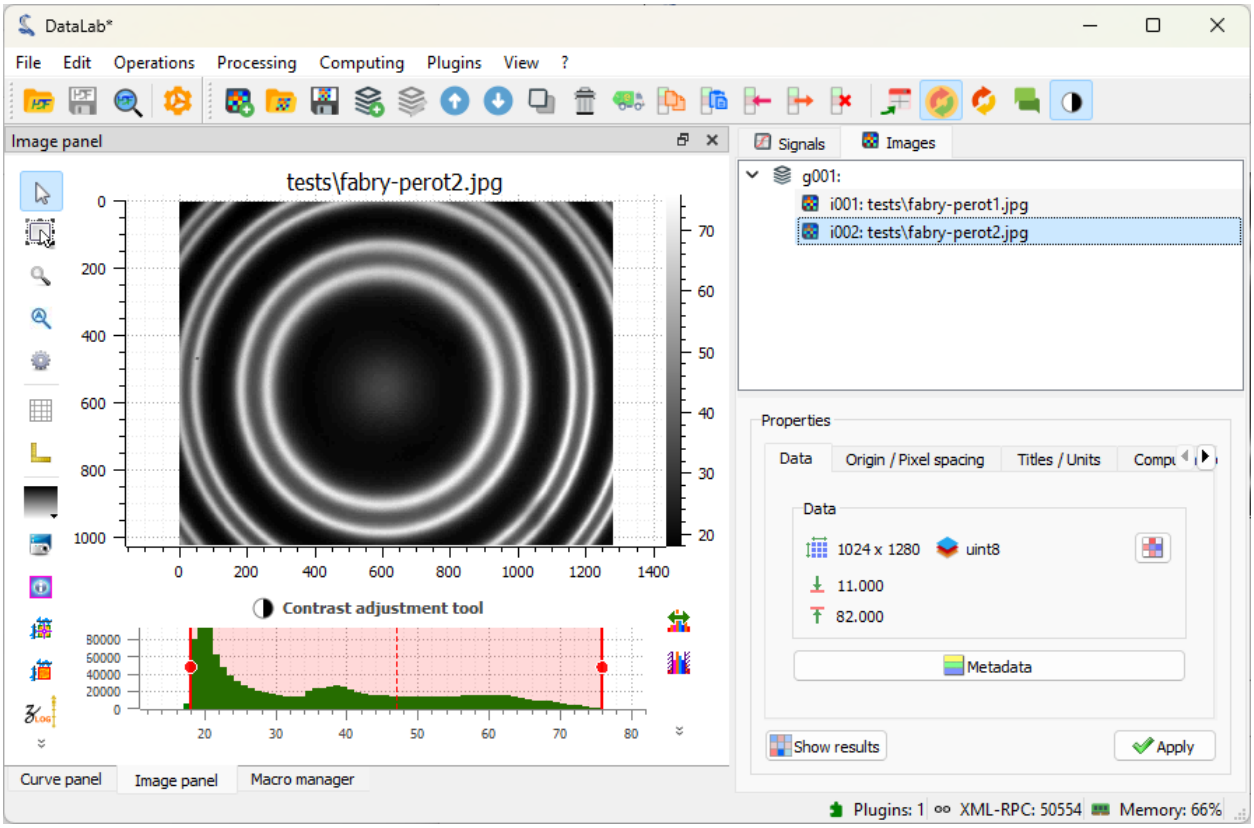


FIG. 41 – Zoomer avec le bouton droit de la souris. Déplacer l'image avec le bouton du milieu de la souris.

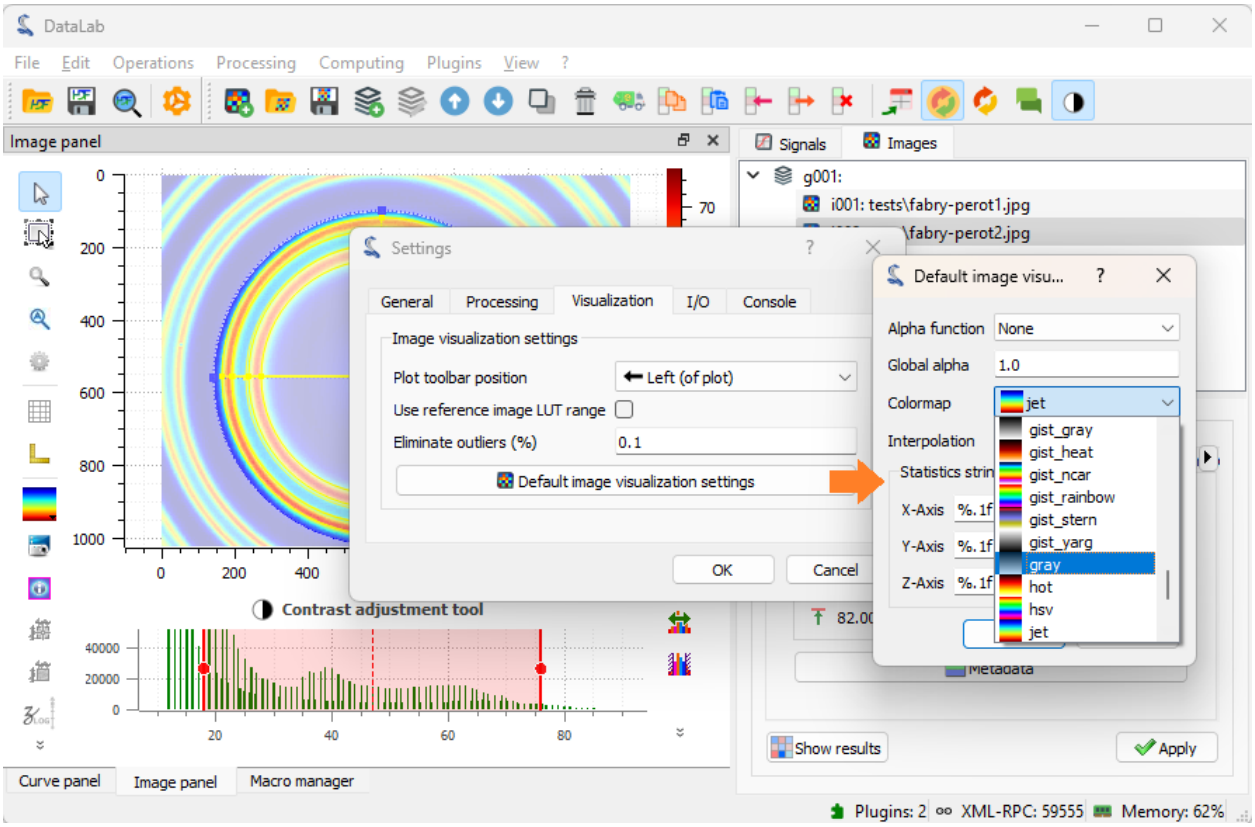


FIG. 42 – Sélectionnez l'onglet « Visualisation », et sélectionnez la colormap « gray ».

Ensuite, définissons une région d'intérêt circulaire (ROI) autour de la frange centrale.

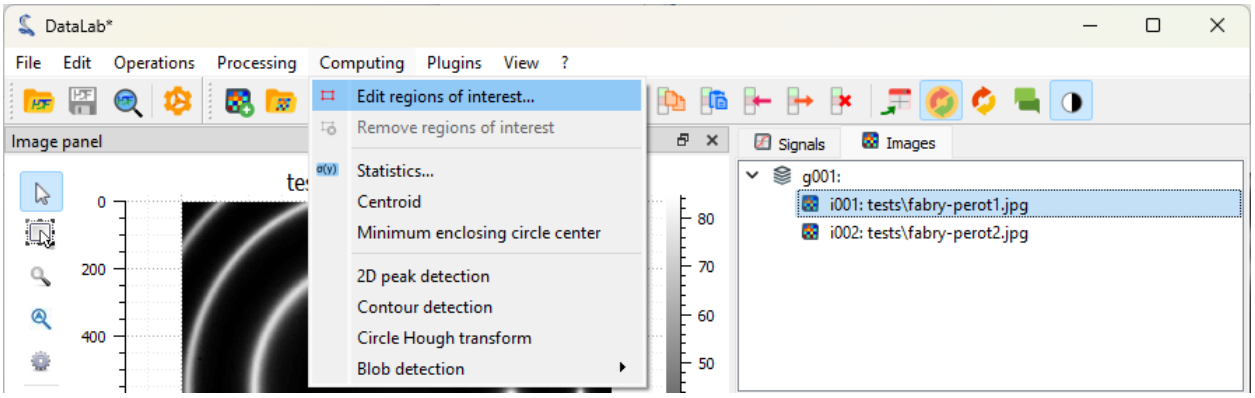


FIG. 43 – Sélectionnez l'outil « Modifier les régions d'intérêt » dans le menu « Calcul ».

A présent, détectons les contours dans la ROI et ajustons-les à des cercles.

Note : Si vous souhaitez afficher à nouveau les résultats de calcul, vous pouvez sélectionner l'entrée « Afficher les résultats » dans le menu « Calcul », ou le bouton « Afficher les résultats » , en dessous de la liste des images :

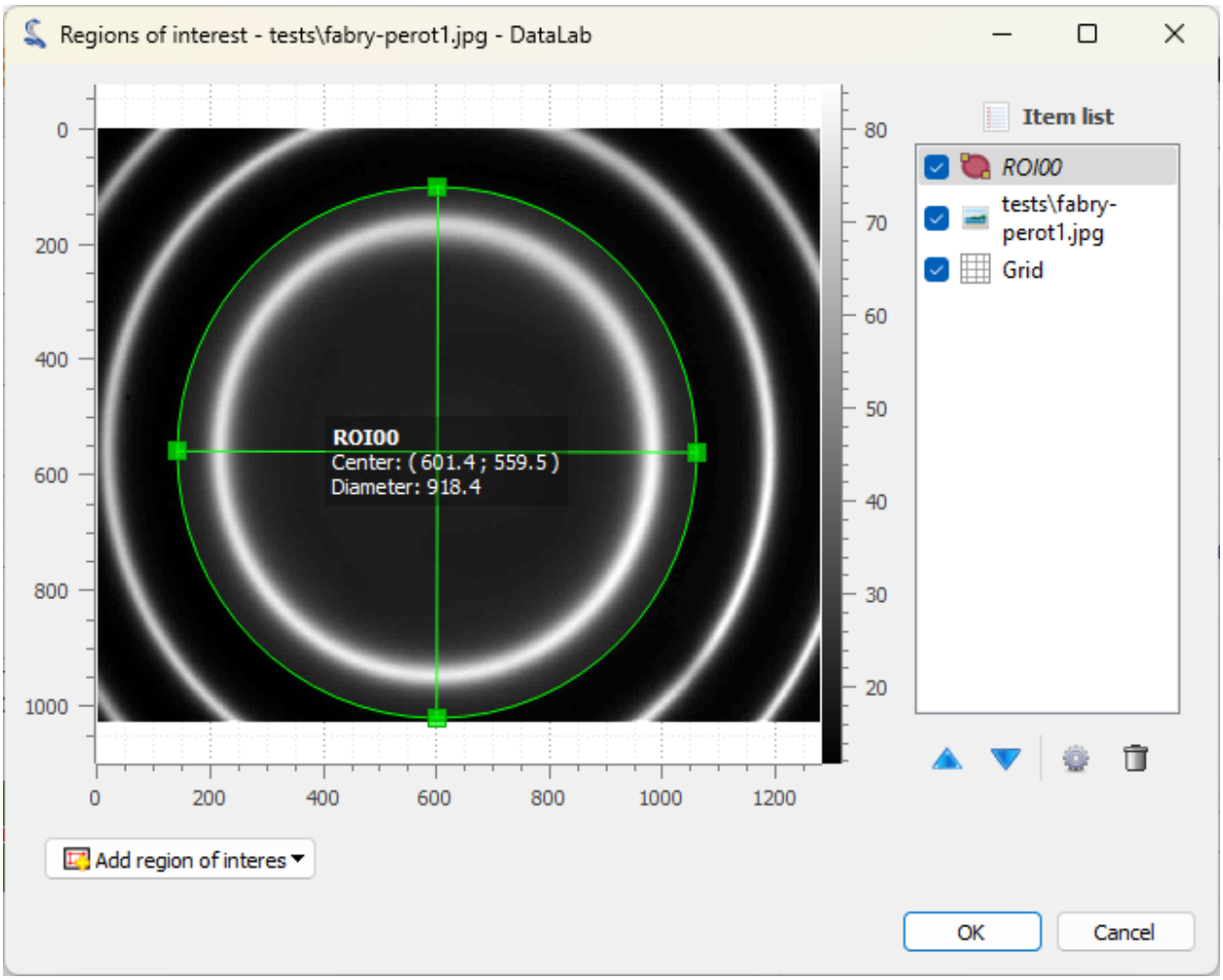


FIG. 44 – La boîte de dialogue « Régions d'intérêt » s'ouvre. Cliquez sur « Ajouter une région d'intérêt » et sélectionnez une ROI circulaire. Redimensionnez la ROI prédéfinie en faisant glisser les poignées. Notez que vous pouvez modifier le rayon de la ROI tout en gardant son centre fixe en appuyant sur la touche « Ctrl ». Cliquez sur « OK » pour fermer la boîte de dialogue.

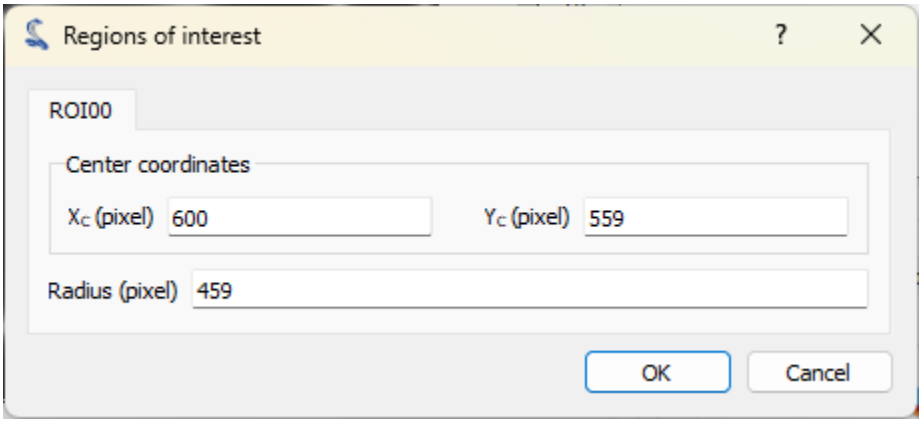


FIG. 45 – Une autre boîte de dialogue s'ouvre et vous demande de confirmer les paramètres de la ROI. Cliquez sur « OK ».

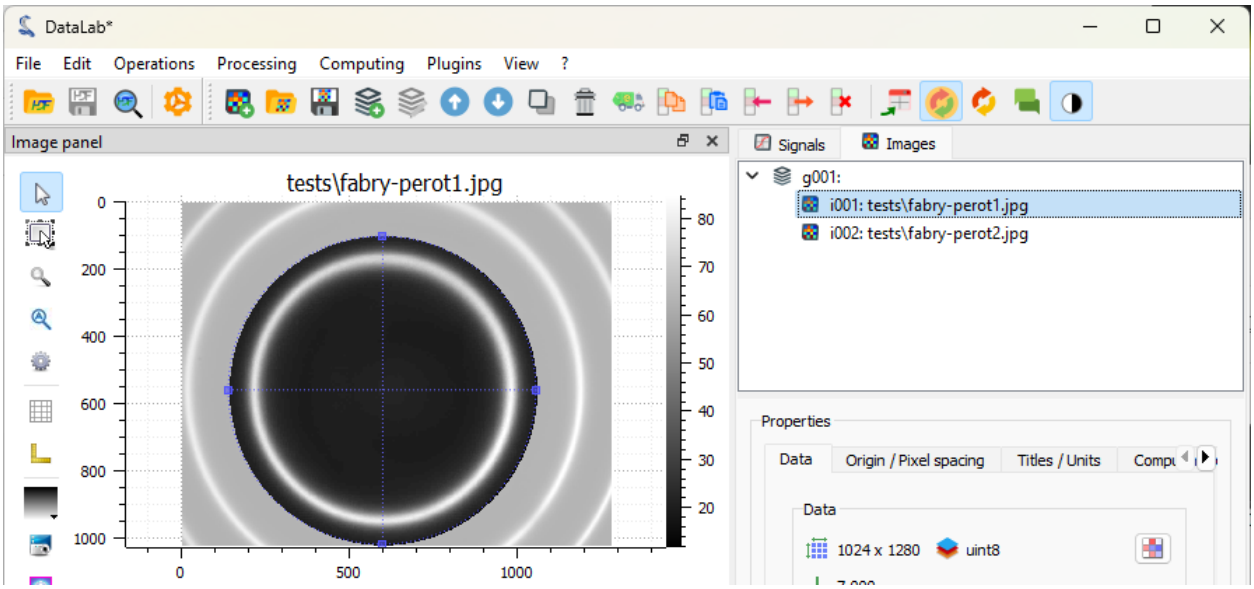


FIG. 46 – La ROI est affichée sur l'image : les pixels masqués sont grisés, et la limite de la ROI est affichée en bleu (notez que, en interne, la ROI est définie par un masque binaire, c'est-à-dire que les données d'image sont représentées sous la forme d'un tableau masqué NumPy).

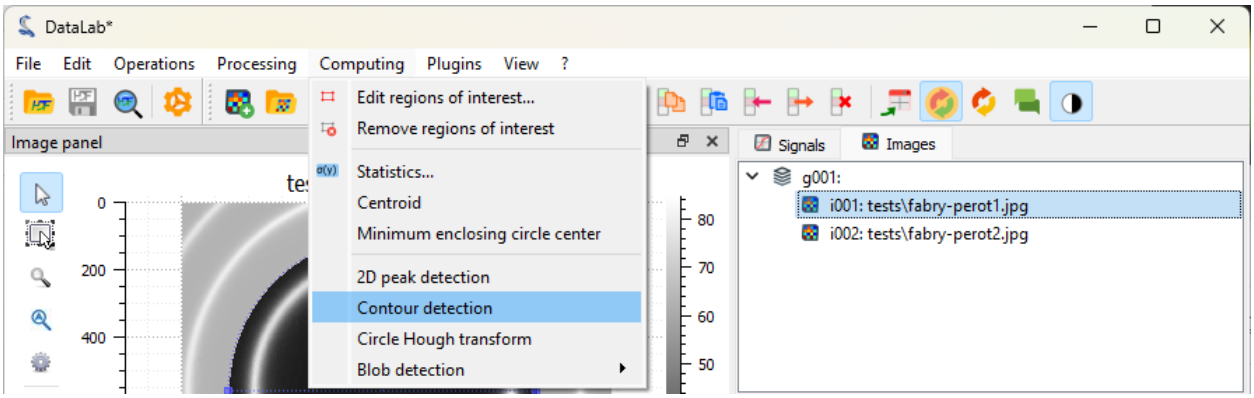


FIG. 47 – Sélectionnez l'outil « Détection de contour » dans le menu « Calcul ».

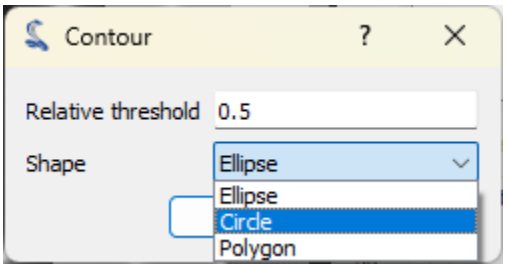


FIG. 48 – La boîte de dialogue « Contour » s'ouvre. Sélectionnez la forme « Cercle » et cliquez sur « OK ».

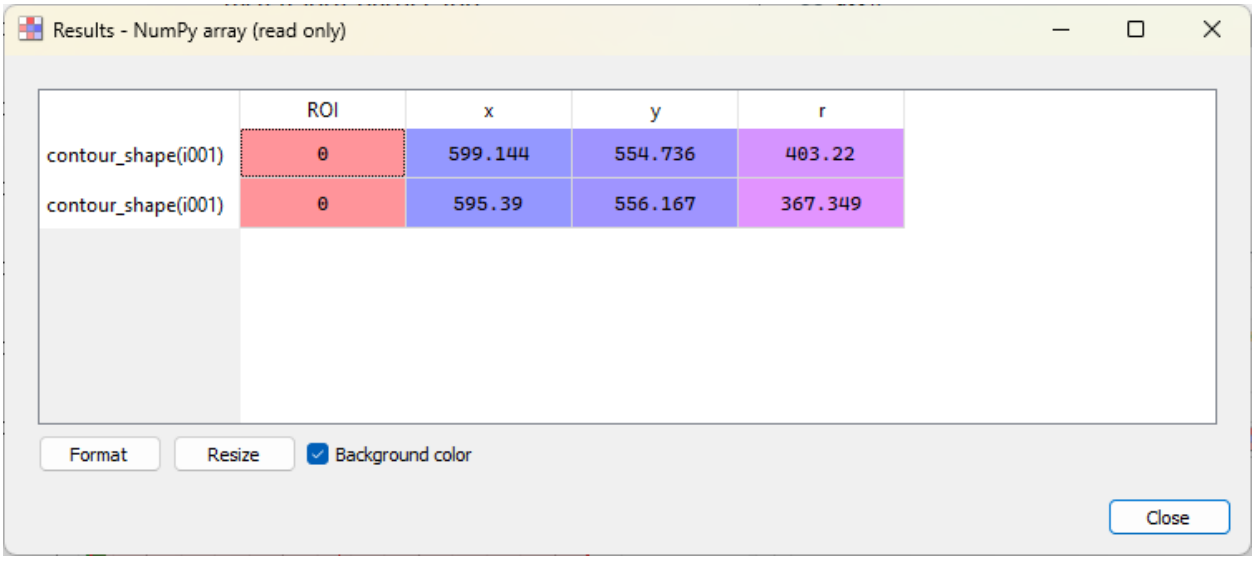


FIG. 49 – La boîte de dialogue « Résultats » s'ouvre et affiche les paramètres du cercle ajusté. Cliquez sur « OK ».

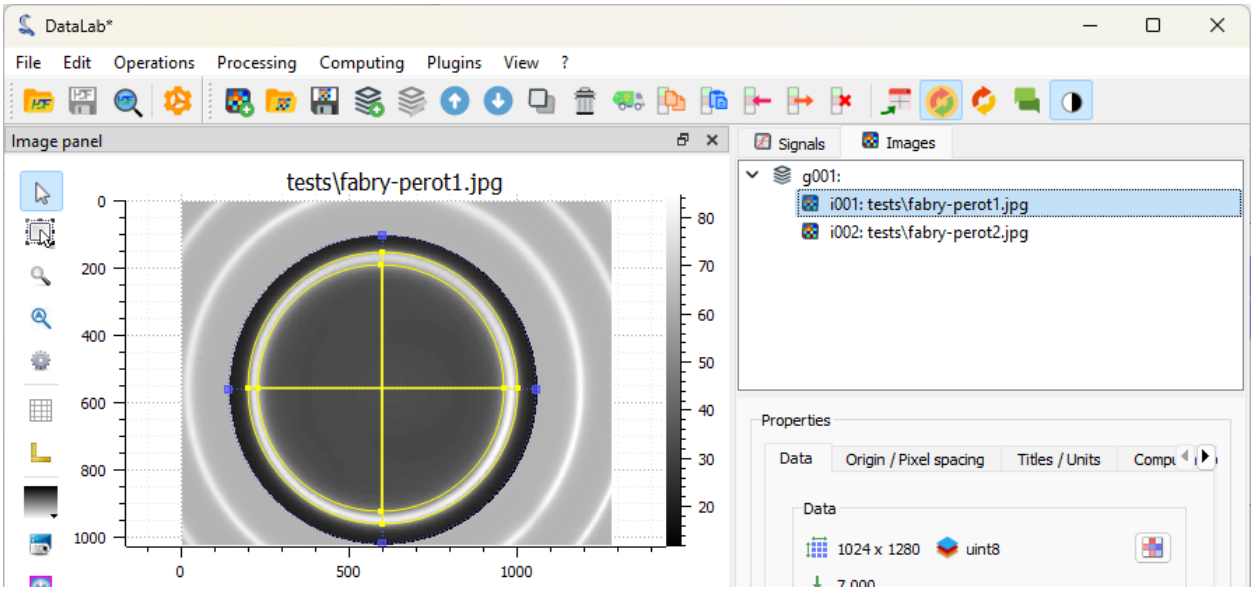
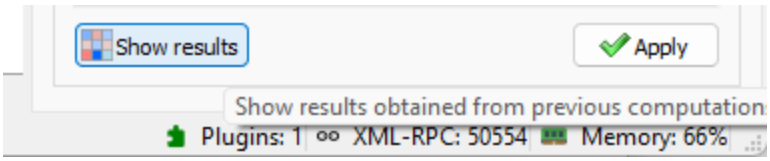


FIG. 50 – Les cercles ajustés sont affichés sur l'image.



Les images (ou signaux) peuvent également être affichés dans une fenêtre séparée, en cliquant sur l'entrée « Afficher dans une nouvelle fenêtre » dans le menu « Affichage » (ou le bouton dans la barre d'outils). Ceci est utile pour comparer côte à côte des images ou des signaux.

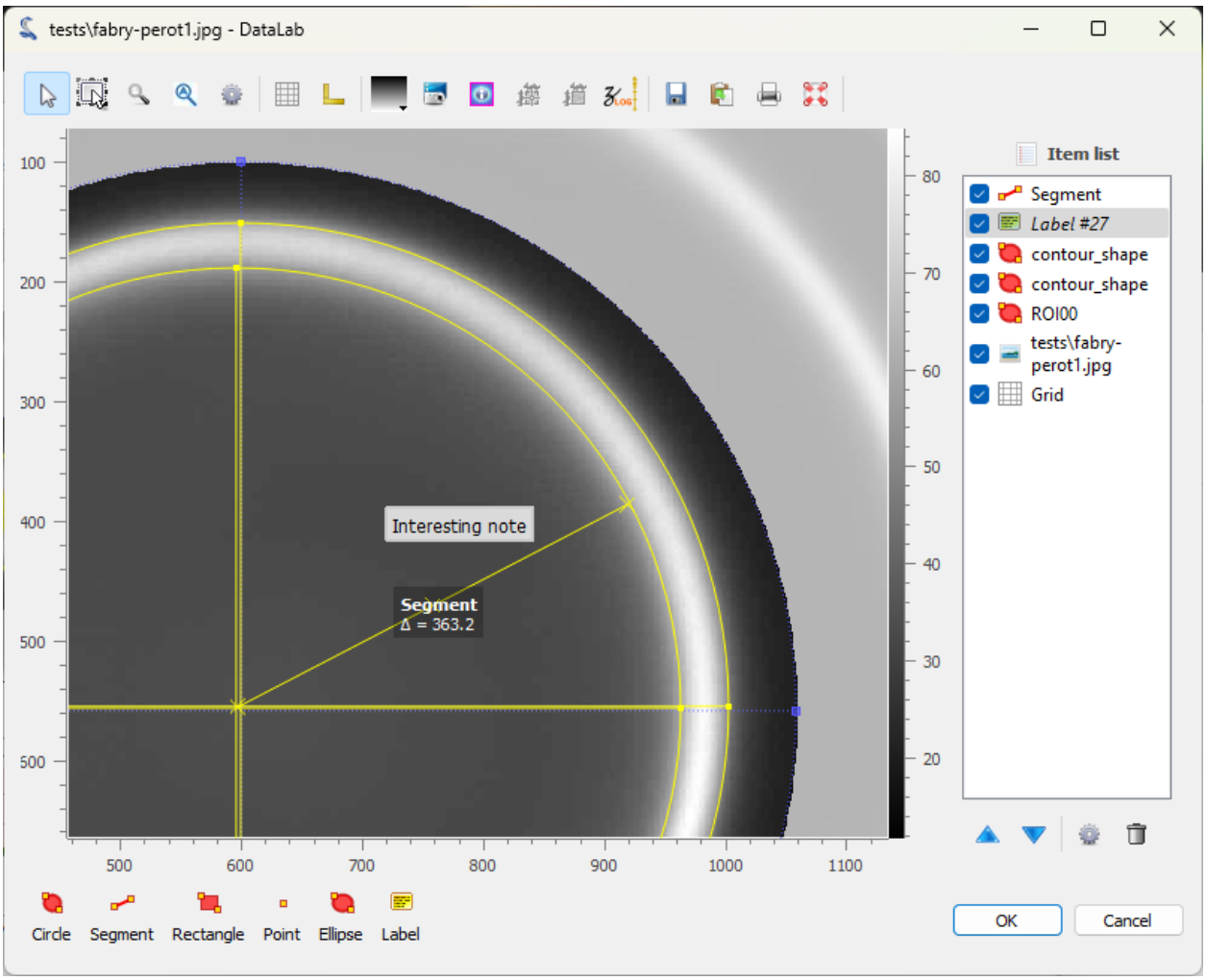


FIG. 51 – L'image est affichée dans une fenêtre séparée. La ROI et les cercles ajustés sont également affichés. Des annotations peuvent être ajoutées à l'image en cliquant sur les boutons en bas de la fenêtre. Les annotations sont stockées dans les métadonnées de l'image, et avec les données de l'image lorsque l'espace de travail est sauvegardé. Cliquez sur « OK » pour fermer la fenêtre.

Si vous souhaitez examiner de plus près les métadonnées, vous pouvez ouvrir la boîte de dialogue « Métadonnées ».

Maintenant, supprimons les métadonnées de l'image (y compris les annotations) pour nettoyer l'image.

Si nous voulons définir la même ROI sur la deuxième image, nous pouvons copier/coller la ROI de la première image vers la deuxième image, en utilisant les métadonnées.

Pour extraire le profil d'intensité le long de l'axe X, nous avons deux options :

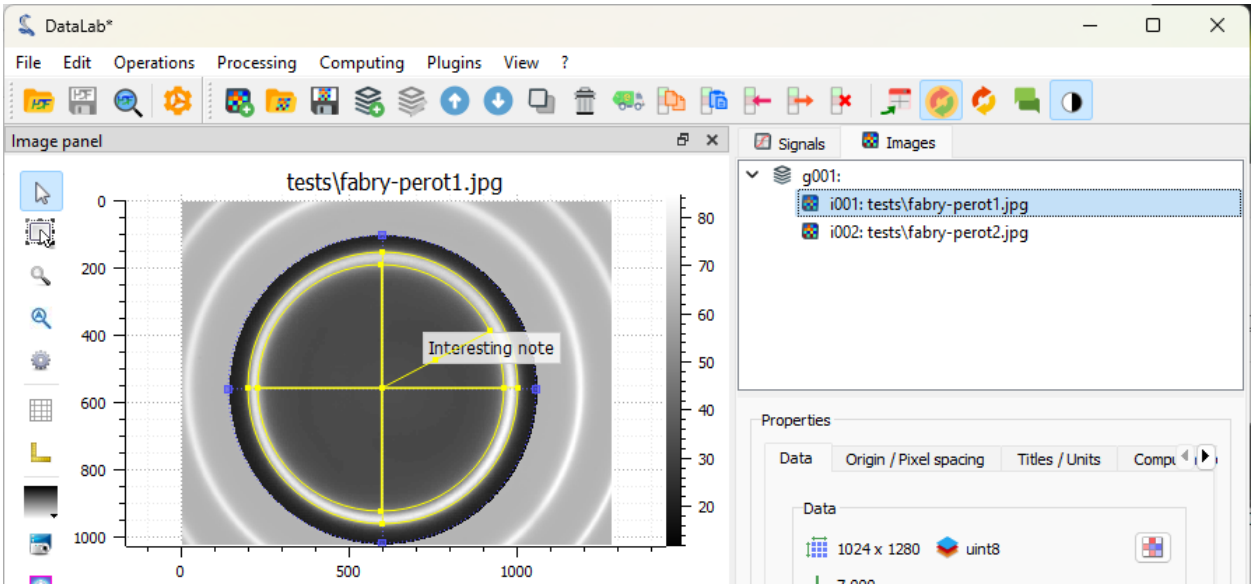


FIG. 52 – L'image est affichée dans la fenêtre principale, avec les annotations.

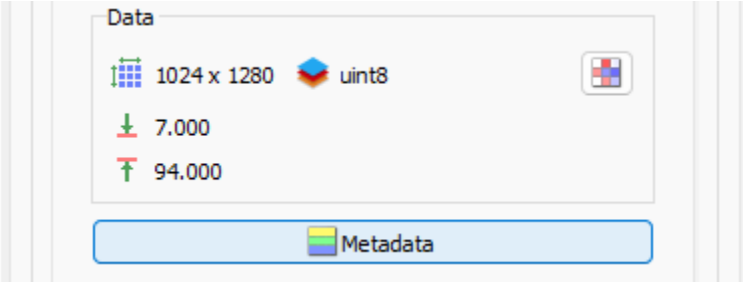


FIG. 53 – Le bouton « Métadonnées » est situé en dessous de la liste des images.

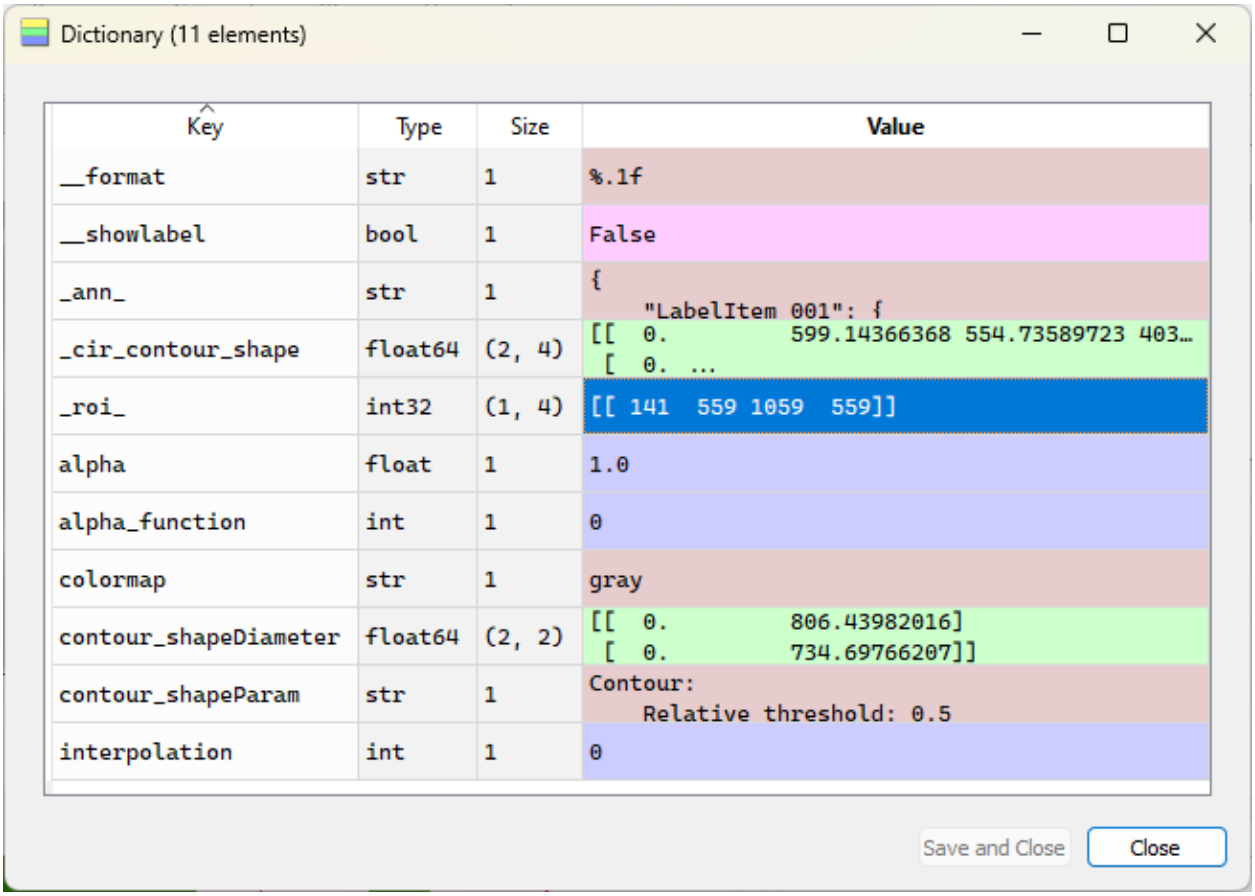


FIG. 54 – La boîte de dialogue « Métadonnées » s'ouvre. Parmi d'autres informations, elle affiche les annotations (dans un format JSON), des informations de style (par exemple la colormap), et la ROI.

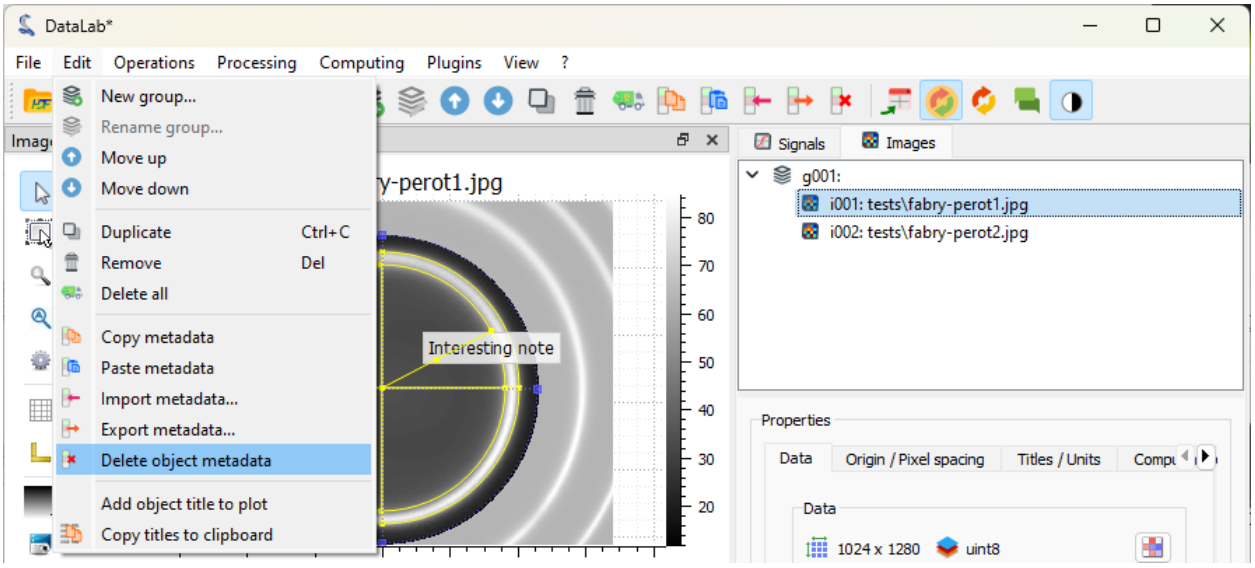


FIG. 55 – Sélectionnez l'entrée « Supprimer les métadonnées » dans le menu « Edition », ou le bouton dans la barre d'outils.

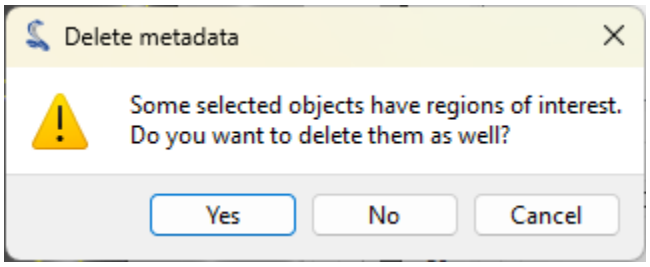


FIG. 56 – La boîte de dialogue « Supprimer les métadonnées » s'ouvre. Cliquez sur « Non » pour conserver la ROI et supprimer le reste des métadonnées.

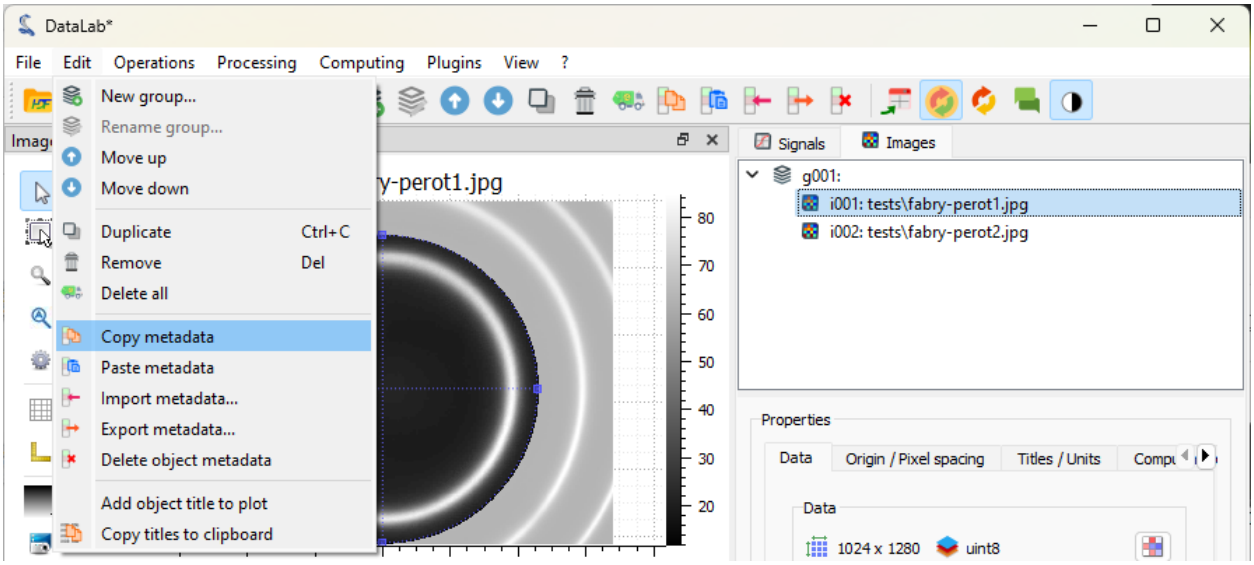


FIG. 57 – Sélectionnez l'entrée « Copier les métadonnées » dans le menu « Edition », ou le bouton dans la barre d'outils.

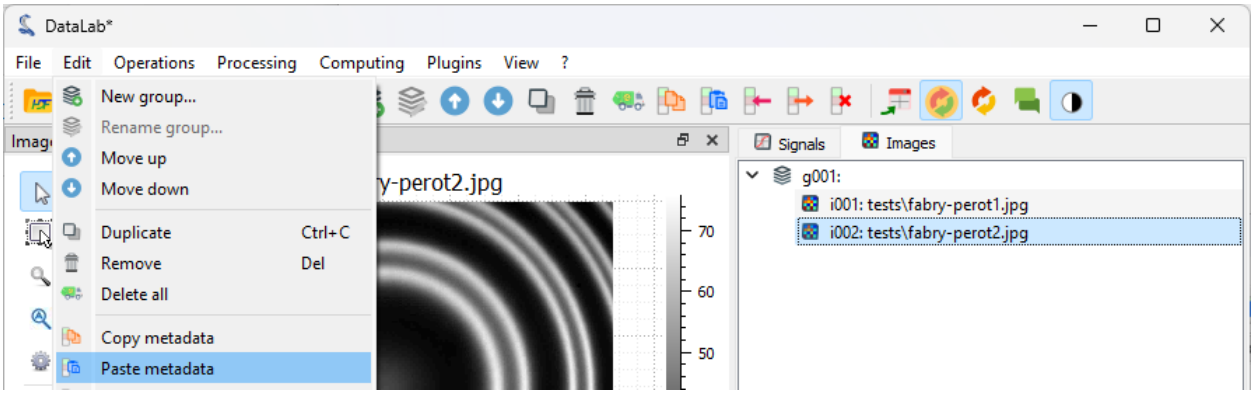


FIG. 58 – Sélectionnez la deuxième image dans le panneau « Images », puis sélectionnez l'entrée « Coller les métadonnées » dans le menu « Edition », ou le bouton dans la barre d'outils.

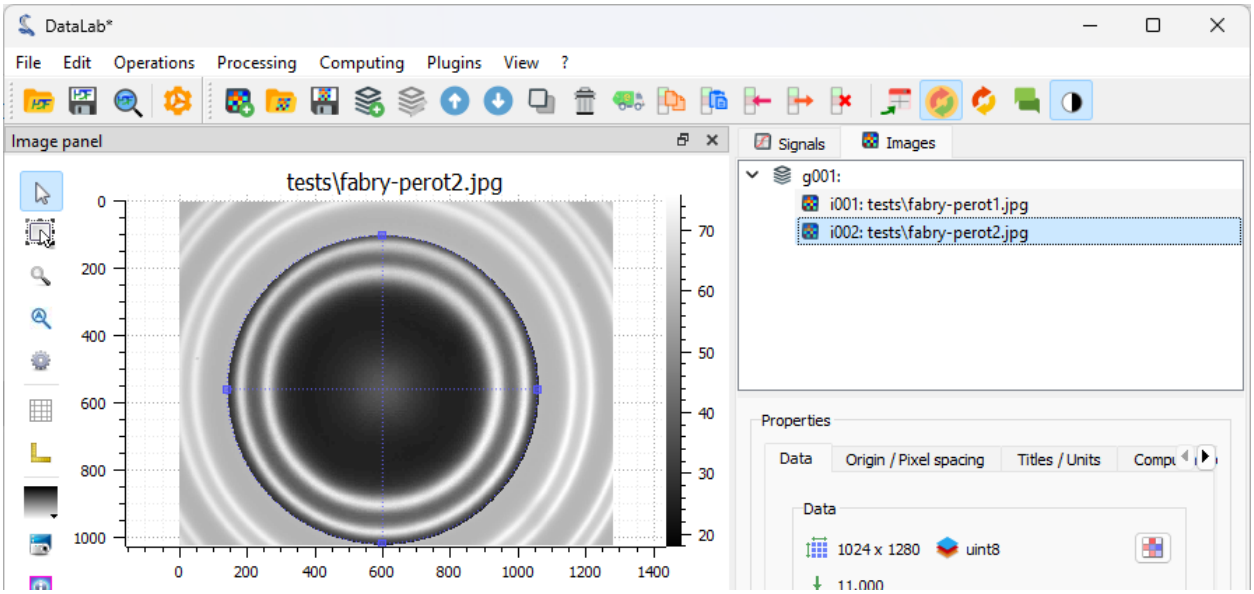


FIG. 59 – La ROI est ajoutée à la deuxième image.

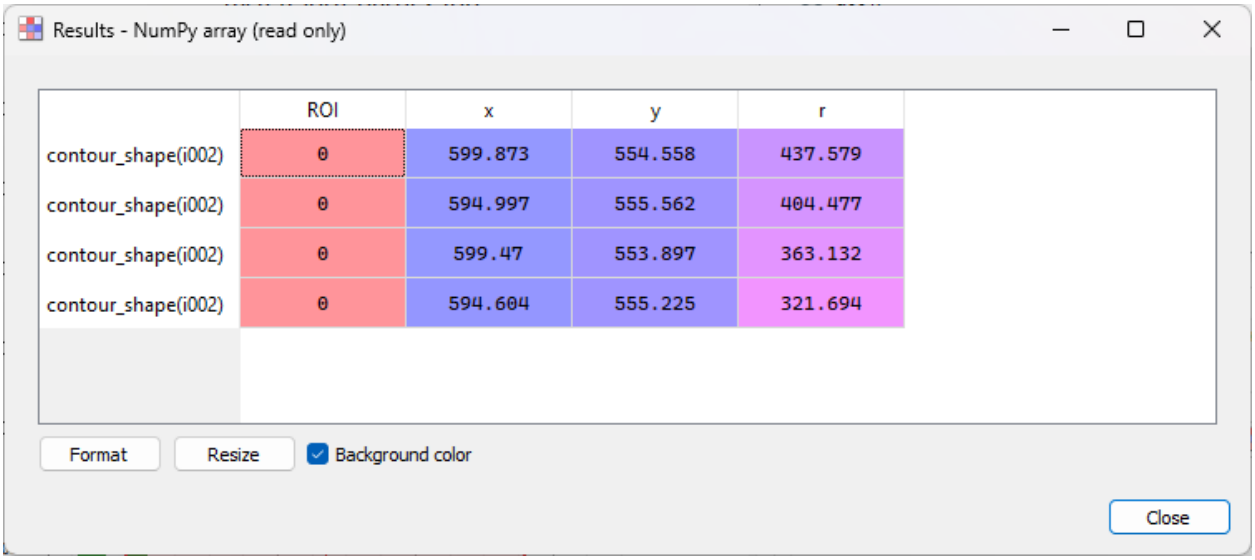


FIG. 60 – Sélectionnez l'outil « Détection de contour » dans le menu « Calcul », avec les mêmes paramètres qu'auparavant (forme « Cercle »). Sur cette image, il y a deux franges, donc quatre cercles sont ajustés. La boîte de dialogue « Résultats » s'ouvre et affiche les paramètres du cercle ajusté. Cliquez sur « OK ».

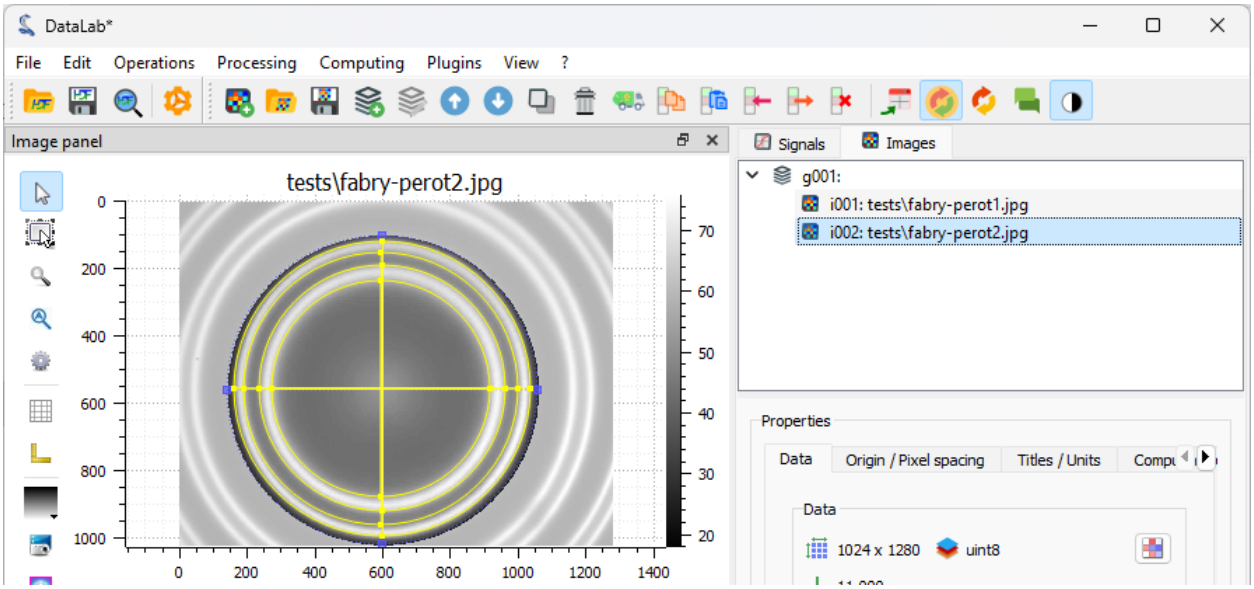


FIG. 61 – Les cercles ajustés sont affichés sur l'image.

— Soit sélectionner l'entrée « Profil rectiligne… » dans le menu « Opérations > Profils d'intensité ».

— Soit activer l'outil « Profil rectiligne » dans la barre d'outils verticale à gauche du panneau de visualisation.

Essayons la première option, en sélectionnant l'entrée « Extraire le profil… » : c'est la façon la plus simple d'extraire un profil d'une image, et cela correspond à la méthode `compute_profile` de l'API de DataLab (donc elle peut être utilisée dans un script, un plugin ou une macro).

Si vous souhaitez effectuer des mesures sur le profil, ou ajouter des annotations, vous pouvez ouvrir le signal dans une fenêtre séparée, en cliquant sur l'entrée « Afficher dans une nouvelle fenêtre » dans le menu « Affichage » (ou le bouton dans la barre d'outils).

Maintenant, essayons la deuxième option pour extraire le profil d'intensité le long de l'axe X, en activant l'outil « Profil rectiligne » dans la barre d'outils verticale à gauche du panneau de visualisation (cet outil est une fonctionnalité de PlotPy). Avant de pouvoir l'utiliser, nous devons sélectionner l'image dans le panneau de visualisation (sinon l'outil est grisée). Ensuite, nous pouvons cliquer sur l'image pour afficher le profil d'intensité le long de l'axe X. DataLab intègre une version modifiée de cet outil, qui permet de transférer le profil vers le panneau « Signaux » pour un traitement ultérieur.

Ensuite, cliquez sur le bouton « Traiter le signal » dans la barre d'outils près du profil pour transférer le profil vers le panneau « Signaux ».

Enfin, nous pouvons sauvegarder l'espace de travail dans un fichier. L'espace de travail contient toutes les images et signaux qui ont été chargés ou traités dans DataLab. Il contient également les résultats de calcul, les paramètres de visualisation (colormaps, contraste, etc.), les métadonnées et les annotations.

Si vous souhaitez charger à nouveau l'espace de travail, vous pouvez utiliser « Fichier > Ouvrir un fichier HDF5… » (ou le bouton dans la barre d'outils) pour charger l'ensemble de l'espace de travail, ou « Fichier > Parcourir un fichier HDF5… » (ou le bouton dans la barre d'outils) pour charger uniquement une sélection d'ensembles de données de l'espace de travail.

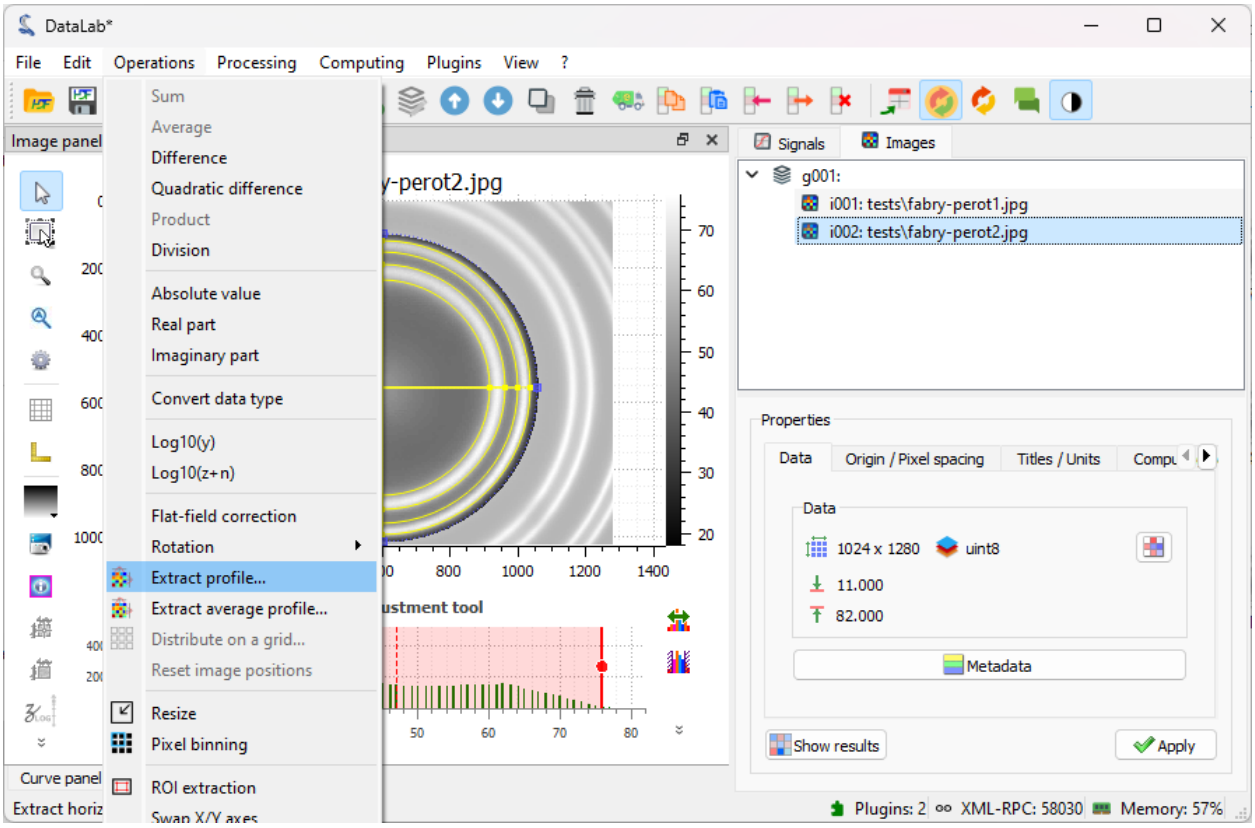


FIG. 62 – Sélectionnez l'entrée « Profil rectiligne… » dans le menu « Opérations ».

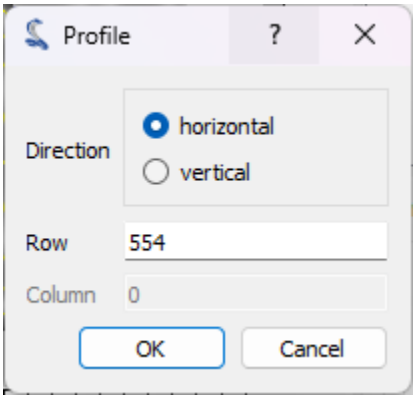


FIG. 63 – La boîte de dialogue « Profil » s'ouvre. Entrez la ligne du profil horizontal (ou la colonne du profil vertical) dans la boîte de dialogue qui s'ouvre. Cliquez sur « OK ».

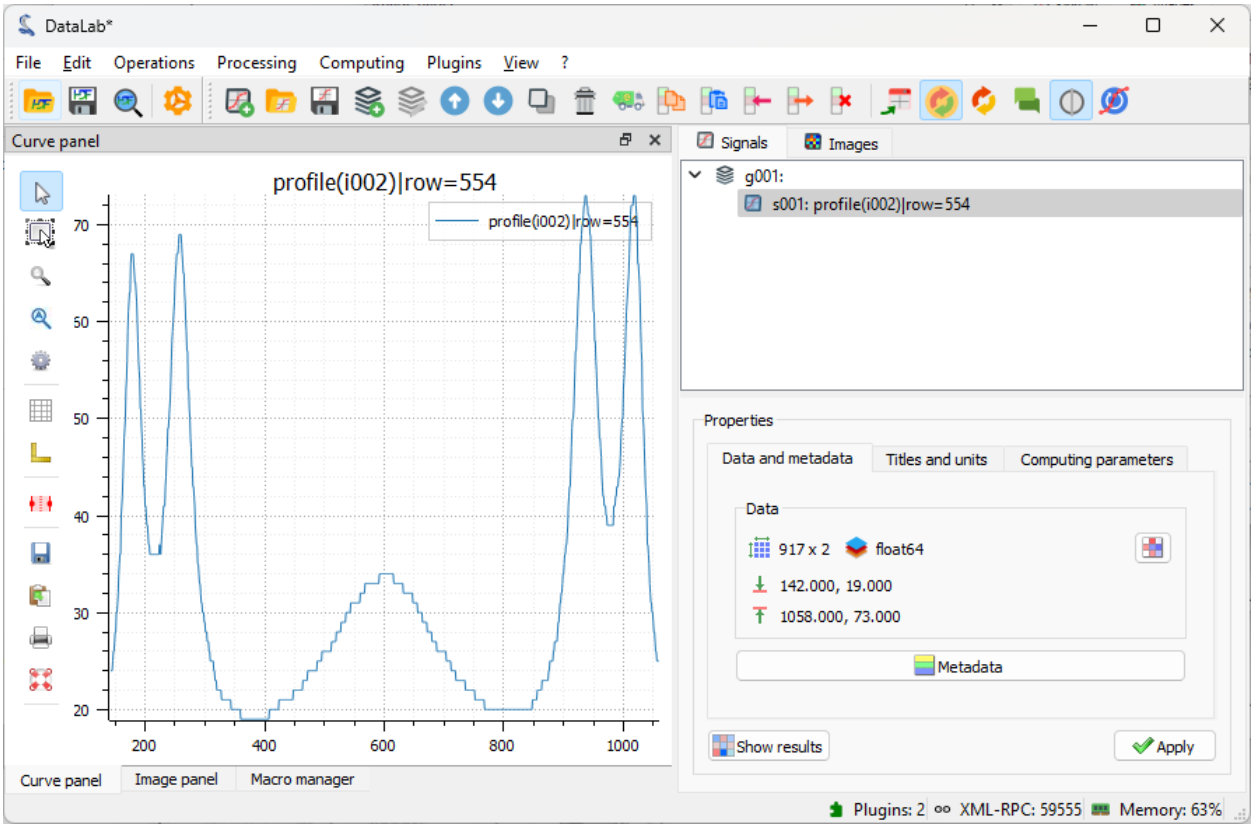


FIG. 64 – Le profil d'intensité est ajouté au panneau « Signaux », et DataLab bascule vers ce panneau pour afficher le profil.

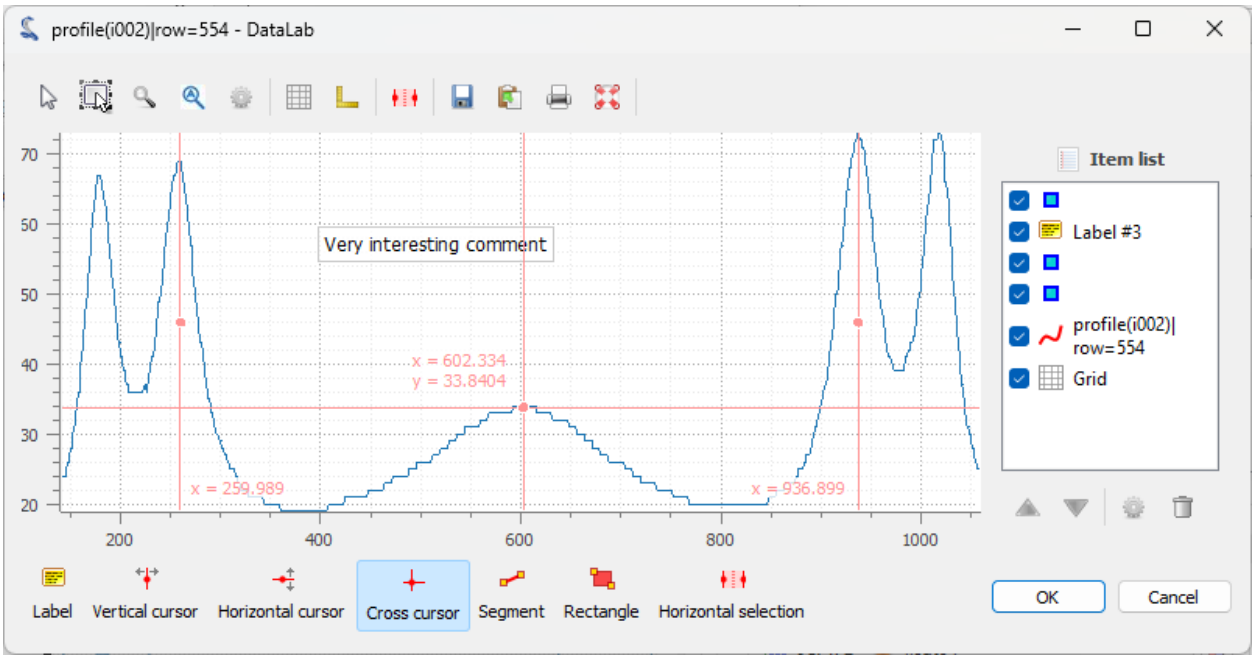


FIG. 65 – Le signal est affiché dans une fenêtre séparée. Ici, nous avons ajouté des curseurs verticaux et une étiquette de texte très intéressante. Comme pour les images, les annotations sont stockées dans les métadonnées du signal, et avec les données du signal lorsque l'espace de travail est sauvegardé. Cliquez sur « OK » pour fermer la fenêtre.

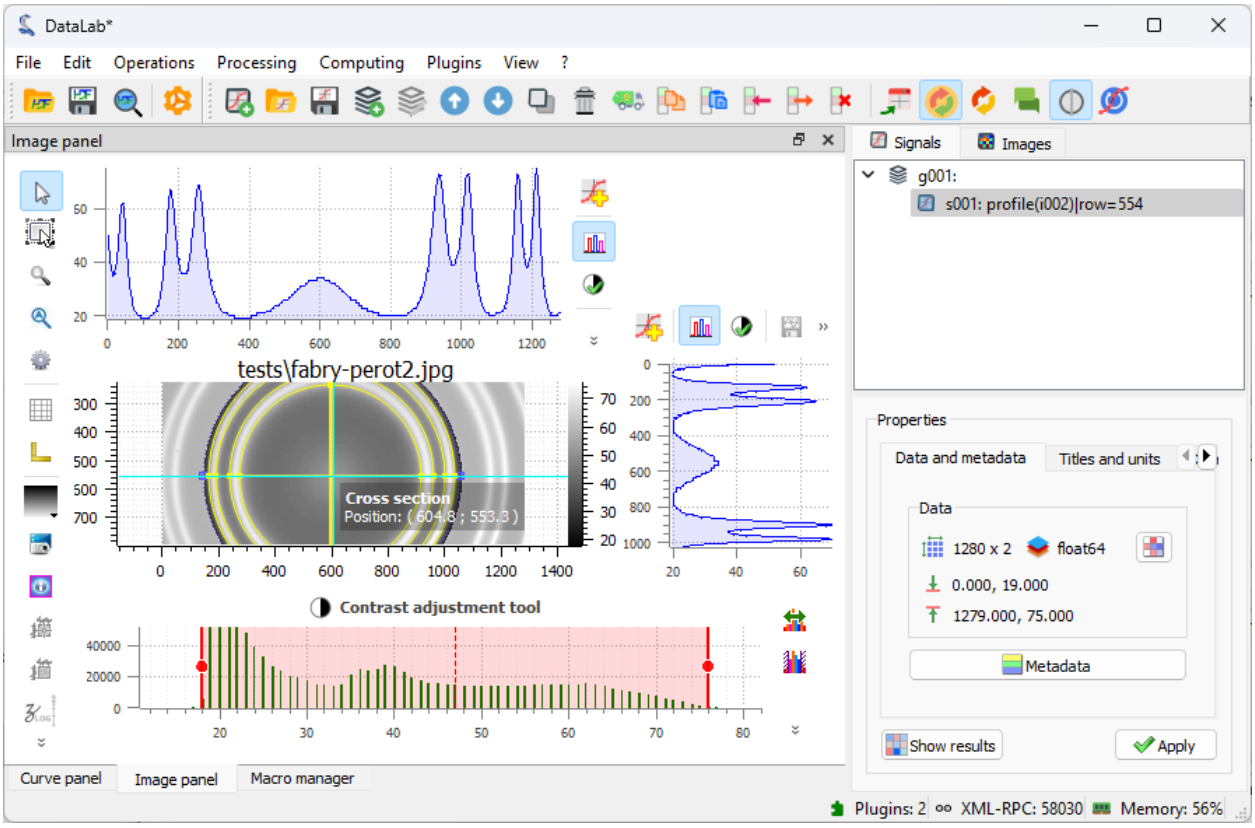


FIG. 66 – Revenez au panneau « Images » et sélectionnez l'image *dans le panneau de visualisation* (sinon l'outil « Section transversale » est grisée). Sélectionnez l'outil « Section transversale » dans la barre d'outils verticale, et cliquez sur l'image pour afficher les profils d'intensité le long des axes X et Y.

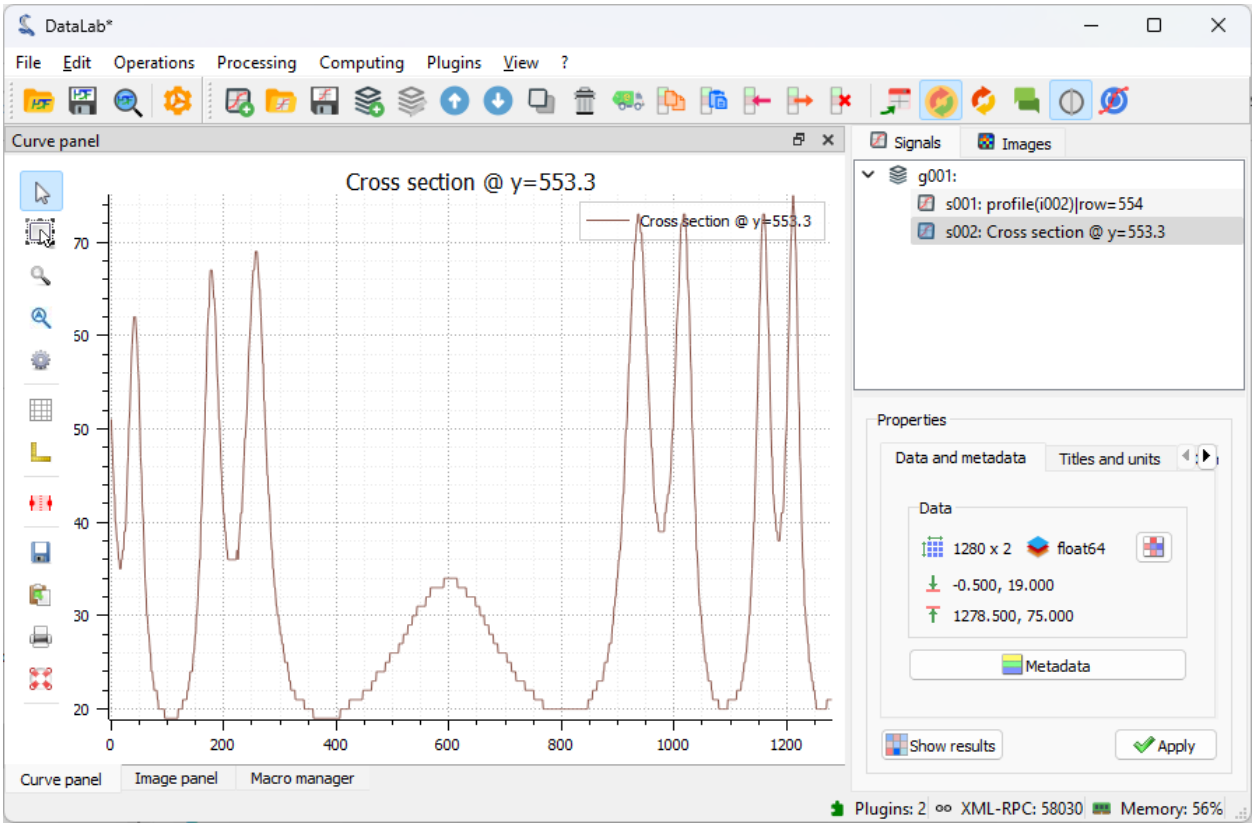


FIG. 67 – Le profil d'intensité est ajouté au panneau « Signaux », et DataLab bascule vers ce panneau pour afficher le profil.

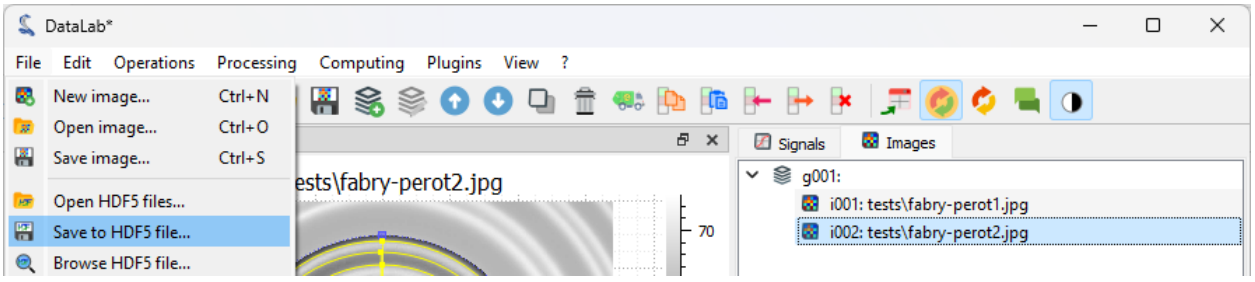


FIG. 68 – Sauvegardez l'espace de travail dans un fichier avec « Fichier > Sauvegarder dans un fichier HDF5… », ou le bouton dans la barre d'outils.

Mesurer la taille d'un faisceau laser

Cet exemple montre comment mesurer la taille d'un faisceau laser le long de l'axe de propagation, en utilisant DataLab :

- Ouvrir toutes les images d'un dossier
- Appliquer un seuillage aux images
- Extraire le profil d'intensité le long d'une ligne horizontale
- Ajuster le profil d'intensité à une fonction gaussienne
- Calculer la largeur à mi-hauteur (FWHM) du profil d'intensité
- Essayer une autre méthode : extraire le profil d'intensité radial
- Calculer la FWHM du profil d'intensité radial
- Effectuer la même analyse sur une pile d'images et sur les profils résultants
- Tracer la taille du faisceau en fonction de la position le long de l'axe de propagation

Tout d'abord, nous ouvrons DataLab et chargeons les images :

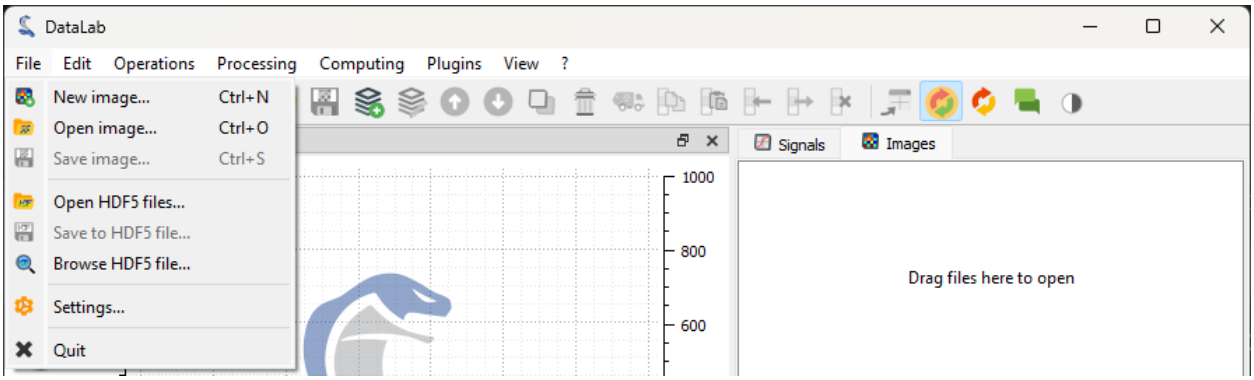


Fig. 69 – Ouvrir les fichiers d'images avec « Fichier > Ouvrir… », ou avec le bouton dans la barre d'outils, ou en faisant glisser et déposer les fichiers dans DataLab (sur le panneau de droite).

Les images sélectionnées sont chargées dans le panneau « Images ». La dernière image est affichée dans la fenêtre principale. Sur chaque image, nous pouvons zoomer en appuyant sur le bouton droit de la souris et en faisant glisser la souris vers le haut et vers le bas. Nous pouvons également déplacer l'image en appuyant sur le bouton du milieu de la souris et en faisant glisser la souris.

Note : Si nous voulons afficher les images côte à côte, nous pouvons sélectionner l'entrée « Distribuer sur une grille » dans le menu « Opérations ».

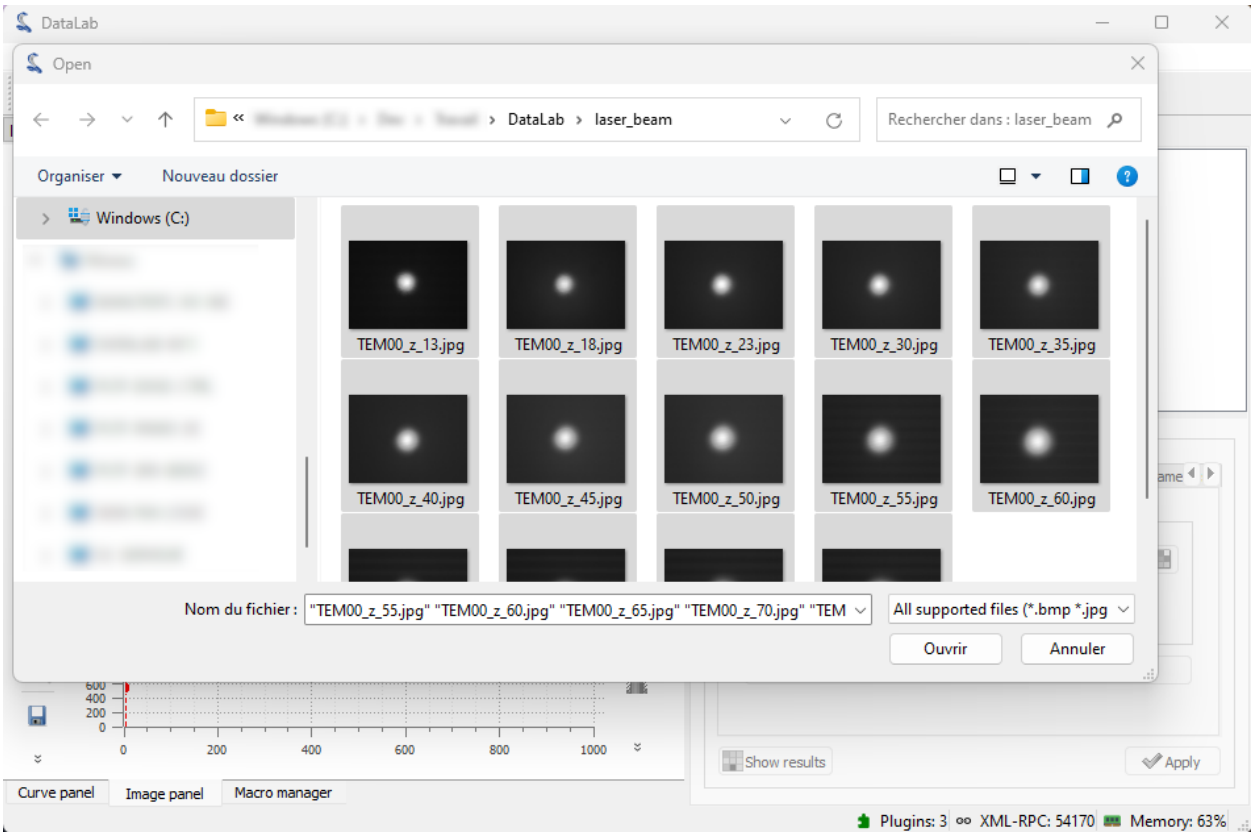


FIG. 70 – Sélectionnez les images de test « TEM00_z_*.jpg » et cliquez sur « Ouvrir ».

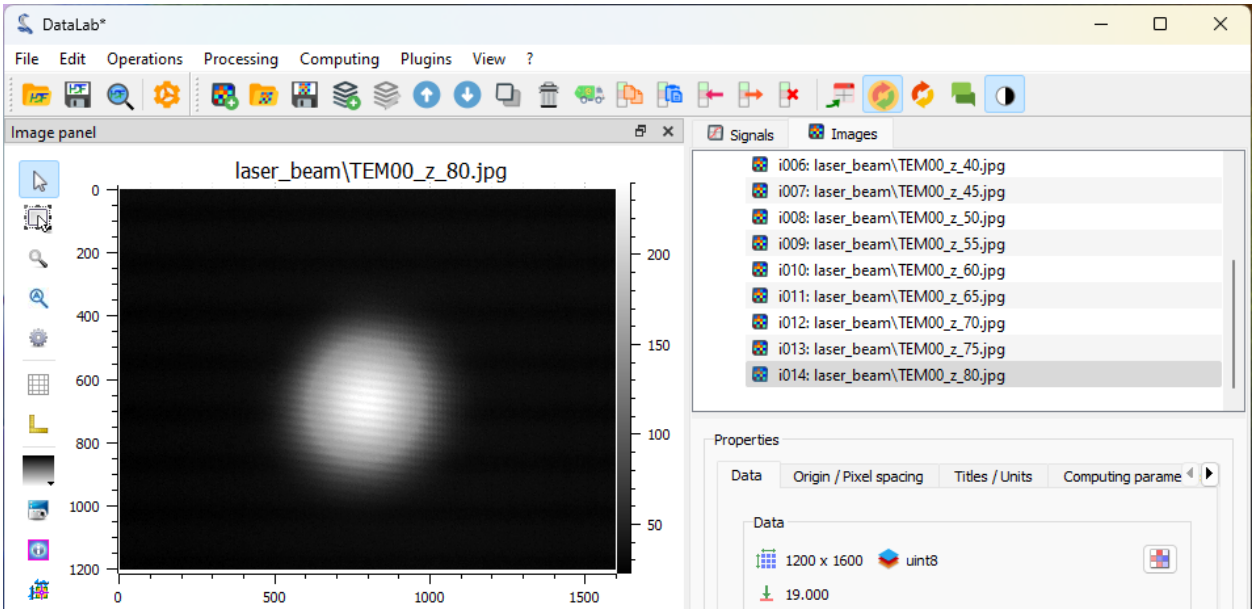


FIG. 71 – Zoomer avec le bouton droit de la souris. Déplacer l'image avec le bouton du milieu de la souris.

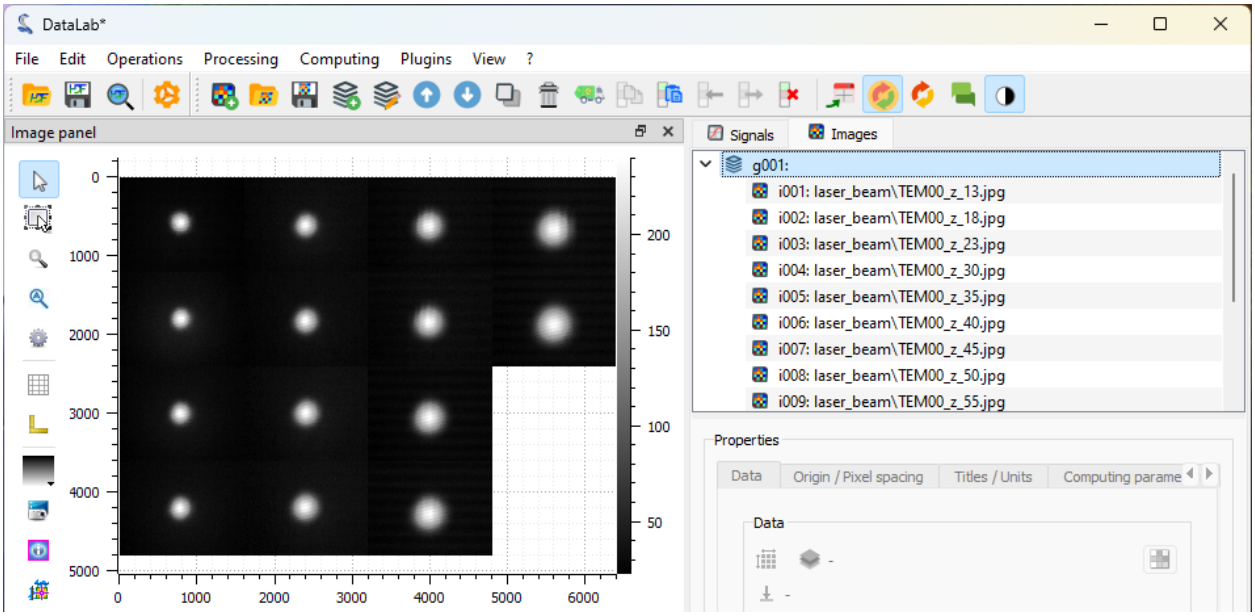


FIG. 72 – Images distribuées sur une grille de 4 lignes

Mais, revenons à l'affichage initial en sélectionnant l'entrée « Réinitialiser les positions des images » dans le menu « Opérations ».

Si nous sélectionnons l'une des images, nous pouvons voir qu'il y a du bruit de fond, il peut donc être utile d'appliquer un seuillage aux images.

Maintenant, essayons une autre méthode pour mesurer la taille du faisceau.

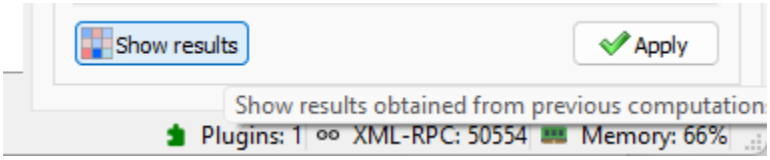
À partir du panneau « Images », nous pouvons extraire le profil d'intensité radial avec « Opérations > Profils d'intensité > Profil radial ».

Toutes ces opérations et calculs que nous avons effectués sur une seule image peuvent être appliqués à toutes les images du panneau « Images ».

Pour ce faire, nous commençons par nettoyer le panneau « Signaux » (avec « Édition > Tout supprimer » ou le bouton dans la barre d'outils). Nous nettoyons également les résultats intermédiaires dans le panneau « Images » en sélectionnant les images obtenues lors de notre prototypage et en les supprimant individuellement (avec « Édition > Supprimer » ou le bouton).

Ensuite, nous sélectionnons toutes les images dans le panneau « Images » (individuellement, ou en sélectionnant l'ensemble du groupe « g001 »).

Note : Si vous souhaitez afficher à nouveau les résultats de calcul, vous pouvez sélectionner l'entrée « Afficher les résultats » dans le menu « Calcul », ou le bouton « Afficher les résultats » , en dessous de la liste des images :



Enfin, nous pouvons enregistrer l'espace de travail dans un fichier . L'espace de travail contient toutes les images et signaux qui ont été chargés ou traités dans DataLab. Il contient également les résultats des calculs, les paramètres de

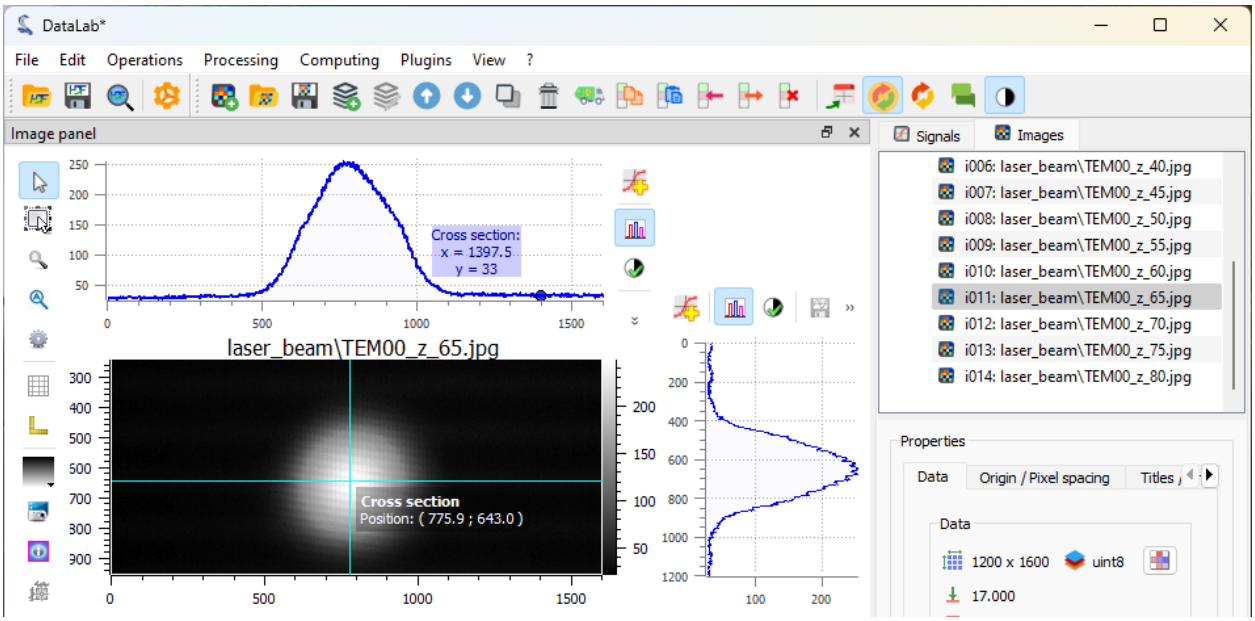


FIG. 73 – Sélectionnez l'une des images dans le panneau « Images », sélectionnez l'image associée dans le panneau de visualisation, et activez l'outil « Section transversale » dans la barre d'outils verticale à gauche du panneau de visualisation (cet outil est une fonctionnalité PlotPy). Sur cette figure, nous pouvons voir que le bruit de fond est d'environ 30 lsb (pour afficher le marqueur de courbe, nous avons dû sélectionner la courbe de profil et cliquer avec le bouton droit de la souris dessus pour afficher le menu contextuel, puis sélectionner « Marqueurs > Lié à l'élément actif »).

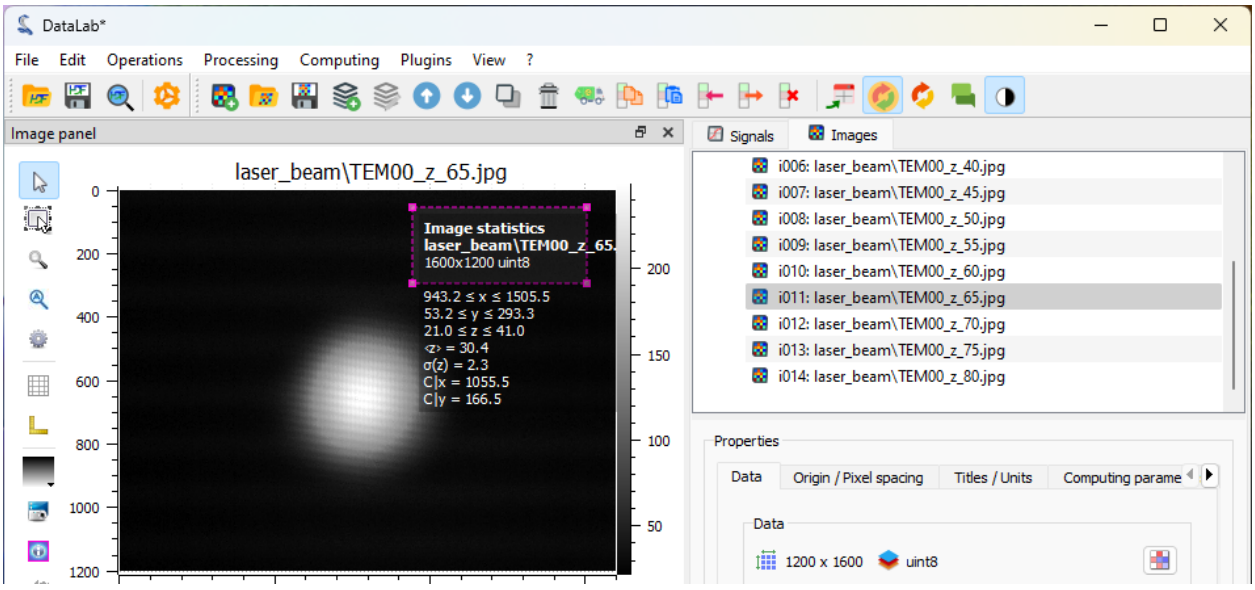


FIG. 74 – Une autre façon de mesurer le bruit de fond est d'utiliser l'outil « Statistiques de l'image » dans la barre d'outils verticale à gauche du panneau de visualisation. Il affiche des statistiques sur une zone rectangulaire définie en faisant glisser la souris sur l'image. Cela confirme que le bruit de fond est d'environ 30 lsb.

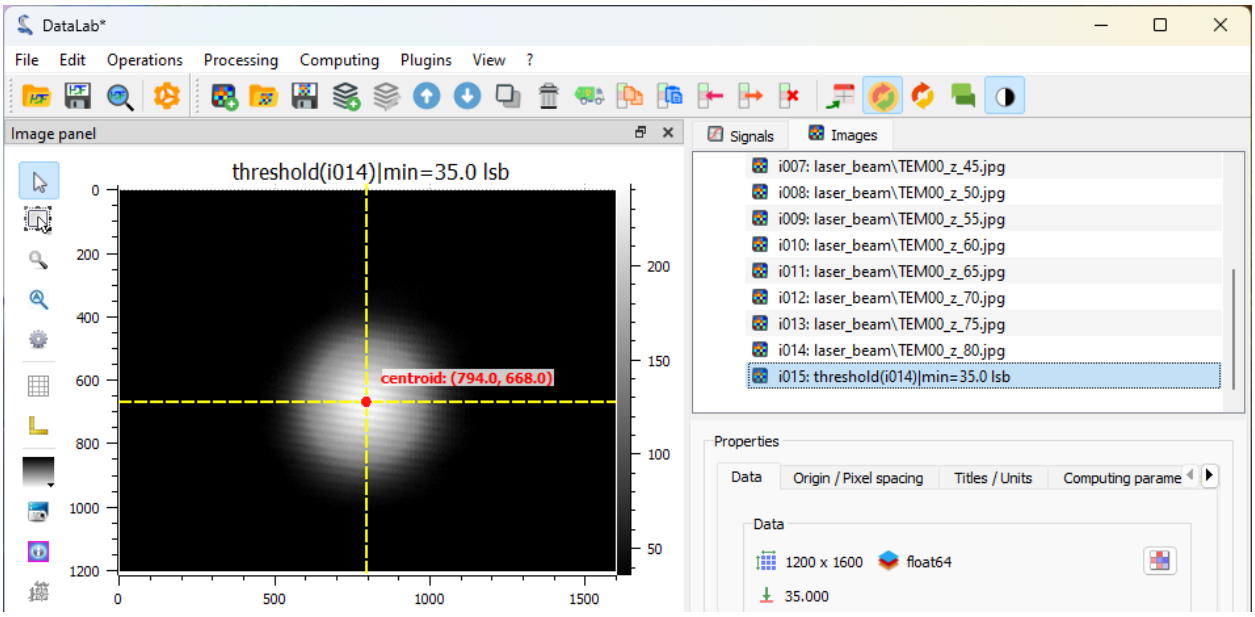


FIG. 75 – Après avoir appliqué un seuil à 35 lsb (avec « Traitement > Seuillage… »), nous pouvons calculer une position plus précise du centre du faisceau en utilisant « Calcul > Centre de gravité ».

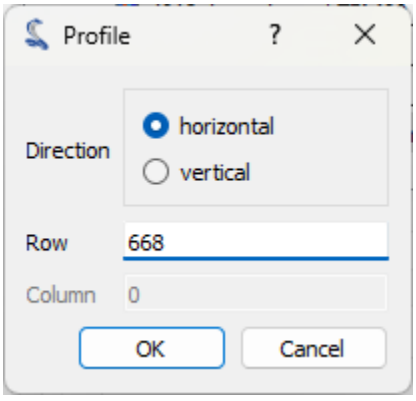


FIG. 76 – Ensuite, nous pouvons extraire un profil de ligne le long de l'axe horizontal avec « Opérations > Profils d'intensité > Profil de ligne ». Nous définissons la position de la ligne sur la position du centre de gravité calculée précédemment (c'est-à-dire 668).

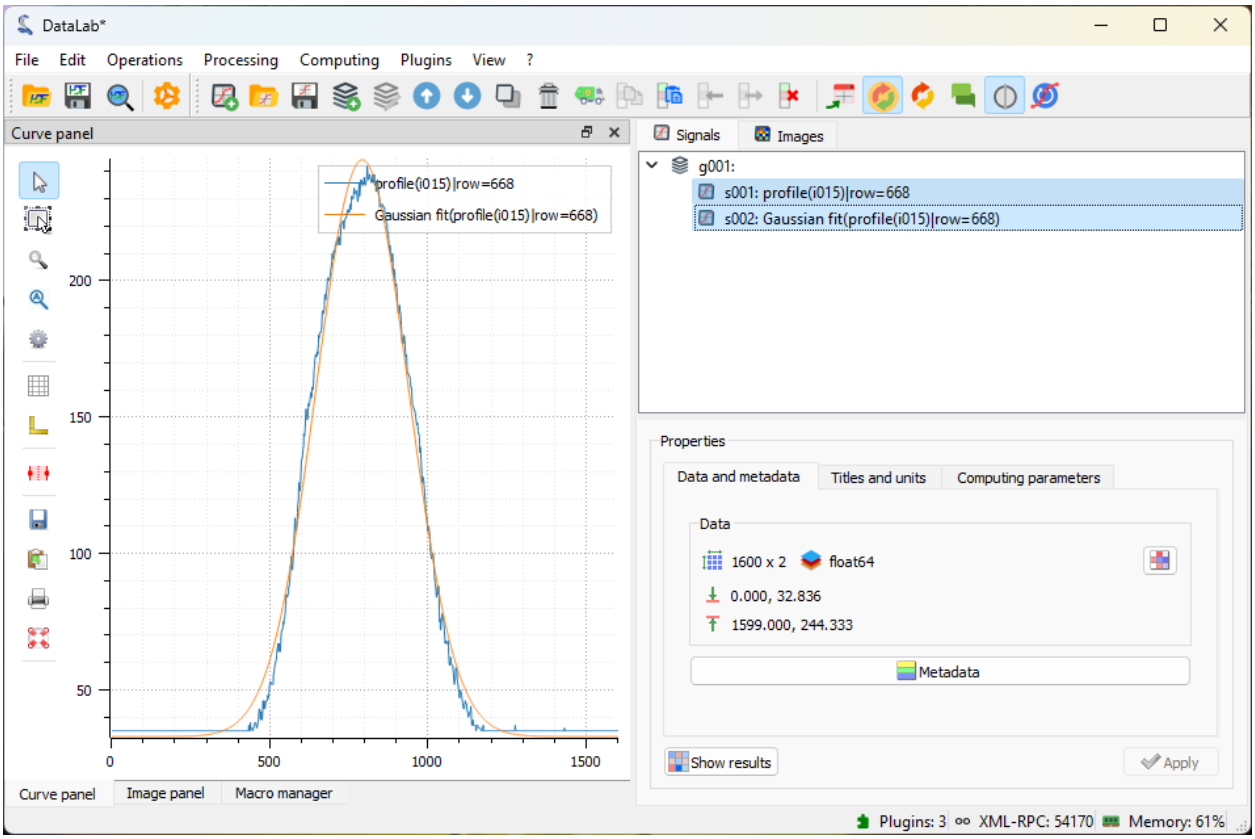


FIG. 77 – Le profil d'intensité est affiché dans le panneau « Signaux ». Nous pouvons ajuster le profil à une fonction gaussienne avec « Traitement > Ajustement > Ajustement gaussien ». Ici, nous avons sélectionné les deux signaux.

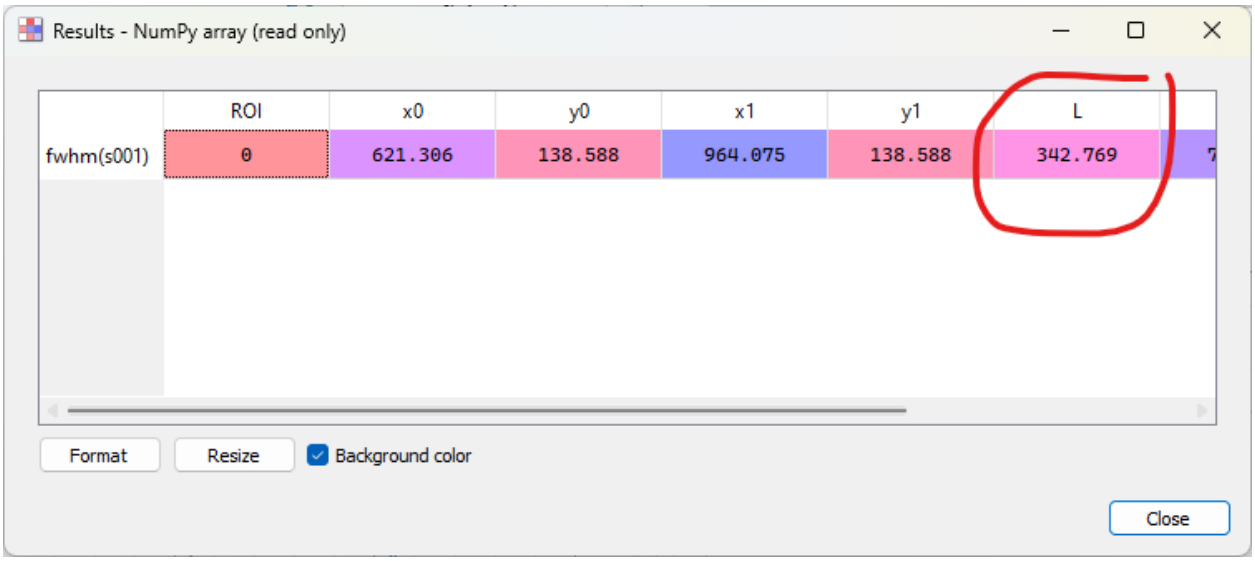


FIG. 78 – Si nous revenons au premier signal, le profil d'intensité, nous pouvons également calculer directement la FWHM avec « Calcul > Largeur à mi-hauteur ». La boîte de dialogue « Résultats » affiche beaucoup d'informations sur le calcul, y compris la valeur FWHM (qui est la colonne *L*, « L » pour « Longueur » car la forme du résultat est un segment et FWHM est la longueur du segment).

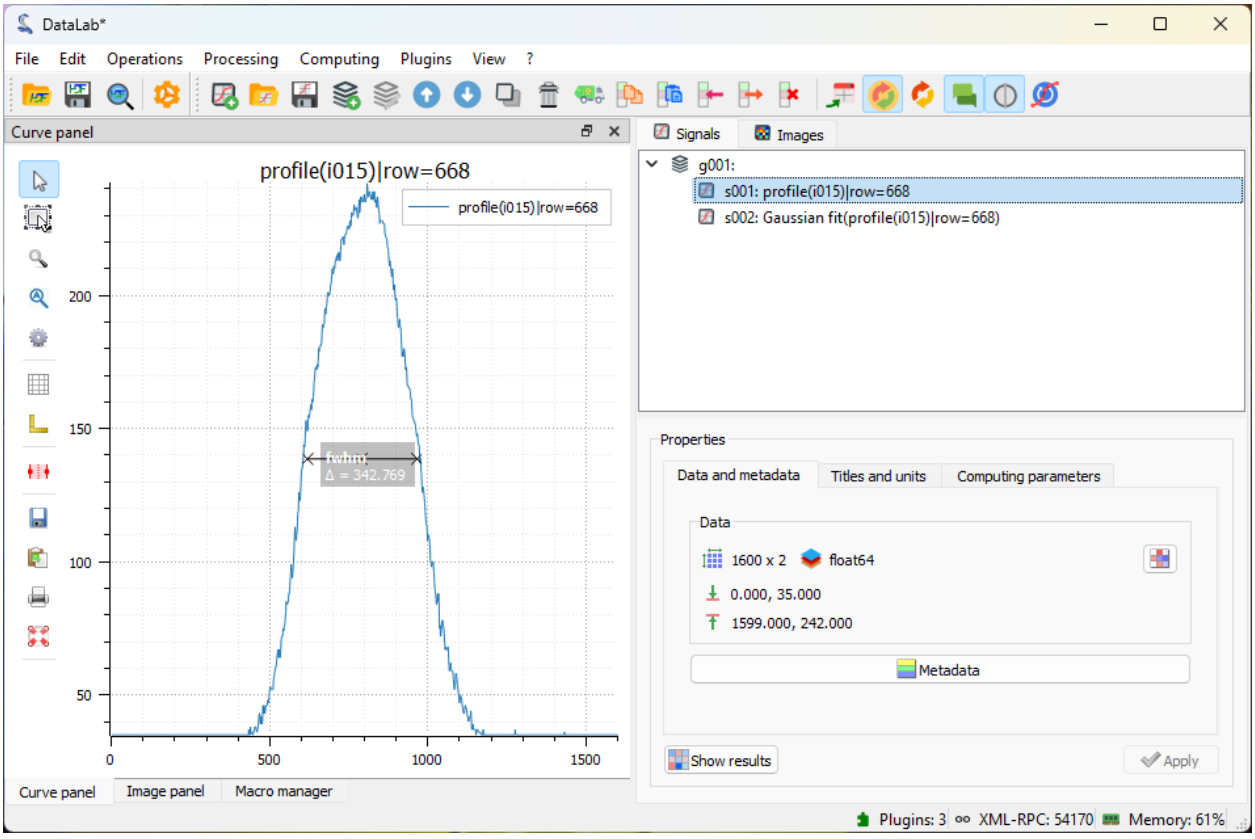


FIG. 79 – La largeur à mi-hauteur est également affichée sur la courbe, avec une étiquette optionnelle (ici, le titre de cette mesure a été affiché avec « Affichage > Afficher les titres des objets graphiques » ou le bouton dans la barre d'outils).

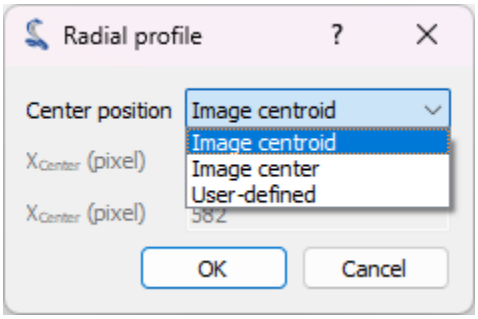


FIG. 80 – Le profil d'intensité radial peut être calculé autour de la position du centre de gravité, ou autour du centre de l'image, ou autour d'une position définie par l'utilisateur. Ici, nous avons sélectionné la position du centre de gravité.

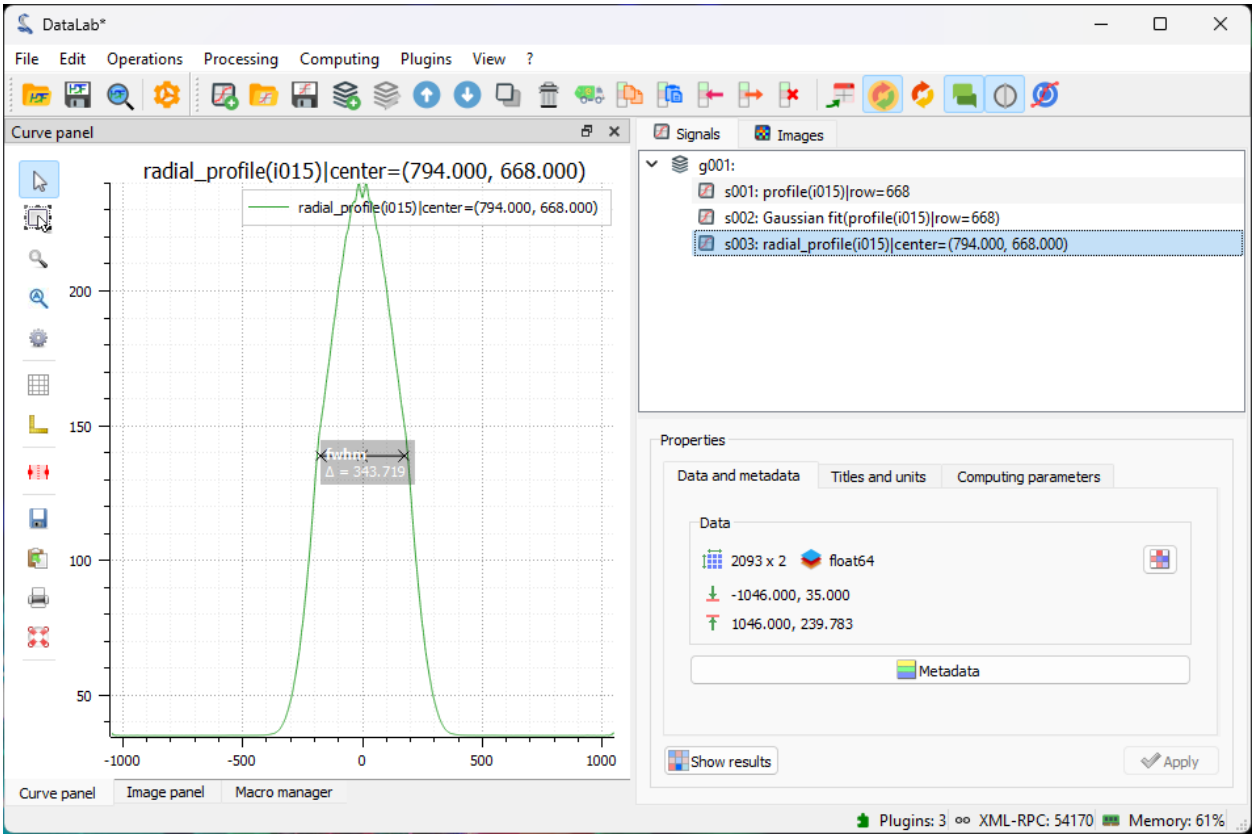


Fig. 81 – Le profil d'intensité radial est affiché dans le panneau « Signaux ». Il est plus lisse que le profil de ligne, car il est calculé à partir d'un plus grand nombre de pixels, ce qui permet de lisser le bruit.

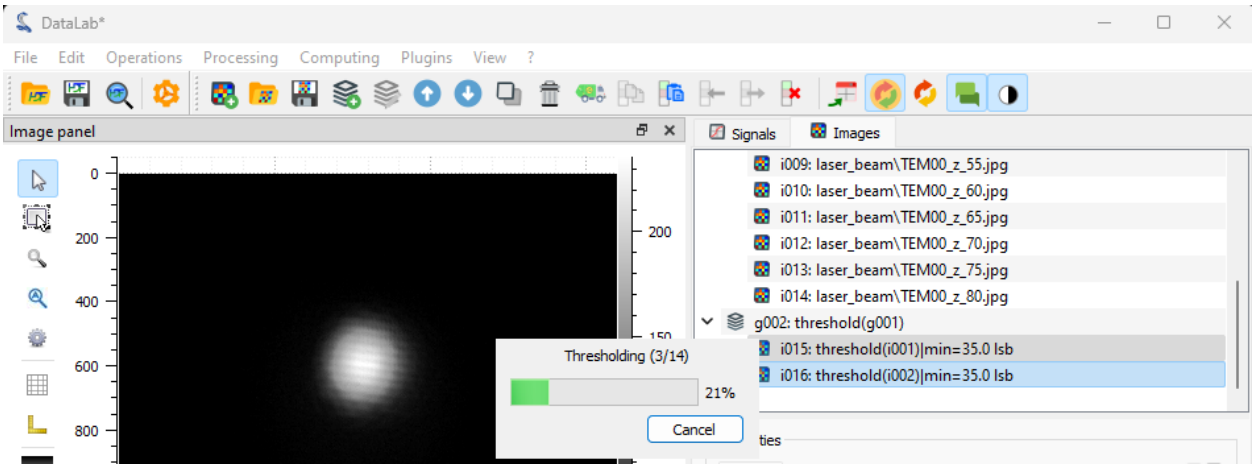


Fig. 82 – Nous appliquons le seuil à toutes les images, puis nous extrayons le profil d'intensité radial pour toutes les images (après avoir sélectionné l'ensemble du groupe « g002 » - il devrait être automatiquement sélectionné si vous aviez sélectionné « g001 » avant d'appliquer le seuil).

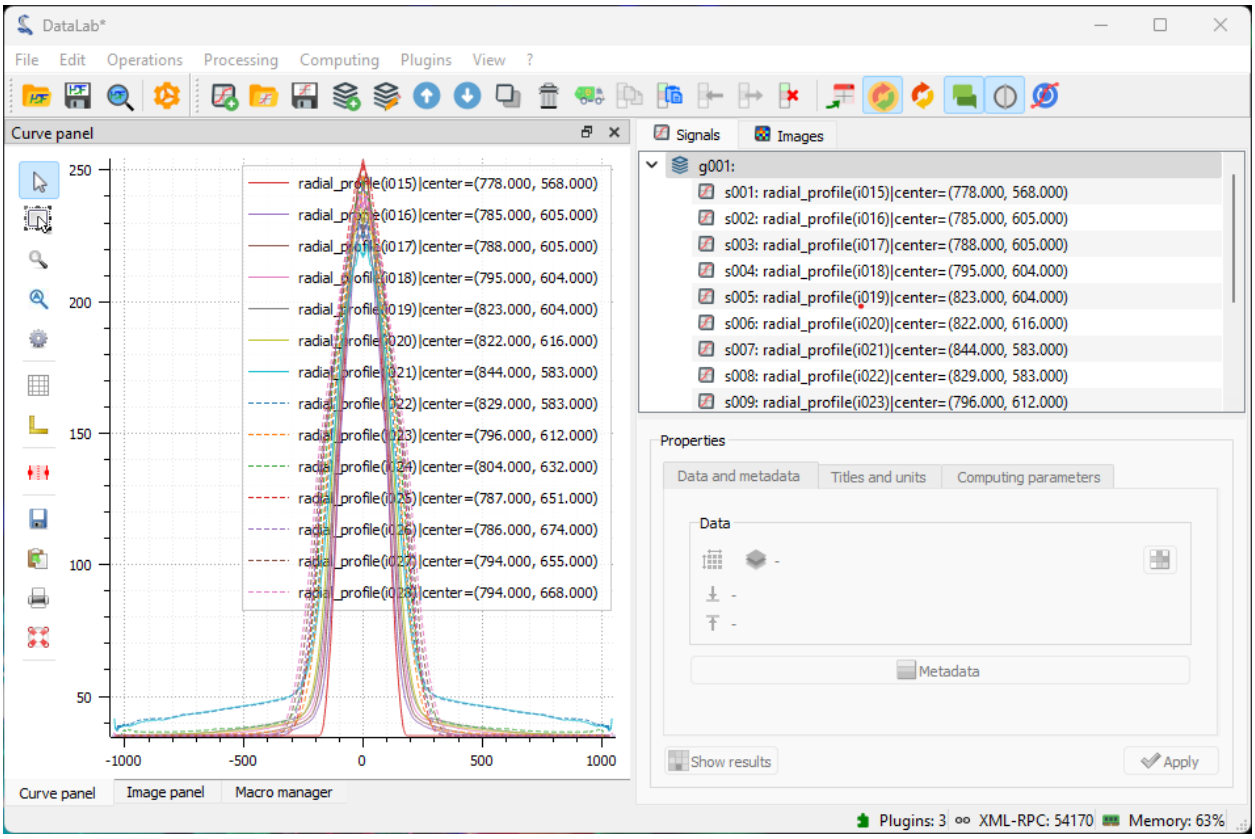


FIG. 83 – Le panneau « Signaux » contient maintenant tous les profils d'intensité radiaux.

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

FIG. 84 – Nous pouvons calculer la FWHM de tous les profils d'intensité radiaux : la boîte de dialogue « Résultats » affiche les valeurs FWHM pour tous les profils.

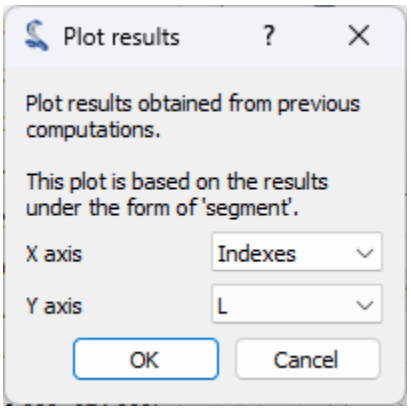


FIG. 85 – Enfin, nous pouvons tracer la taille du faisceau en fonction de la position le long de l'axe de propagation. Pour ce faire, nous utilisons la fonction « Tracer les résultats » dans le menu « Calcul ». Cette fonction permet de tracer les ensembles de données de résultats en choisissant les axes x et y parmi les colonnes de résultats. Ici, nous choisissons de tracer les valeurs FWHM (*L*) en fonction de l'index de l'image (*Indexes*).

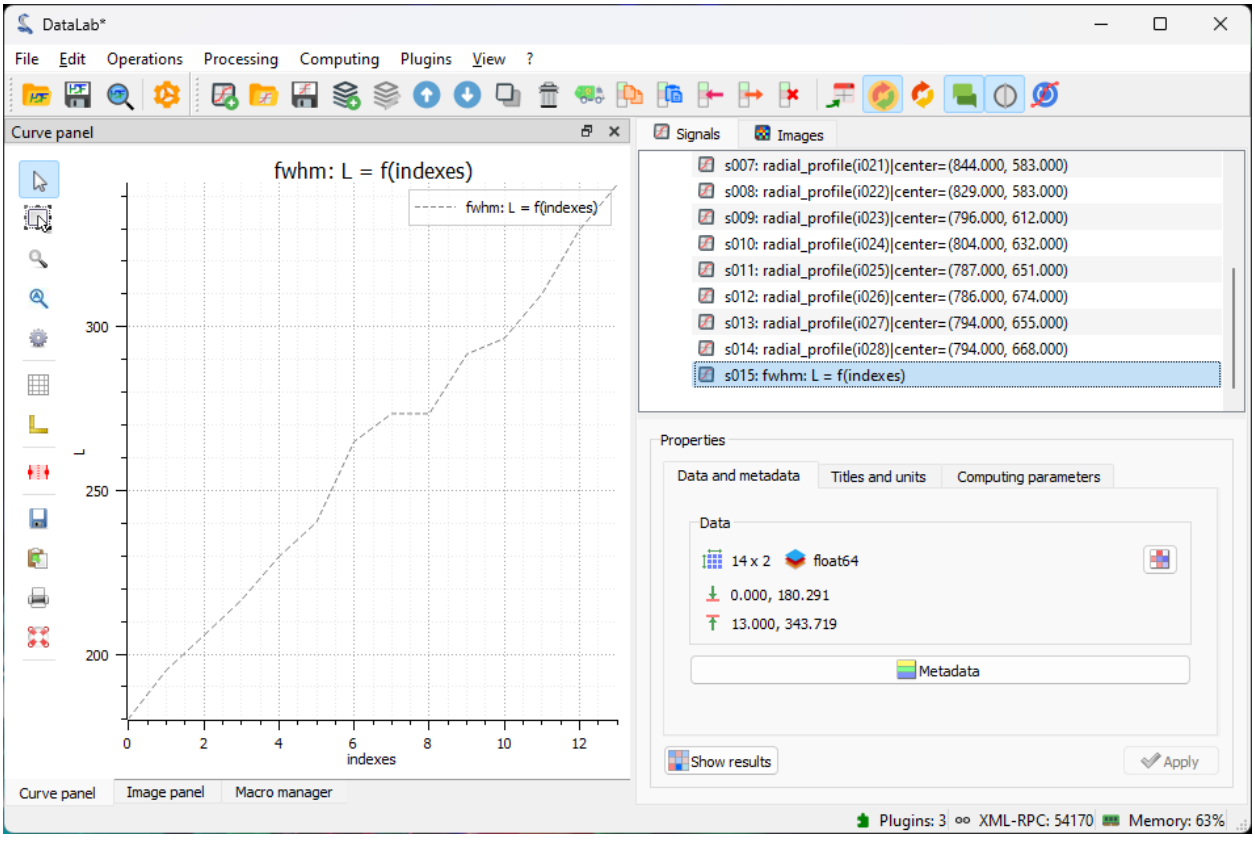


FIG. 86 – Le tracé est affiché dans le panneau « Signaux » et montre que la taille du faisceau augmente avec la position le long de l'axe de propagation (la position est ici en unités arbitraires, l'index de l'image).

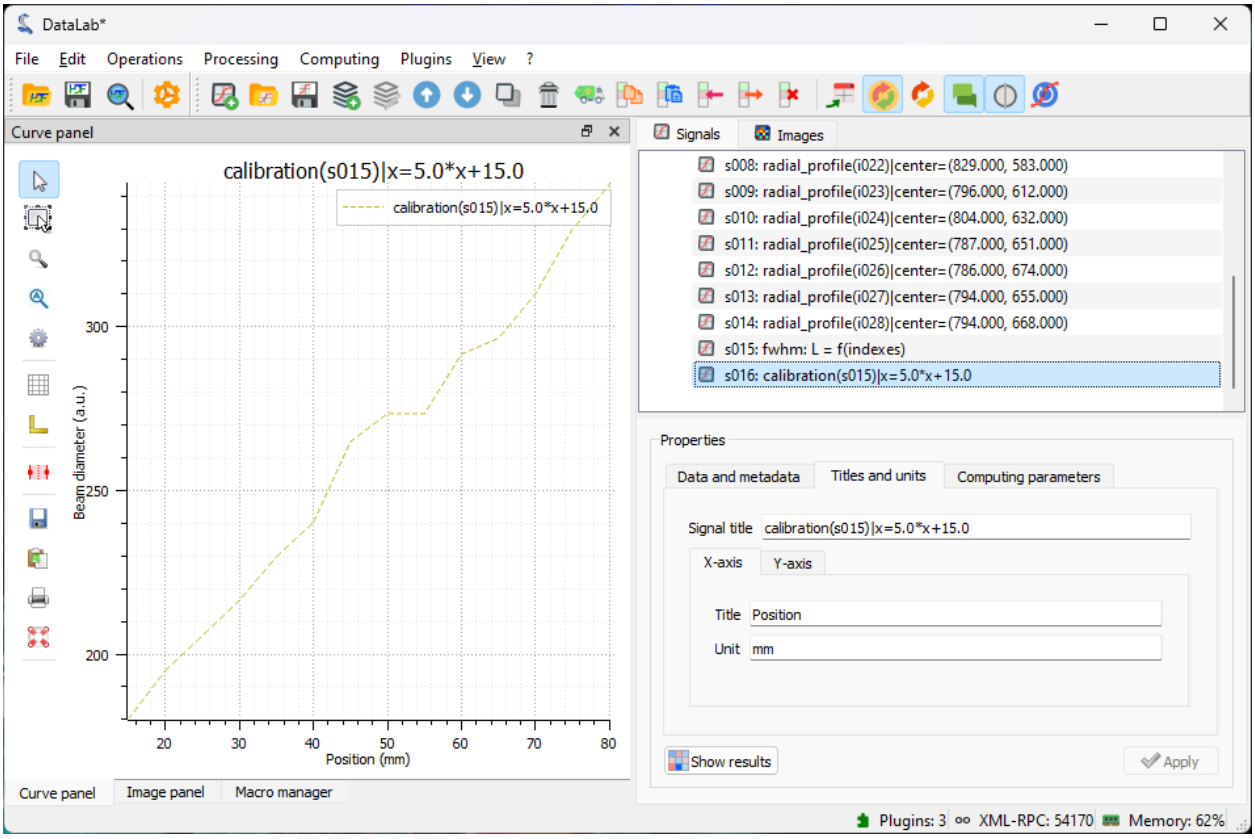


FIG. 87 – Nous pouvons également étalonner les axes X et Y en utilisant « Traitement > Étalonnage linéaire ». Ici, nous avons réglé l'axe X sur la position en mm (et entré le titre et l'unité dans le groupe de boîtes « Propriétés »).

visualisation (cartes de couleurs, contraste, etc.), les métadonnées et les annotations.

Si vous souhaitez charger à nouveau l'espace de travail, vous pouvez utiliser « Fichier > Ouvrir un fichier HDF5… » (ou le bouton dans la barre d'outils) pour charger l'ensemble de l'espace de travail, ou « Fichier > Parcourir un fichier HDF5… » (ou le bouton dans la barre d'outils) pour charger uniquement une sélection d'ensembles de données de l'espace de travail.

Prototypage d'une chaîne de traitement personnalisée

Cet exemple montre comment prototyper une chaîne de traitement d'image personnalisée en utilisant DataLab :

- Définir une fonction de traitement personnalisée
- Créer une macro-commande pour appliquer la fonction à une image
- Utiliser le même code à partir d'un IDE externe (par exemple Spyder) ou d'un notebook Jupyter
- Créer un plugin pour intégrer la fonction dans l'interface graphique de DataLab

Définir une fonction de traitement personnalisée

Pour illustrer l'extensibilité de DataLab, nous utiliserons une fonction de traitement d'image simple qui n'est pas disponible dans la distribution standard de DataLab, et qui représente un cas d'utilisation typique pour le prototypage d'une chaîne de traitement personnalisée.

La fonction sur laquelle nous allons travailler est un filtre de débruitage qui combine les idées de moyennage et de détection de contours. Ce filtre va moyenner les valeurs de pixels dans le voisinage, mais avec une particularité : il donnera moins de poids aux pixels qui sont significativement différents du pixel central, en supposant qu'ils pourraient faire partie d'un bord ou d'un bruit.

Voici le code de la fonction `weighted_average_denoise` :

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

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

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

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

Pour tester notre fonction de traitement, nous utiliserons une image générée à partir d'un exemple de plugin DataLab (plugins/examples/cdl_example_imageproc.py). Avant de commencer, assurez-vous que le plugin est installé dans DataLab (voir les premières étapes du tutoriel *Détection de taches sur une image*).

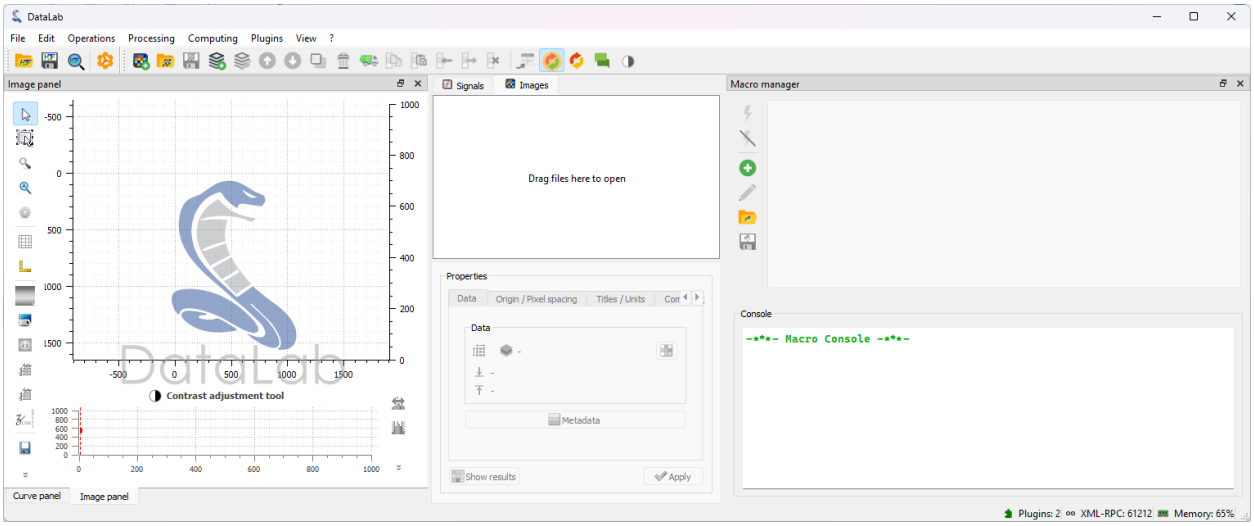


FIG. 88 – Pour commencer, nous réorganisons la disposition de la fenêtre de DataLab pour avoir le « Panneau Image » à gauche et le « Gestionnaire de Macros » à droite.

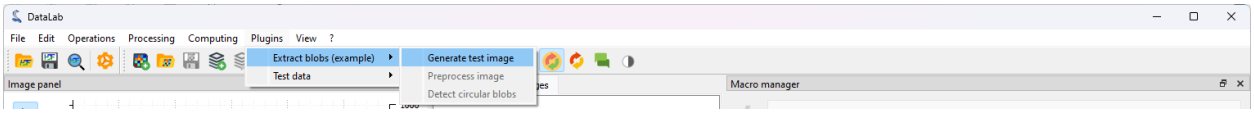


FIG. 89 – Nous générons une nouvelle image en utilisant le menu « Plugins > Extract blobs (example) > Generate test image ».

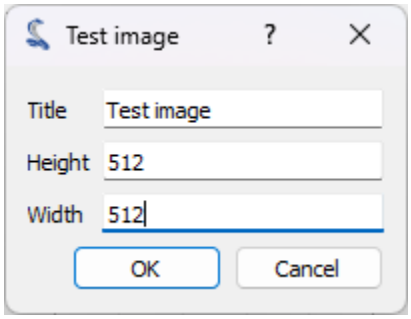


FIG. 90 – Nous sélectionnons une taille limitée pour l'image (par exemple 512x512 pixels) car notre algorithme est assez lent, et cliquons sur « OK ».

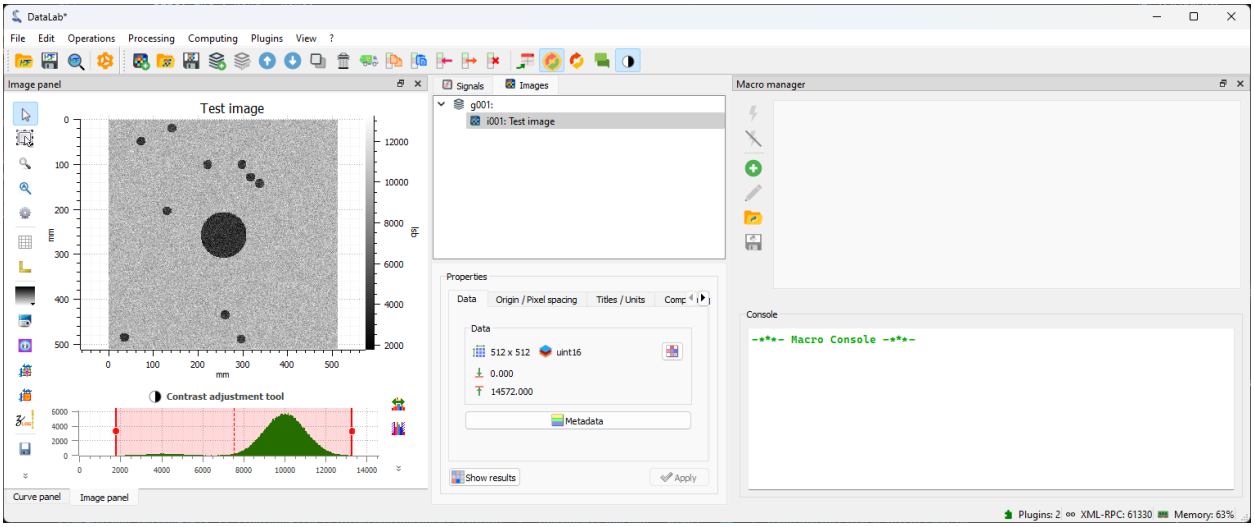


FIG. 91 – Nous pouvons maintenant voir l'image générée dans le « Panneau Image ».

Créer une macro-commande

Revenons à notre fonction personnalisée. Nous pouvons créer une nouvelle macro-commande qui appliquera la fonction à l'image actuelle. Pour ce faire, nous ouvrons le « Gestionnaire de Macros » et cliquons sur le bouton « Nouvelle macro » .

DataLab crée une nouvelle macro-commande qui n'est pas vide : elle contient un code d'exemple qui montre comment créer une nouvelle image et l'ajouter au « Panneau Image ». Nous pouvons supprimer ce code et le remplacer par le nôtre :

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

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

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

# Initialize the proxy to DataLab
proxy = RemoteProxy()

# Switch to the "Image Panel" and get the current image
proxy.set_current_panel("image")
image = proxy.get_object()
if image is None:
```

(suite sur la page suivante)

(suite de la page précédente)

```
    # We raise an explicit error if there is no image to process
    raise RuntimeError("No image to process!")

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

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

Dans DataLab, les macro-commandes sont simplement des scripts Python :

- Les macros font partie de l'**espace de travail** de DataLab, ce qui signifie qu'elles sont sauvegardées et restaurées lors de l'exportation et de l'importation vers/depuis un fichier HDF5.
- Les macros sont exécutées dans un processus séparé, nous devons donc importer les modules nécessaires et initialiser le proxy vers DataLab. Le proxy est un objet spécial qui permet de communiquer avec DataLab.
- En conséquence, **lors de la définition d'un plugin ou du contrôle de DataLab à partir d'un IDE externe, nous pouvons utiliser exactement le même code que dans la macro-commande**. C'est un point très important, car cela signifie que nous pouvons prototyper notre chaîne de traitement dans DataLab, puis utiliser le même code dans un plugin ou dans un IDE externe pour le développer davantage.

Note : La macro-commande est exécutée dans l'environnement Python de DataLab, nous pouvons donc utiliser les modules disponibles dans DataLab. Cependant, nous pouvons également utiliser nos propres modules, tant qu'ils sont installés dans l'environnement Python de DataLab ou dans une distribution Python compatible avec l'environnement Python de DataLab.

Si vos modules personnalisés ne sont pas installés dans l'environnement Python de DataLab, et s'ils sont compatibles avec la version Python de DataLab, vous pouvez préfixer `sys.path` avec le chemin vers la distribution Python qui contient vos modules :

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

Cela vous permettra d'importer vos modules dans la macro-commande et de les mélanger avec les modules disponibles dans DataLab.

Avertissement : Si vous utilisez cette méthode, assurez-vous que vos modules sont compatibles avec la version Python de DataLab. Sinon, vous obtiendrez des erreurs lors de leur importation.

Maintenant, exécutons la macro-commande en cliquant sur le bouton « Exécuter la macro » :

- La macro-commande est exécutée dans un processus séparé, nous pouvons donc continuer à travailler dans DataLab pendant que la macro-commande s'exécute. Et, si la macro-commande prend trop de temps à s'exécuter, nous pouvons l'arrêter en cliquant sur le bouton « Arrêter la macro » .
- Pendant l'exécution de la macro-commande, nous pouvons voir la progression dans la fenêtre « Gestionnaire de Macros » : la sortie standard du processus est affichée dans la « Console » en dessous de l'éditeur de macro. Nous pouvons voir les messages suivants :
  - `---[...]---[# ==> Running 'Untitled 01' macro...]` : la macro-commande démarre
  - `Connecting to DataLab XML-RPC server...OK [...]` : le proxy est connecté à DataLab
  - `---[...]---[# <== 'Untitled 01' macro has finished]` : la macro-commande se termine

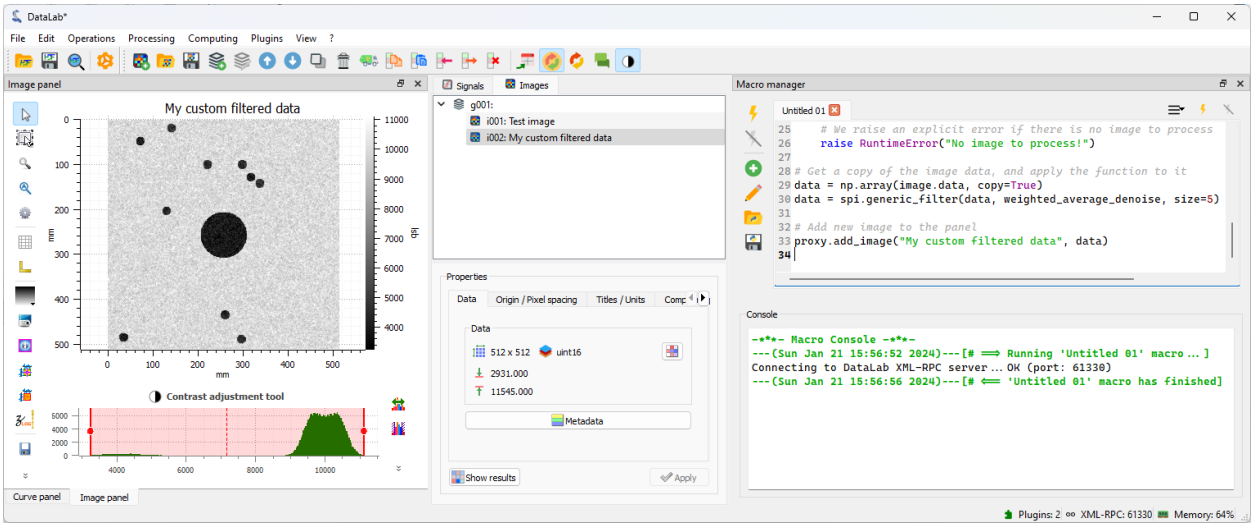


FIG. 92 – Lorsque la macro-commande est terminée, nous pouvons voir la nouvelle image dans le « Panneau Image ». Notre filtre a été appliqué à l'image, et nous pouvons voir que le bruit a été réduit.

Prototypage avec un IDE externe

Maintenant que nous avons un prototype fonctionnel de notre chaîne de traitement, nous pouvons utiliser le même code dans un IDE externe pour le développer davantage.

Par exemple, nous pouvons utiliser l'IDE Spyder pour déboguer notre code. Pour ce faire, nous devons installer Spyder mais pas nécessairement dans l'environnement Python de DataLab (dans le cas de la version autonome de DataLab, ce ne serait de toute façon pas possible).

Le seul prérequis est d'installer un client DataLab dans l'environnement Python de Spyder :

— Si vous utilisez la version autonome de DataLab ou si vous voulez ou devez garder DataLab et Spyder dans des environnements Python séparés, vous pouvez installer le DataLab Simple Client (`cdl-client`) en utilisant le gestionnaire de paquets `pip` :

```
pip install cdl-client
```

Ou vous pouvez également installer le paquet Python DataLab (`cdl`) qui inclut le client (mais aussi d'autres modules, donc nous ne recommandons pas cette méthode si vous n'avez pas besoin de toutes les fonctionnalités de DataLab dans cet environnement Python) :

```
pip install cdl
```

— Si vous utilisez le paquet Python DataLab, vous pouvez exécuter Spyder dans le même environnement Python que DataLab, vous n'avez donc pas besoin d'installer le client : il est déjà disponible dans le paquet principal DataLab (le paquet `cdl`).

Une fois le client installé, nous pouvons démarrer Spyder et créer un nouveau script Python :

```
# -*- 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

```

(suite sur la page suivante)

(suite de la page précédente)

```
@author: p.raybaut
"""

# %% Importing necessary modules

import numpy as np
import scipy.ndimage as spi
from cdlclient import SimpleRemoteProxy

# %% Connecting to DataLab current session

proxy = SimpleRemoteProxy()
proxy.connect()

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


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

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

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

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


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

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

# Add new image to the panel
proxy.add_image("Filtered using Spyder", data)
```

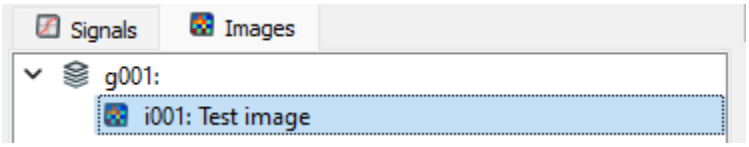


FIG. 93 – Nous retournons à DataLab et sélectionnons la première image dans le « Panneau Image ».

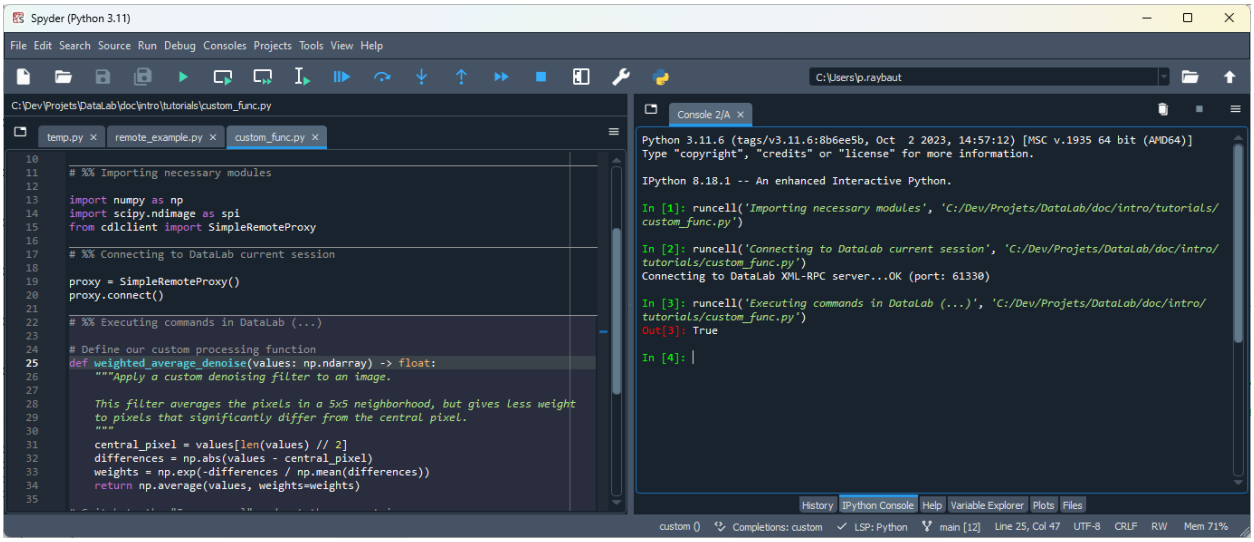


FIG. 94 – Ensuite, nous exécutons le script dans Spyder, étape par étape (en utilisant les cellules définies), et nous pouvons voir le résultat dans DataLab.

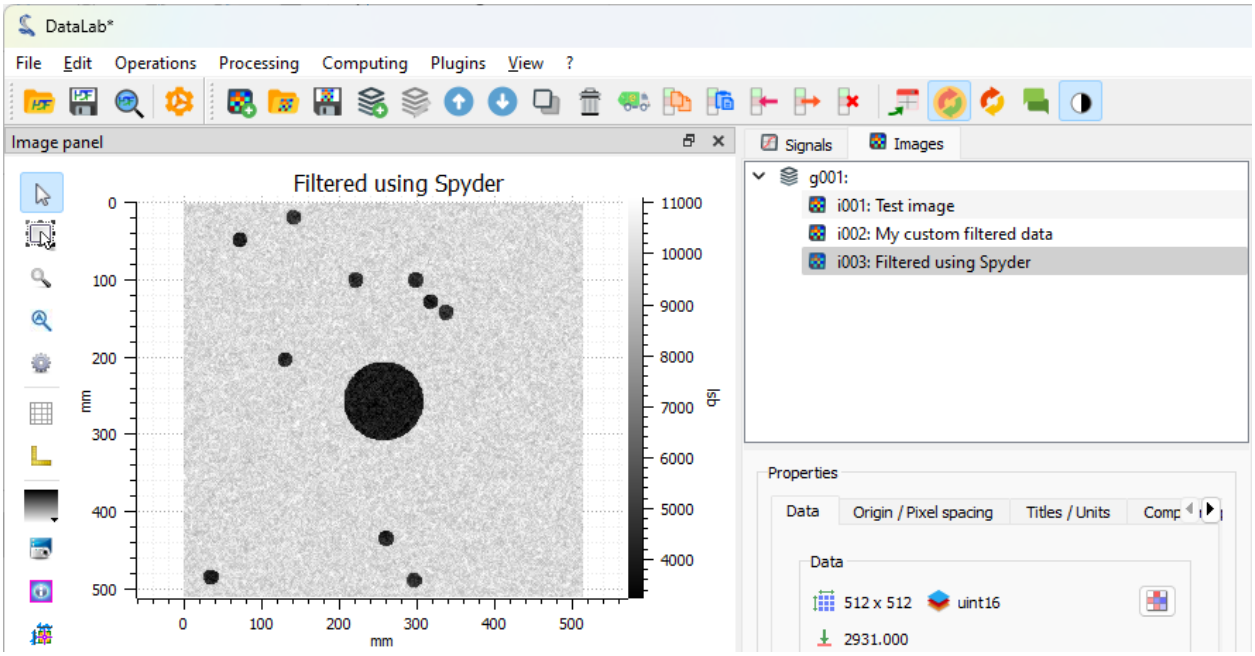


FIG. 95 – Nous pouvons voir dans DataLab qu'une nouvelle image a été ajoutée au « Panneau Image ». Cette image est le résultat de l'exécution du script dans Spyder. Ici, nous avons utilisé le script sans aucune modification, mais nous aurions pu le modifier pour tester de nouvelles idées, puis utiliser le script modifié dans DataLab.

Prototypage avec un notebook Jupyter

Nous pouvons également utiliser un notebook Jupyter pour prototyper notre chaîne de traitement. Pour ce faire, nous devons installer Jupyter mais pas nécessairement dans l'environnement Python de DataLab (dans le cas de la version autonome de DataLab, ce ne serait de toute façon pas possible).

Le seul prérequis est d'installer un client DataLab dans l'environnement Python de Jupyter (voir la section précédente pour plus de détails : c'est exactement la même procédure que pour Spyder ou tout autre IDE comme Visual Studio Code, par exemple).

DataLab custom function example

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

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

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

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

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

Connecting to DataLab current session:

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

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

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

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

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

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

FIG. 96 – Une fois le client installé, nous pouvons démarrer Jupyter et créer un nouveau notebook.

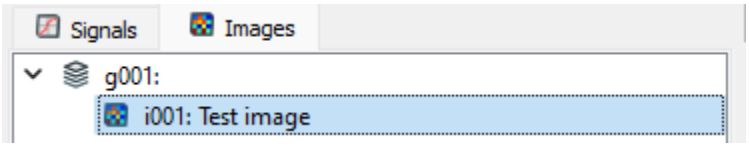


FIG. 97 – Nous retournons à DataLab et sélectionnons la première image dans le « Panneau Image ».

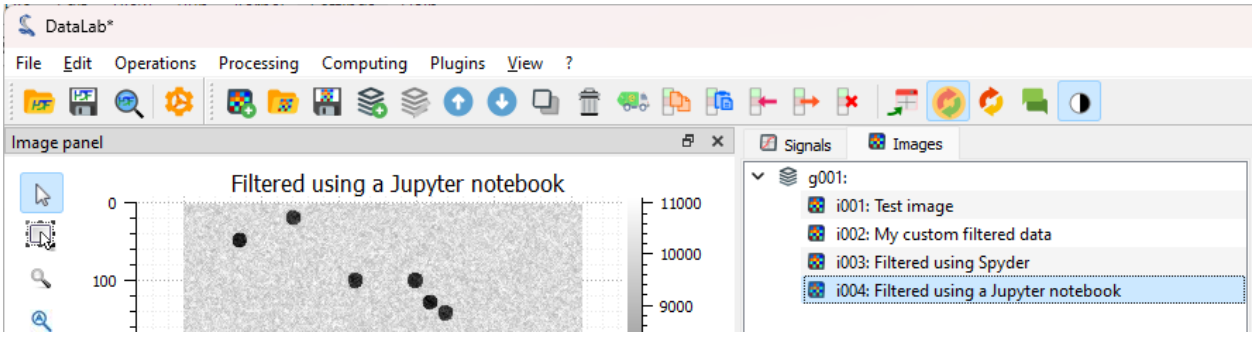


FIG. 98 – Ensuite, nous exécutons le notebook dans Jupyter, étape par étape (en utilisant les cellules définies), et nous pouvons voir le résultat dans DataLab. Une fois de plus, nous pouvons voir dans DataLab qu'une nouvelle image a été ajoutée au « Panneau Image ». Cette image est le résultat de l'exécution du notebook dans Jupyter. Comme pour le script dans Spyder, nous aurions pu modifier le notebook pour tester de nouvelles idées, puis utiliser le notebook modifié dans DataLab.

Création d'un plugin

Maintenant que nous avons un prototype fonctionnel de notre chaîne de traitement, nous pouvons créer un plugin pour l'intégrer dans l'interface graphique de DataLab. Pour ce faire, nous devons créer un nouveau module Python qui contiendra le code du plugin. Nous pouvons utiliser le même code que dans la macro-commande, mais nous devons apporter quelques modifications.

Voir aussi :

Le système de plugins est décrit dans la section *Plugins*.

Mis à part l'intégration de la fonctionnalité à l'interface graphique de DataLab qui est plus pratique pour l'utilisateur, l'avantage de créer un plugin est que nous pouvons tirer parti de l'infrastructure de DataLab, si nous encapsulons notre fonction de traitement d'une certaine manière (voir ci-dessous) :

— Notre fonction sera exécutée dans un processus séparé, nous pouvons donc l'interrompre si elle prend trop de temps à s'exécuter.
— Les avertissements et les erreurs seront gérés par DataLab, nous n'avons donc pas besoin de les gérer nous-mêmes.

Le changement le plus significatif est que nous devons définir une fonction qui fonctionnera sur les objets d'image natifs de DataLab (*cdl.obj.ImageObj*), au lieu de fonctionner sur des tableaux NumPy. Nous devons donc trouver un moyen d'appeler notre fonction personnalisée `weighted_average_denoise` avec un *cdl.obj.ImageObj* en entrée et en sortie. Pour éviter d'écrire beaucoup de code de base, nous pouvons utiliser l'enveloppe de fonction fournie par DataLab : *cdl.core.computation.image.Wrap11Func*.

Par ailleurs, nous devons définir une classe qui décrit notre plugin, qui doit hériter de *cdl.plugins.PluginBase* et nommer le script Python qui contient le code du plugin avec un nom qui commence par `cdl_` (par exemple `cdl_custom_func.py`), afin que DataLab puisse le découvrir au démarrage.

De plus, dans le code du plugin, nous voulons ajouter une entrée dans le menu « Plugins », afin que l'utilisateur puisse accéder à notre plugin depuis l'interface graphique.

Voici le code du plugin :

```
# -*- coding: utf-8 -*-

"""
Custom denoising filter plugin
==============================
```

(suite sur la page suivante)

(suite de la page précédente)

```
    This is a simple example of a DataLab image processing plugin.

    It is part of the DataLab custom function tutorial.

    .. note::

        This plugin is not installed by default. To install it, copy this file to
        your DataLab plugins directory (see `DataLab documentation
        <https://datalab-platform.com/en/features/general/plugins.html>`_).
"""

import numpy as np
import scipy.ndimage as spi

import cdl.core.computation.image as cpi
import cdl.obj
import cdl.param
import cdl.plugins


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

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

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

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


class CustomFilters(cdl.plugins.PluginBase):
    """DataLab Custom Filters Plugin"""

    PLUGIN_INFO = cdl.plugins.PluginInfo(
        name="My custom filters",
        version="1.0.0",
        description="This is an example plugin",
    )

    def create_actions(self) -> None:
        """Create actions"""
        acth = self.imagepanel.acthandler
        proc = self.imagepanel.processor
        with acth.new_menu(self.PLUGIN_INFO.name):
```

(suite sur la page suivante)

(suite de la page précédente)

```
            for name, func in (("Weighted average denoise", weighted_average_denoise),):
                # Wrap function to handle ``ImageObj`` objects instead of NumPy arrays
                wrapped_func = cpi.Wrap11Func(func)
                acth.new_action(
                    name, triggered=lambda: proc.compute_11(wrapped_func, title=name)
                )
```

Pour le tester, nous devons ajouter le script du plugin à l'un des répertoires de plugins qui sont découverts par DataLab au démarrage (voir la section *Plugins* pour plus de détails, ou le *Détection de taches sur une image* pour un exemple).

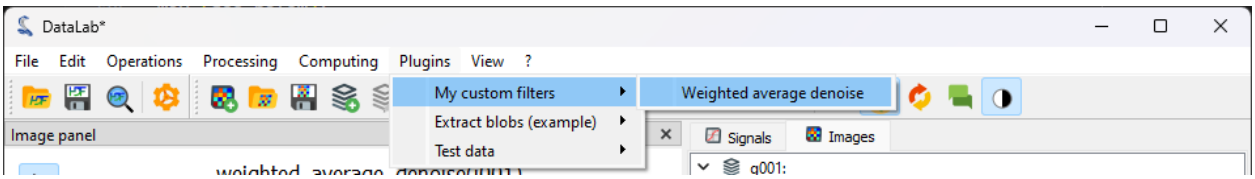


FIG. 99 – Nous redémarrons DataLab et nous pouvons voir que le plugin a été chargé.

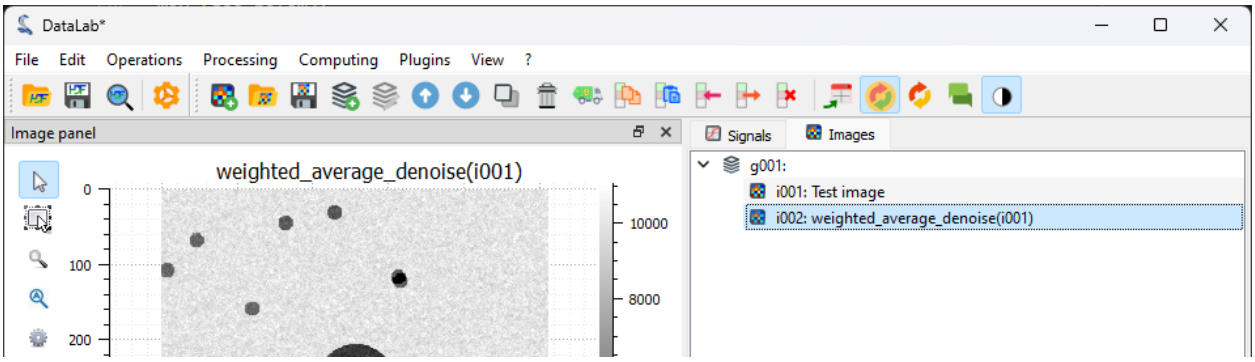
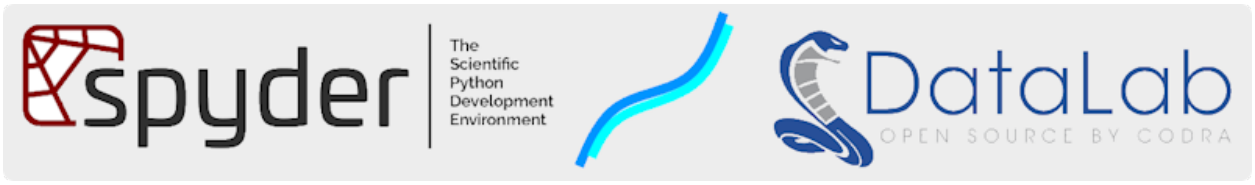


FIG. 100 – Nous générons à nouveau notre image de test en utilisant (voir les premières étapes du tutoriel), et nous la traitons en utilisant le plugin : « Plugins > My custom filters > Weighted average denoise ».

DataLab et Spyder : un mariage parfait

Ce tutoriel montre comment utiliser Spyder pour travailler avec DataLab à travers un exemple, en utilisant des algorithmes et des données fictifs qui représentent un travail de recherche/technique hypothétique. L'objectif est d'illustrer comment utiliser DataLab pour tester vos algorithmes avec des données, et comment les déboguer si nécessaire.

L'exemple est assez simple, mais il illustre les concepts de base du travail avec DataLab *et* Spyder.



Note : DataLab et Spyder sont des outils **complémentaires**. Alors que Spyder est un environnement de développement puissant avec des capacités de calcul scientifique interactif, DataLab est un outil d'analyse de données polyvalent qui peut être utilisé pour effectuer un large éventail de tâches, de la simple visualisation de données à l'analyse et au traitement de données complexes. En d'autres termes, Spyder est un outil de **développement**, tandis que DataLab est

un outil d'**analyse de données**. Vous pouvez utiliser Spyder pour développer des algorithmes, puis utiliser DataLab pour analyser les données avec ces algorithmes.

Concepts de base

Dans le contexte de votre travail de recherche ou technique, nous supposons que vous développez un logiciel pour traiter des données (signaux ou images) : ce logiciel peut être une application autonome ou une bibliothèque que vous utiliserez dans d'autres applications, ou même un simple script que vous exécuterez à partir de la ligne de commande. Dans tous les cas, vous devrez suivre un processus de développement qui comprendra les étapes suivantes :

0. Prototyper l'algorithme dans un environnement de développement, tel que Spyder.
1. Développer l'algorithme dans un environnement de développement, tel que Spyder.
2. Tester l'algorithme avec des données.
3. Déboguer l'algorithme si nécessaire.
4. Répéter les étapes 2 et 3 jusqu'à ce que l'algorithme fonctionne comme prévu.
5. Utiliser l'algorithme dans votre application.

Note : DataLab peut vous aider avec l'étape 0 car il fournit toutes les primitives de traitement dont vous avez besoin pour prototyper votre algorithme : vous pouvez charger des données, les visualiser et effectuer des opérations de traitement de base. Nous ne couvrirons pas cette étape dans les paragraphes suivants car la documentation de DataLab fournit déjà beaucoup d'informations à ce sujet.

Dans ce tutoriel, nous verrons comment utiliser DataLab pour effectuer les étapes 2 et 3. Nous supposons que vous avez déjà prototypé (de préférence dans DataLab !) et développé votre algorithme dans Spyder. Maintenant, vous voulez le tester avec des données, mais sans quitter Spyder car vous devrez peut-être apporter des modifications à votre algorithme et le re-tester. De plus, votre flux de travail est déjà configuré dans Spyder et vous ne voulez pas le changer.

Note : Dans ce tutoriel, nous supposons que vous avez déjà installé DataLab et que vous l'avez démarré. Si vous ne l'avez pas encore fait, veuillez vous référer à la section *Installation* de la documentation.

De plus, nous supposons que vous avez déjà installé Spyder et que vous l'avez démarré. Si vous ne l'avez pas encore fait, veuillez vous référer à la documentation de Spyder. **Notez que vous n'avez pas besoin d'installer DataLab dans le même environnement que Spyder.** : c'est tout l'intérêt de DataLab, c'est une application autonome qui peut être utilisée à partir de n'importe quel environnement. Pour ce tutoriel, vous n'avez besoin d'installer que le client DataLab Simple (*pip install cdlclient*) dans le même environnement que Spyder.

Tester votre algorithme avec DataLab

Supposons que vous ayez développé des algorithmes dans le module `my_work` de votre projet. Vous les avez déjà prototypés dans DataLab, et vous les avez développés dans Spyder en écrivant des fonctions qui prennent des données en entrée et renvoient des données traitées en sortie. Maintenant, vous voulez tester ces algorithmes avec des données.

Pour tester ces algorithmes, vous avez écrit deux fonctions dans le module `my_work` :

— `test_my_1d_algorithm` : cette fonction renvoie des données 1D qui vous permettront de valider votre premier algorithme qui fonctionne sur des données 1D.
— `test_my_2d_algorithm` : cette fonction renvoie des données 2D qui vous permettront de valider votre deuxième algorithme qui fonctionne sur des données 2D.

Vous pouvez maintenant utiliser DataLab pour visualiser les données renvoyées par ces fonctions directement depuis Spyder :

— Tout d'abord, vous devez démarrer DataLab et Spyder.

— Rappelez-vous que DataLab est une application autonome qui peut être utilisée à partir de n'importe quel environnement, vous n'avez donc pas besoin de l'installer dans le même environnement que Spyder car la connexion entre ces deux applications se fait via un protocole de communication.

Voici comment faire :

```
# %% Connecting to DataLab current session

from cdlclient import SimpleRemoteProxy

proxy = SimpleRemoteProxy()
proxy.connect()

# %% Visualizing 1D data from my work

from my_work import test_my_1d_algorithm

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


# %% Visualizing 2D data from my work

from my_work import test_my_2d_algorithm

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

Si nous exécutons les deux premières cellules, nous verrons la sortie suivante dans la console Spyder :

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

IPython 8.15.0 -- An enhanced Interactive Python.

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

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

Déboguer votre algorithme avec DataLab

Maintenant que vous avez testé vos algorithmes avec des données, vous voudrez peut-être les déboguer si nécessaire. Pour ce faire, vous pouvez combiner les capacités de débogage de Spyder avec DataLab.

Voici le code de l'algorithme fictif que nous voulons déboguer, dans lequel nous avons introduit un paramètre optionnel `debug_with_datalab` qui - s'il est défini sur `True` - créera un objet proxy permettant de visualiser les données étape par étape dans DataLab :

```
def generate_2d_data(
    num_lines: int = 10,
    noise_std: float = 0.1,
```

(suite sur la page suivante)

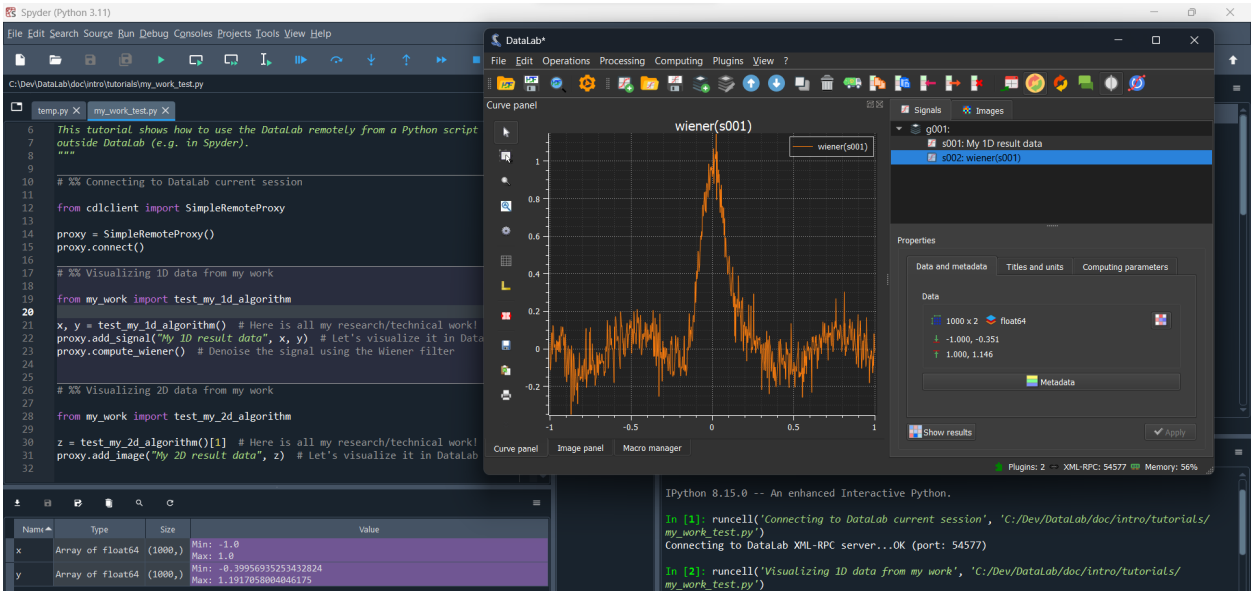


FIG. 101 – Sur cette capture d'écran, nous pouvons voir le résultat de l'évaluation des deux premières cellules : la première cellule se connecte à DataLab, et la deuxième cellule visualise les données 1D renvoyées par la fonction `test_my_1d_algorithm`.

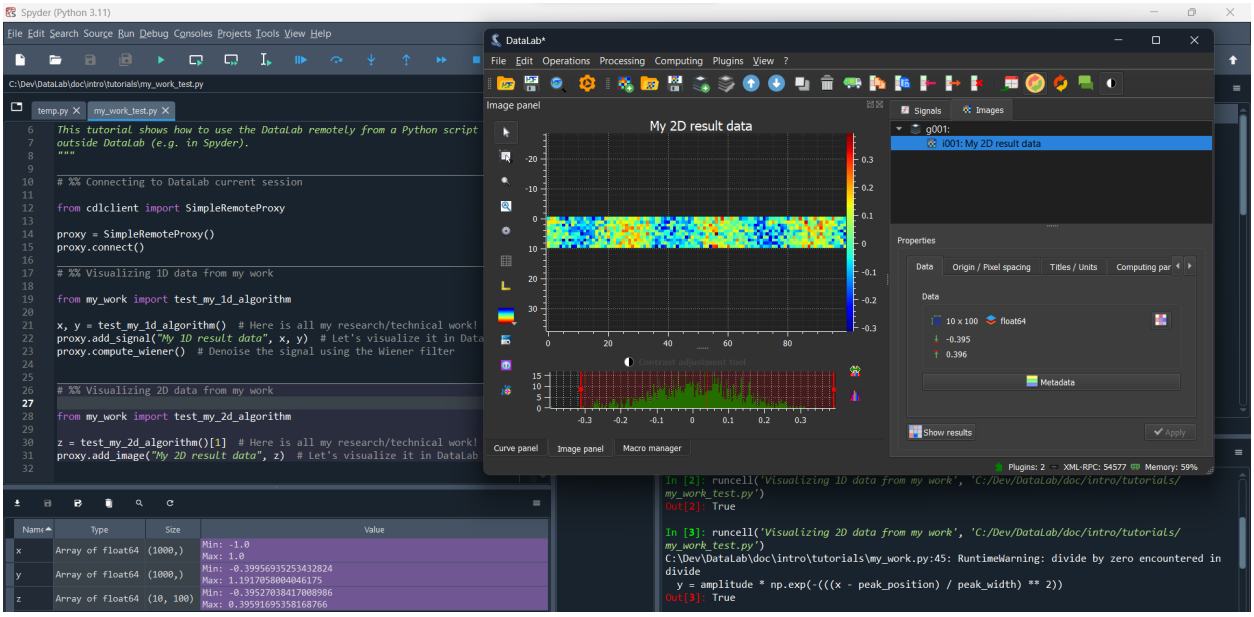


FIG. 102 – Sur cette capture d'écran, nous pouvons voir le résultat de l'évaluation de la troisième cellule : la fonction `test_my_2d_algorithm` renvoie un tableau 2D, et nous pouvons le visualiser directement dans DataLab.

(suite de la page précédente)

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

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

    Returns:
        Generated data (x, y; where y is a 2D array and x is a 1D array).
    """
    proxy = None
    if debug_with_datalab:
        proxy = SimpleRemoteProxy()
        proxy.connect()
    z = np.zeros((num_lines, num_points))
    for i in range(num_lines):
        amplitude = 0.1 * i**2
        peak_position = 0.5 * i**2
        peak_width = 0.1 * i**2
        x, y = generate_1d_data(
            amplitude,
            peak_position,
            peak_width,
            relaxation_oscillations=True,
            noise=True,
            noise_std=noise_std,
            x_min=x_min,
            x_max=x_max,
            num_points=num_points,
        )
        z[i] = y
        if proxy is not None:
            proxy.add_signal(f"Line {i}", x, y)
    return x, z
```

La fonction `test_my_2d_algorithm` correspondante a également un paramètre optionnel `debug_with_datalab` qui est simplement passé à la fonction `generate_2d_data`.

Maintenant, nous pouvons utiliser Spyder pour déboguer la fonction `test_my_2d_algorithm` :

```
# %% Debugging my work with DataLab

from my_work import generate_2d_data
```

(suite sur la page suivante)

(suite de la page précédente)

```
x, z = generate_2d_data(debug_with_datalab=True)
```

Dans cet exemple simple, l'algorithme itère simplement 10 fois et génère un tableau 1D à chaque itération. Chaque tableau 1D est ensuite empilé dans un tableau 2D qui est renvoyé par la fonction `generate_2d_data`. Avec le paramètre `debug_with_datalab` défini sur `True`, nous pouvons visualiser chaque tableau 1D dans DataLab : de cette façon, nous pouvons vérifier que l'algorithme fonctionne comme prévu.

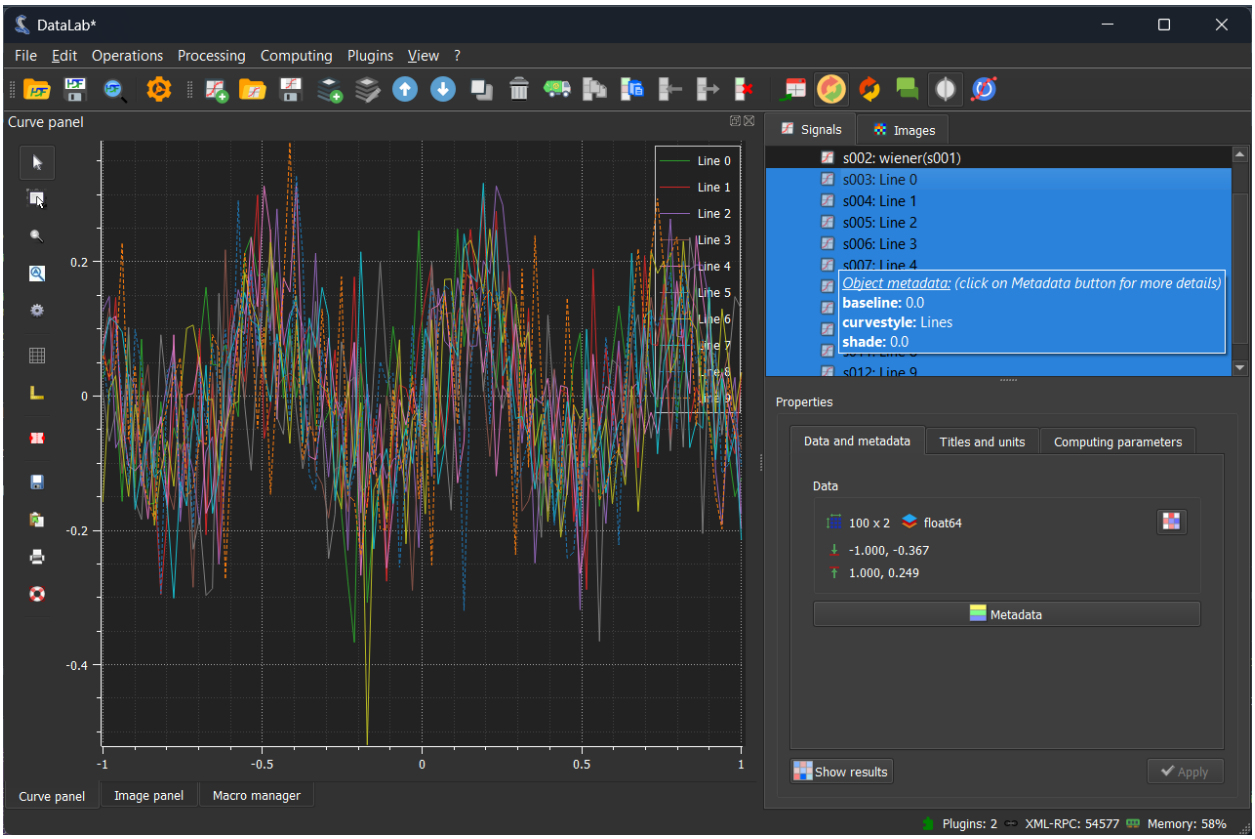


FIG. 103 – Sur cette capture d'écran, nous pouvons voir le résultat de l'évaluation de la première cellule : la fonction `test_my_2d_algorithm` est appelée avec le paramètre `debug_with_datalab` défini sur `True` : 10 tableaux 1D sont générés et visualisés dans DataLab.

Note : Si nous avions exécuté le script en utilisant le débogueur Spyder et défini un point d'arrêt dans la fonction `generate_2d_data`, nous aurions vu les tableaux 1D générés dans DataLab à chaque itération : comme DataLab est exécuté dans un processus séparé, nous aurions pu manipuler les données dans DataLab pendant que l'algorithme est en pause dans Spyder.

CHAPITRE 2

Fonctionnalités

Les fonctionnalités de DataLab sont détaillées dans les sections suivantes. La première section décrit les fonctionnalités générales de DataLab, puis les deux sections suivantes décrivent les fonctionnalités spécifiques aux panneaux de traitement du signal et d'image.

Avant de plonger dans les détails des autres parties de la documentation, il est recommandé de commencer par la page *Espace de travail*, qui décrit les concepts de base de DataLab.

Voir aussi :

Pour une vue d'ensemble synthétique des fonctionnalités de DataLab, veuillez vous référer à la page *Fonctionnalités principales*.

2.1 Fonctionnalités générales

Cette section décrit les fonctionnalités générales de DataLab, qui concernent à la fois les panneaux de traitement du signal et d'image.

2.1.1 Espace de travail

Travailler avec DataLab est très simple. L'interface est intuitive et appréhendable facilement. La fenêtre principale est divisée en deux zones principales :

- La zone de droite affiche la liste des jeux de données actuellement chargés dans DataLab, répartis sur deux onglets : **Signaux** et **Images**. L'utilisateur peut basculer entre les deux onglets en cliquant sur l'onglet correspondant : cela bascule la fenêtre principale vers le panneau correspondant, ainsi que le contenu du menu et de la barre d'outils. Sous la liste des jeux de données, une vue **Propriétés** affiche des informations sur le jeu de données actuellement sélectionné.
- La zone de gauche affiche la visualisation du jeu de données actuellement sélectionné. La visualisation est mise à jour automatiquement lorsque l'utilisateur sélectionne un nouveau jeu de données dans la liste des jeux de données.

DataLab a son propre modèle de données interne, dans lequel les jeux de données sont organisés autour d'une structure arborescente. Chaque panneau de la fenêtre principale correspond à une branche de l'arbre. Chaque jeu de données

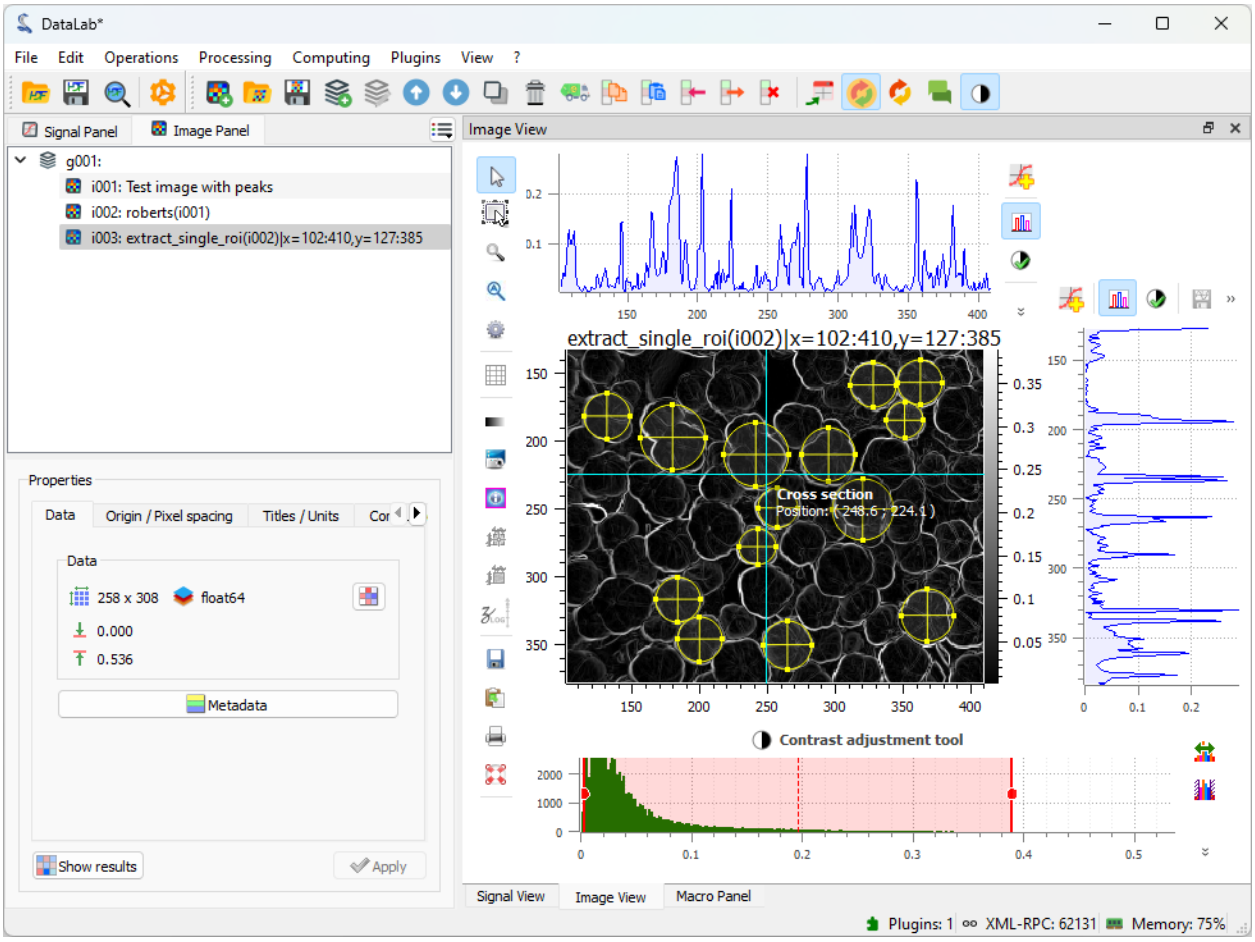


FIG. 1 – Fenêtre principale de DataLab

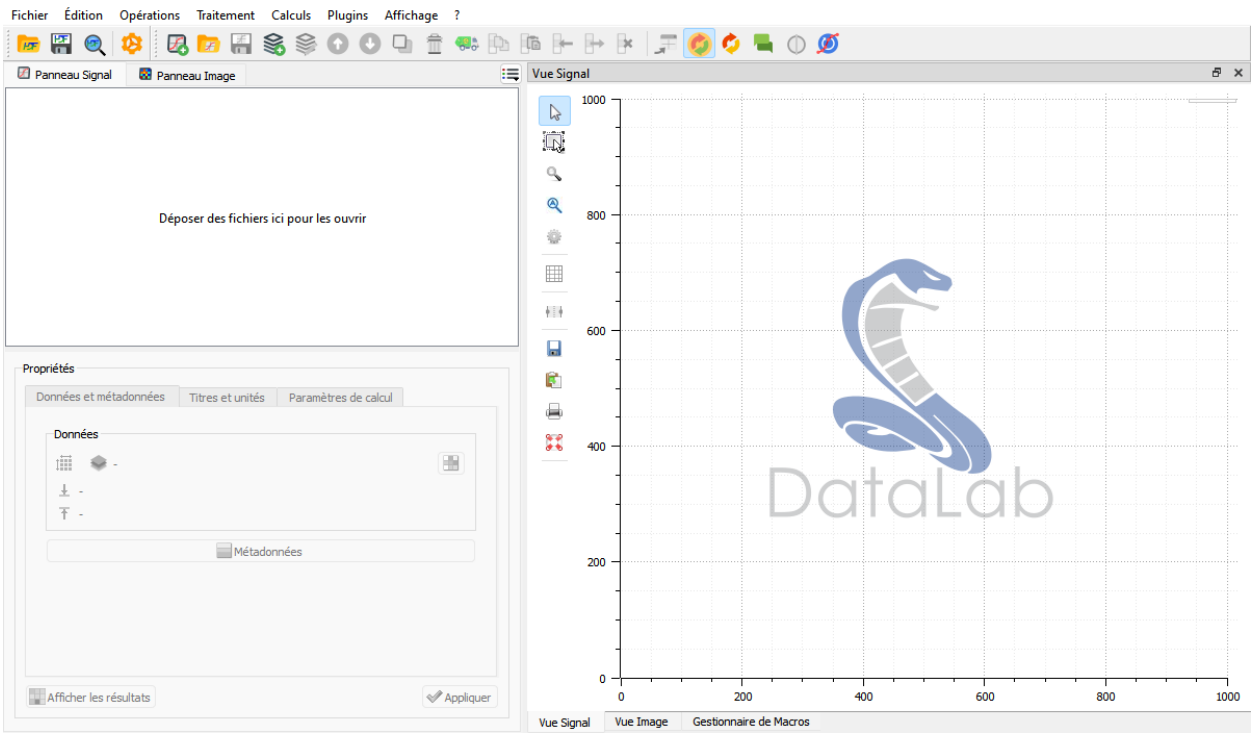


FIG. 2 – Fenêtre principale de DataLab, au démarrage.

affiché dans les panneaux correspond à une feuille de l'arbre. À l'intérieur du jeu de données, les données sont organisées de manière orientée objet, avec un ensemble d'attributs et de méthodes. Le modèle de données est décrit plus en détail dans la section API (voir *cdl.obj*).

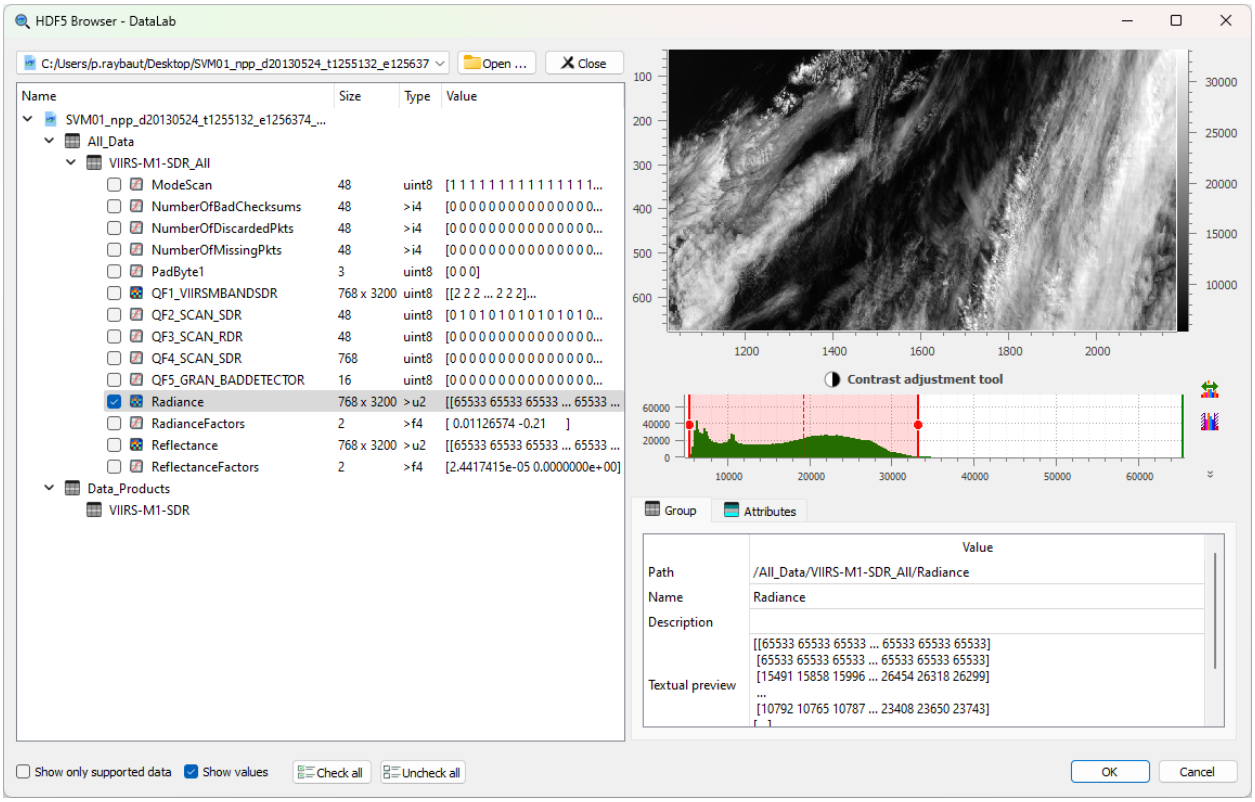
Pour chaque jeu de données (signal 1D ou image 2D), non seulement les données elles-mêmes sont stockées, mais aussi un ensemble de métadonnées, qui décrit les données ou la façon dont elles doivent être affichées. Les métadonnées sont stockées dans un dictionnaire, qui est accessible via l'attribut `metadata` du jeu de données (et peuvent également être parcourues dans la vue **Propriétés**, avec le bouton **Métadonnées**).

L'**Espace de travail** de DataLab est défini comme l'ensemble de tous les jeux de données qui sont actuellement chargés dans DataLab, dans les panneaux **Signaux** et **Images**. L'espace de travail peut être enregistré dans un fichier HDF5, puis rechargé ultérieurement. Il est également possible d'importer des jeux de données à partir d'un fichier HDF5 dans l'espace de travail, en utilisant l'*Explorateur HDF5*.

Note : Les jeux de données peuvent également être enregistrés ou chargés individuellement, en utilisant des formats de données tels que *.txt* ou *.npy* pour les signaux 1D (voir *Ouvrir un signal* pour la liste des formats pris en charge), ou *.tiff* ou *.dcm* pour les images 2D (voir *Ouvrir une image* pour la liste des formats pris en charge).

2.1.2 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).

Le panneau de droite affiche les données de courbe ou d'image sélectionnées. Il affiche également des informations sur le « Groupe » (chemin, description, etc.) et les « Attributs » (noms et valeurs).

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 »

Note : L'explorateur HDF5 peut être utilisé pour explorer plusieurs fichiers HDF5 à la fois, permettant ainsi d'importer des données de différents fichiers dans le même espace de travail de DataLab.

2.1.3 Macros

Généralités

Plusieurs méthodes permettent d'étendre les fonctionnalités de DataLab (voir *Plugins* ou *Contrôle à distance*). La manière la plus simple de le faire est d'utiliser des macros. Les macros sont de petits scripts Python qui peuvent être exécutés depuis le « Gestionnaire de Macros » dans DataLab.

Fig. 3 – Le Gestonnaire de Macros dans DataLab.

Les macros peuvent être utilisées pour automatiser des tâches répétitives, ou pour créer de nouvelles fonctionnalités. Comme le système de plugins et de contrôle à distance, les macros reposent sur l'API de haut niveau de DataLab pour interagir avec l'application. Cela signifie que vous pouvez réutiliser les mêmes extraits de code dans les macros, les plugins et les scripts de contrôle à distance.

Avertissement : DataLab gère les macros comme des scripts Python. Cela signifie que vous pouvez utiliser toute la puissance de Python pour créer vos macros. Même si c'est une fonctionnalité puissante, cela signifie également que vous devez être prudent lorsque vous exécutez des macros à partir de sources inconnues, car elles peuvent potentiellement endommager votre système.

Voir aussi :

L'API de haut niveau de DataLab est documentée dans la section *API*. Le système de plugins est documenté dans la section *Plugins*, et le système de contrôle à distance est documenté dans la section *Contrôle à distance*.

Fonctionnalités principales

Le Gestionnaire de Macros est une interface simple pour :

- Créer de nouvelles macros, en utilisant le bouton « Nouvelle macro » .
- Renommer des macros existantes, en utilisant le bouton « Renommer macro » .
- Importer/exporter des macros depuis/vers des fichiers, en utilisant les boutons « Importer macro » et « Exporter macro » .
- Exécuter des macros, en utilisant le bouton « Exécuter macro » .
- Arrêter l'exécution d'une macro, en utilisant le bouton « Arrêter macro » .

Les macros sont intégrées dans l'espace de travail de DataLab, elles sont donc enregistrées avec le reste des données (c'est-à-dire avec les signaux et les images) lors de l'exportation de l'espace de travail vers un fichier HDF5. Cela signifie que vous pouvez partager vos macros avec d'autres utilisateurs simplement en partageant le fichier d'espace de travail.

Note : Les macros sont exécutées dans un processus séparé, elles ne bloqueront donc pas l'application principale de DataLab. Cela signifie que vous pouvez continuer à travailler avec DataLab pendant qu'une macro est en cours d'exécution et que *vous pouvez arrêter une macro à tout moment* en utilisant le bouton .

Exemple

Pour un exemple détaillé de création d'une macro, voir le tutoriel *Prototypage d'une chaîne de traitement personnalisée*.

2.1.4 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 : Si vous recherchez une solution alternative légère pour contrôler DataLab à distance (c'est-à-dire sans avoir à installer l'ensemble du package DataLab et ses dépendances sur votre environnement), veuillez consulter le package DataLab Simple Client (*pip install cdlclient*).

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.

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.

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`

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)

```

(suite sur la page suivante)

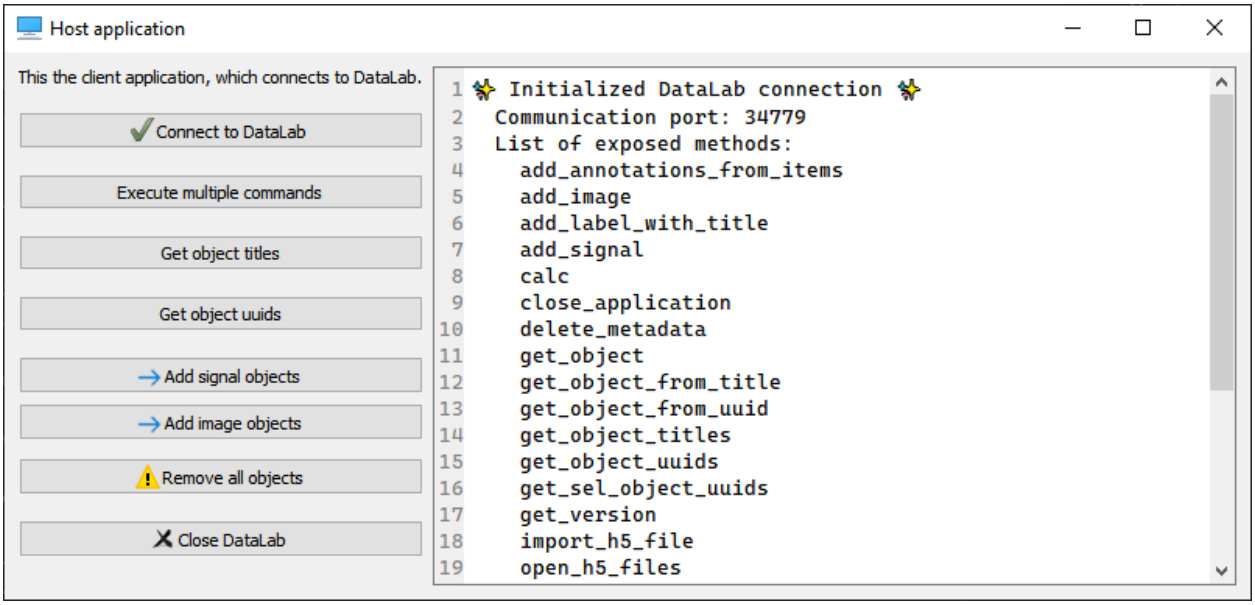


FIG. 4 – Capture d'écran du test de l'application cliente distante (`cdl.tests.remoteclient_app`)

(suite de la page précédente)

```
Created on Fri May 12 12:28:56 2023

@author: p.raybaut
"""

# %% Importing necessary modules

# NumPy for numerical array computations:
import numpy as np

# DataLab remote control client:
from cdl.proxy import RemoteProxy

# %% Connecting to DataLab current session

proxy = RemoteProxy()

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

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 (...)
```

(suite sur la page suivante)

(suite de la page précédente)

```
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


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()
```

(suite sur la page suivante)

(suite de la page précédente)

```
    ini.read(fname)
    try:
        return ini.get("main", "rpc_server_port")
    except (cp.NoSectionError, cp.NoOptionError):
        raise ConnectionRefusedError("DataLab has not yet been executed")


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

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

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

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

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

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

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

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

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

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

(suite sur la page suivante)

(suite de la page précédente)

```
        self.serverproxy.toggle_auto_refresh(state)

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

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

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

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

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

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

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

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

    def get_object(self, nb_id_title=None, panel=None):
```

(suite sur la page suivante)

(suite de la page précédente)

```
        """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()
```

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`.

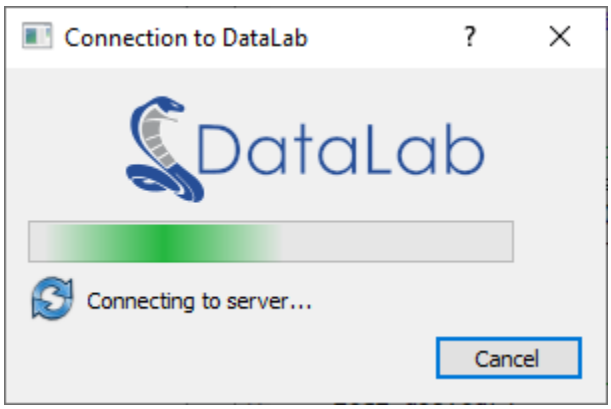


FIG. 5 – Capture d'écran de la boîte de dialogue de connexion (`cdl.widgets.connection.ConnectionDialog`)

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
"""
```

(suite sur la page suivante)

(suite de la page précédente)

```
# guitest: show,skip

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

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


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


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

API publique : client distant

class cdl.core.remote.**RemoteClient**

Objet représentant un proxy/client vers le serveur XML-RPC de DataLab. Cet objet est utilisé pour appeler les fonctions de DataLab à partir d'un script Python.

Exemples

Voici un exemple simple de l'utilisation de RemoteClient dans un script Python ou dans un notebook Jupyter :

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

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

Tentative de connexion au serveur XML-RPC de DataLab.

Paramètres

- **port** (*str | None*) – Port XML-RPC auquel se connecter. Si non spécifié, le port est automatiquement récupéré à partir de la configuration de DataLab.
- **timeout** (*float | None*) – Délai d'attente en secondes. Par défaut, 5.0.
- **retries** (*int | None*) – Nombre de tentatives. Par défaut, 10.

Lève

- **ConnectionRefusedError** – Impossible de se connecter à DataLab
- **ValueError** – Délai d'attente non valide (doit être >= 0.0)
- **ValueError** – Nombre de tentatives non valide (doit être >= 1)

disconnect() → None

Déconnexion du serveur XML-RPC de DataLab.

is_connected() → bool

Renvoie True si connecté au serveur XML-RPC de DataLab.

get_method_list() → list[str]

Renvoie la liste des méthodes disponibles.

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

Ajouter des données de signal à DataLab.

Paramètres

- **title** (*str*) – Titre du signal
- **xdata** (*numpy.ndarray*) – Données X
- **ydata** (*numpy.ndarray*) – Données Y
- **xunit** (*str | None*) – Unité X. Par défaut, None.
- **yunit** (*str | None*) – Unité Y. Par défaut, None.
- **xlabel** (*str | None*) – Libellé X. Par défaut, None.
- **ylabel** (*str | None*) – Libellé Y. Par défaut, None.

Renvoie

True si le signal a été ajouté avec succès, False sinon

Type renvoyé

bool

Lève

- **ValueError** – Type de données xdata non valide
- **ValueError** – Type de données ydata non valide

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

Ajouter des données d'image à DataLab.

Paramètres

- **title** (*str*) – Titre de l'image
- **data** (*numpy.ndarray*) – Données de l'image
- **xunit** (*str | None*) – Unité X. Par défaut, None.
- **yunit** (*str | None*) – Unité Y. Par défaut, None.
- **zunit** (*str | None*) – Unité Z. Par défaut, None.
- **xlabel** (*str | None*) – Libellé X. Par défaut, None.
- **ylabel** (*str | None*) – Libellé Y. Par défaut, None.
- **zlabel** (*str | None*) – Libellé Z. Par défaut, None.

Renvoie

True si l'image a été ajoutée avec succès, False sinon

Type renvoyé
bool

Lève
ValueError – Type de données non valide

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

Appeler la fonction de calcul **name** dans le processeur du panneau en cours.

Paramètres
— **name** (*str*) – Nom de la fonction de calcul
— **param** (*guidata.dataset.DataSet | None*) – Paramètre de la fonction de calcul. Par défaut, None.

Renvoie
Résultat de la fonction de calcul

Type renvoyé
guidata.dataset.DataSet

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

Obtenir un objet (signal/image) à partir de l'index.

Paramètres
— **nb_id_title** – Numéro d'objet, ou identifiant d'objet, ou titre d'objet. Par défaut, None (objet en cours).
— **panel** – Nom du panneau. Par défaut, None (panneau en cours).

Renvoie
Objet

Lève
KeyError – si l'objet n'est pas trouvé

get_object_shapes(*nb_id_title : int | str | None = None*, *panel : str | None = None*) → list

Obtenir les formes géométriques des éléments de tracé associés à l'objet (signal/image).

Paramètres
— **nb_id_title** – Numéro d'objet, ou identifiant d'objet, ou titre d'objet. Par défaut, None (objet en cours).
— **panel** – Nom du panneau. Par défaut, None (panneau en cours).

Renvoie
Liste des formes géométriques des éléments de tracé

add_annotations_from_items(*items : list*, *refresh_plot : bool = True*, *panel : str | None = None*) → None

Ajouter des annotations d'objet (éléments de tracé d'annotation).

Paramètres
— **items** (*list*) – éléments de tracé d'annotation
— **refresh_plot** (*bool | None*) – rafraîchir le tracé. Par défaut, True.
— **panel** (*str | None*) – nom du panneau (valeurs valides : « signal », « image »). Si None, le panneau en cours est utilisé.

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

Ajouter un objet à DataLab.

Paramètres
obj (SignalObj | ImageObj) – Objet signal ou image

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

Ajouter une étiquette avec le titre de l'objet sur le tracé associé

Paramètres
— **title** (*str | None*) – Titre de l'étiquette. Par défaut, None. Si None, le titre est le titre de l'objet.

— **panel** (*str | None*) – nom du panneau (valeurs valides : « signal », « image »). Si None, le panneau en cours est utilisé.

close_application() → None

Fermer l'application DataLab

context_no_refresh() → Generator[None, None, None]

Renvoie un gestionnaire de contexte pour désactiver temporairement le rafraîchissement automatique.

Renvoie
Gestionnaire de contexte

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*, *keep_roi : bool = False*) → None

Supprimer les métadonnées des objets sélectionnés

Paramètres
— **refresh_plot** – Rafraîchir le tracé. Par défaut, True.
— **keep_roi** – Conserver la ROI. Par défaut, False.

get_current_panel() → str

Renvoie le nom du panneau en cours.

Renvoie
Nom du panneau (valeurs valides : « signal », « image », « macro »))

Type renvoyé
str

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

Renvoie les titres des groupes et les listes des uuids et titres des objets internes.

Renvoie
titres des groupes, listes des uuids et titres des objets internes

Type renvoyé
Tuple

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

Obtenir la liste des objets (signal/image) pour le panneau en cours. Les objets sont triés par numéro de groupe et index d'objet dans le groupe.

Paramètres
panel – nom du panneau (valeurs valides : « signal », « image », « macro »). Si None, le panneau de données en cours est utilisé (c'est-à-dire le panneau de signal ou d'image).

Renvoie
Liste des titres des objets

Lève
ValueError – si le panneau n'est pas trouvé

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

Obtenir la liste des uuids des objets (signal/image) pour le panneau en cours. Les objets sont triés par numéro de groupe et index d'objet dans le groupe.

Paramètres
panel (*str | None*) – nom du panneau (valeurs valides : « signal », « image »). Si None, le panneau en cours est utilisé.

Renvoie
liste des uuids des objets

Type renvoyé
list[str]

Lève
ValueError – si le panneau n'est pas trouvé

classmethod get_public_methods() → list[str]

Renvoie toutes les méthodes publiques de la classe, sauf elle-même.

Renvoie
Liste des méthodes publiques

Type renvoyé
list[str]

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

Renvoie les uuids des objets sélectionnés.

Paramètres
include_groups – Si True, renvoie également les objets des groupes sélectionnés.

Renvoie
Liste des uuids des objets sélectionnés.

get_version() → str

Renvoie la version de DataLab.

Renvoie
Version de DataLab

Type renvoyé
str

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

Ouvrir le navigateur HDF5 de DataLab pour importer un fichier HDF5.

Paramètres
— **filename** (*str*) – Nom du fichier HDF5
— **reset_all** (*bool | None*) – Réinitialiser toutes les données de l'application. Par défaut, None.

import_macro_from_file(*filename : str*) → None

Importer une macro à partir d'un fichier

Paramètres
filename – Nom du fichier.

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

Ouvrir un fichier HDF5 de DataLab ou importer à partir de tout autre fichier HDF5.

Paramètres
— **h5files** (*list[str] | None*) – Liste des fichiers HDF5 à ouvrir. Par défaut, None.
— **import_all** (*bool | None*) – Importer tous les objets à partir des fichiers HDF5. Par défaut, None.

— **reset_all** (*bool | None*) – Réinitialiser toutes les données de l'application. Par défaut, None.

open_object(*filename : str*) → None

Ouvrir un objet à partir d'un fichier dans le panneau en cours (signal/image).

Paramètres
filename (*str*) – Nom du fichier

raise_window() → None

Faire apparaître la fenêtre de DataLab

reset_all() → None

Réinitialiser toutes les données de l'application

run_macro(*number_or_title : int | str | None = None*) → None

Exécute la macro.

Paramètres
number_or_title – Numéro de macro, ou titre de macro. Par défaut, None (macro en cours).

Lève
ValueError – si la macro n'est pas trouvée

save_to_h5_file(*filename : str*) → None

Enregistrer dans un fichier HDF5 de DataLab.

Paramètres
filename (*str*) – Nom du fichier HDF5

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

Sélectionner des groupes dans le panneau en cours.

Paramètres
— **selection** – Liste des numéros de groupe (1 à N), ou liste des uuids de groupe, ou None pour sélectionner tous les groupes. Par défaut, None.
— **panel** (*str | None*) – nom du panneau (valeurs valides : « signal », « image »). Si None, le panneau en cours est utilisé. Par défaut, None.

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

Sélectionner des objets dans le panneau en cours.

Paramètres
— **selection** – Liste des numéros d'objet (1 à N) ou uuids à sélectionner
— **panel** – nom du panneau (valeurs valides : « signal », « image »). Si None, le panneau en cours est utilisé. Par défaut, None.

set_current_panel(*panel : str*) → None

Basculer vers le panneau.

Paramètres
panel (*str*) – Nom du panneau (valeurs valides : « signal », « image », « macro »))

stop_macro(*number_or_title : int | str | None = None*) → None

Arrête la macro.

Paramètres
number_or_title – Numéro de macro, ou titre de macro. Par défaut, None (macro en cours).

Lève
ValueError – si la macro n'est pas trouvée

toggle_auto_refresh(*state : bool*) → None

Basculer l'état du rafraîchissement automatique.

Paramètres

state (*bool*) – État du rafraîchissement automatique

toggle_show_titles(*state : bool*) → None

Basculer l'état d'affichage des titres.

Paramètres

state (*bool*) – État d'affichage des titres

API publique : méthodes supplémentaires

Les méthodes de la classe de contrôle à distance (en utilisant le proxy ou le client distant) peuvent être complétées par des méthodes supplémentaires qui sont ajoutées dynamiquement à l'exécution. Ce mécanisme permet d'accéder aux méthodes des objets « processeur » de DataLab.

Processeur de signal

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

Objet de traitement de signal : opérations, traitement, calcul

compute_sum() → None

Calculer la somme

compute_average() → None

Calculer la moyenne

compute_product() → None

Calculer le produit

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

Extraire une région d'intérêt (ROI) des données

compute_swap_axes() → None

Permuter les axes des données

compute_abs() → None

Calculer la valeur absolue

compute_re() → None

Calculer la partie réelle

compute_im() → None

Calculer la partie imaginaire

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

Convertir le type de données

compute_log10() → None

Calculer Log10

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

Calculer la différence entre deux signaux

compute_quadratic_difference(*obj2 :* SignalObj | *None* = *None*) → None
Calculer la différence quadratique entre deux signaux

compute_division(*obj2 :* SignalObj | *None* = *None*) → None
Calculer la division entre deux signaux

compute_peak_detection(*param :* PeakDetectionParam | *None* = *None*) → None
Détecter les pics à partir des données

compute_normalize(*param :* NormalizeYParam | *None* = *None*) → None
Normaliser les données

compute_derivative() → None
Calculer la dérivée

compute_integral() → None
Calculer l'intégrale

compute_calibration(*param :* XYCalibrateParam | *None* = *None*) → None
Calculer l'étalonnage linéaire des données

compute_threshold(*param :* ThresholdParam | *None* = *None*) → None
Calculer le seuillage des données

compute_clip(*param :* ClipParam | *None* = *None*) → None
Calculer le seuillage maximum des données

compute_gaussian_filter(*param :* GaussianParam | *None* = *None*) → None
Calculer le filtre gaussien

compute_moving_average(*param :* MovingAverageParam | *None* = *None*) → None
Calculer la moyenne mobile

compute_moving_median(*param :* MovingMedianParam | *None* = *None*) → None
Calculer la médiane mobile

compute_wiener() → None
Calculer le filtre de Wiener

compute_fft(*param :* FFTParam | *None* = *None*) → None
Calculer l'iFFT

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

compute_interpolation(*obj2 :* SignalObj | *None* = *None*, *param :* InterpolationParam | *None* = *None*)
Calculer l'interpolation

compute_resampling(*param :* ResamplingParam | *None* = *None*)
Calculer le rééchantillonnage

compute_detrending(*param :* DetrendingParam | *None* = *None*)
Calculer l'élimination de la tendance

compute_convolution(*obj2 :* SignalObj | *None* = *None*) → None
Calculer la convolution

compute_fit(*name*, *fitdlgfunc*)
Calculer la courbe de régression

compute_polyfit(*param :* PolynomialFitParam | *None* = *None*) → None

Calculer la courbe de régression polynomiale

compute_multigaussianfit() → None

Calculer la courbe de régression multi-Gaussienne

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

Calculer la LMH

compute_fw1e2() → dict[str, *ResultShape*]

Calculer la largeur à $1/e^2$

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

Calcule l'intégrale

Processeur d'image

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

Objet de traitement d'image : opérations, traitement, calcul

compute_sum() → None

Calculer la somme

compute_average() → None

Calculer la moyenne

compute_product() → None

Calculer le produit

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

Calculer le logarithme en base 10

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

Rotation arbitraire des données

compute_rotate90() → None

Rotation des données de 90°

compute_rotate270() → None

Rotation des données de 270°

compute_fliph() → None

Retournement horizontal des données

compute_flipv() → None

Retournement vertical des données

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

Distribuer les images sur une grille

reset_positions() → None

Réinitialiser les positions des images

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

Redimensionner l'image

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

compute_roi_extraction(*param :* ROIDataParam | *None* = *None*) → None
Extraire une région d'intérêt (ROI) des données

compute_profile(*param :* ProfileParam | *None* = *None*) → None
Calculer le profil

compute_average_profile(*param :* AverageProfileParam | *None* = *None*) → None
Calculer le profil moyen

compute_radial_profile(*param :* RadialProfileParam | *None* = *None*) → None
Calculer le profil radial

compute_histogram(*param :* HistogramParam | *None* = *None*) → None
Calcule l'intégrale

compute_swap_axes() → None
Permuter les axes des données

compute_abs() → None
Calculer la valeur absolue

compute_re() → None
Calculer la partie réelle

compute_im() → None
Calculer la partie imaginaire

compute_astype(*param :* DataTypeIParam | *None* = *None*) → None
Convertir le type de données

compute_log10() → None
Calculer Log10

compute_difference(*obj2 :* ImageObj | *None* = *None*) → None
Calculer la différence entre deux images

compute_quadratic_difference(*obj2 :* ImageObj | *None* = *None*) → None
Calculer la différence quadratique entre deux images

compute_division(*obj2 :* ImageObj | *None* = *None*) → None
Calculer la division entre deux images

compute_flatfield(*obj2 :* ImageObj | *None* = *None*, *param :* FlatFieldParam | *None* = *None*) → None
Calculer la correction de champ plat

compute_calibration(*param :* ZCalibrateParam | *None* = *None*) → None
Calculer l'étalonnage linéaire des données

compute_threshold(*param :* ThresholdParam | *None* = *None*) → None
Calculer le seuillage des données

compute_clip(*param :* ClipParam | *None* = *None*) → None
Calculer le seuillage maximum des données

compute_gaussian_filter(*param :* GaussianParam | *None* = *None*) → None
Calculer le filtre gaussien

compute_moving_average(*param :* MovingAverageParam | *None* = *None*) → None
Calculer la moyenne mobile

compute_moving_median(*param :* MovingMedianParam | *None* = *None*) → None
Calculer la médiane mobile

compute_wiener() → None
Calculer le filtre de Wiener

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

compute_ifft(*param :* FFTParam | *None* = *None*) → None
Calculer l'iFFT

compute_butterworth(*param :* ButterworthParam | *None* = *None*) → None
Calculer le filtre de Butterworth

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

compute_adjust_log(*param :* AdjustLogParam | *None* = *None*) → None
Calculer la correction logarithmique

compute_adjust_sigmoid(*param :* AdjustSigmoidParam | *None* = *None*) → None
Calculer la correction sigmoïde

compute_rescale_intensity(*param :* RescaleIntensityParam | *None* = *None*) → None
Rééchelonner les niveaux d'intensité de l'image

compute_equalize_hist(*param :* EqualizeHistParam | *None* = *None*) → None
Égalisation d'histogramme

compute_equalize_adapthist(*param :* EqualizeAdaptHistParam | *None* = *None*) → None
Égalisation d'histogramme adaptative

compute_denoise_tv(*param :* DenoiseTVParam | *None* = *None*) → None
Calculer le débruitage par variation totale

compute_denoise_bilateral(*param :* DenoiseBilateralParam | *None* = *None*) → None
Calculer le débruitage par filtre bilatéral

compute_denoise_wavelet(*param :* DenoiseWaveletParam | *None* = *None*) → None
Calculer le débruitage par ondelettes

compute_denoise_tophat(*param :* MorphologyParam | *None* = *None*) → None
Débruitage par White Top-Hat

compute_all_denoise(*params : list* | *None* = *None*) → None
Calculer tous les filtres de débruitage

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

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

compute_erosion(*param :* MorphologyParam | *None* = *None*) → None
Calculer l'érosion

compute_dilation(*param :* MorphologyParam | *None* = *None*) → None
Calculer la dilatation

compute_opening(*param :* MorphologyParam | *None* = *None*) → None
Calculer l'ouverture morphologique

compute_closing(*param :* MorphologyParam | *None* = *None*) → None
Calculer la fermeture morphologique

compute_all_morphology(*param :* MorphologyParam | *None* = *None*) → None
Calculer tous les filtres de morphologie

compute_canny(*param :* CannyParam | *None* = *None*) → None
Calculer le filtre de Canny

compute_roberts() → None
Calculer le filtre de Roberts

compute_prewitt() → None
Calculer le filtre de Prewitt

compute_prewitt_h() → None
Calculer le filtre de Prewitt (horizontal)

compute_prewitt_v() → None
Calculer le filtre de Prewitt (vertical)

compute_sobel() → None
Calculer le filtre de Sobel

compute_sobel_h() → None
Calculer le filtre de Sobel (horizontal)

compute_sobel_v() → None
Calculer le filtre de Sobel (vertical)

compute_scharr() → None
Calculer le filtre de Scharr

compute_scharr_h() → None
Calculer le filtre de Scharr (horizontal)

compute_scharr_v() → None
Calculer le filtre de Scharr (vertical)

compute_farid() → None
Calculer le filtre de Farid

compute_farid_h() → None
Calculer le filtre de Farid (horizontal)

compute_farid_v() → None
Calculer le filtre de Farid (vertical)

compute_laplace() → None
Calculer le filtre de Laplace

compute_all_edges() → None
Calculer tous les contours

compute_centroid() → dict[str, *ResultShape*]

Calculer le centre de gravité de l'image

compute_enclosing_circle() → dict[str, *ResultShape*]

Calculer le cercle d'encadrement minimum

compute_peak_detection(*param :* Peak2DDetectionParam | *None* = *None*) → dict[str, *ResultShape*]

Calculer la détection de pics 2D

compute_contour_shape(*param :* ContourShapeParam | *None* = *None*) → dict[str, *ResultShape*]

Calculer l'ajustement de la forme du contour

compute_hough_circle_peaks(*param :* HoughCircleParam | *None* = *None*) → dict[str, *ResultShape*]

Calculer la détection de pics basée sur une transformée de Hough circulaire

compute_blob_dog(*param :* BlobDOGParam | *None* = *None*) → dict[str, *ResultShape*]

Calculer la détection de taches en utilisant la méthode de différence de Gaussienne

compute_blob_doh(*param :* BlobDOHParam | *None* = *None*) → dict[str, *ResultShape*]

Calculer la détection de taches en utilisant la méthode du déterminant de Hessian

compute_blob_log(*param :* BlobLOGParam | *None* = *None*) → dict[str, *ResultShape*]

Calculer la détection de taches en utilisant la méthode du Laplacien de Gaussienne

compute_blob_opencv(*param :* BlobOpenCVParam | *None* = *None*) → dict[str, *ResultShape*]

Calculer la détection de taches en utilisant OpenCV

2.1.5 Modèle de données interne

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

- *cdl.obj.SignalObj*, qui représente un objet signal, et
- *cdl.obj.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*.

Voir aussi :

La section *API* pour plus d'informations sur l'API publique.

2.1.6 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.

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 *cdl.plugins.PluginBase*. Le nom de la classe n'est pas important, tant qu'elle est dérivée de *cdl.plugins.PluginBase* et qu'elle dispose d'un attribut `PLUGIN_INFO` qui est une instance de la classe *cdl.plugins.PluginInfo*. L'attribut `PLUGIN_INFO` est utilisé par DataLab pour récupérer des informations sur le plugin.

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).

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(
```

(suite sur la page suivante)

(suite de la page précédente)

```
    src: dlo.SignalObj, p: test_data.GaussianNoiseParam
) -> dlo.SignalObj:
    """Add gaussian noise to signal"""
    dst = cpsig.dst_11(src, "add_gaussian_noise", f"mu={p.mu},sigma={p.sigma}")
    test_data.add_gaussian_noise_to_signal(dst, p)
    return dst


def add_noise_to_image(src: dlo.ImageObj, p: dlo.NormalRandomParam) -> dlo.ImageObj:
    """Add gaussian noise to image"""
    dst = cpima.dst_11(src, "add_gaussian_noise", f"mu={p.mu},sigma={p.sigma}")
    test_data.add_gaussian_noise_to_image(dst, p)
    return dst


class PluginTestData(PluginBase):
    """DataLab Test Data Plugin"""

    PLUGIN_INFO = PluginInfo(
        name=_("Test data"),
        version="1.0.0",
        description=_("Testing DataLab functionalities"),
    )

    # Signal processing features ------------------------------------------------
    def add_noise_to_signal(self) -> None:
        """Add noise to signal"""
        self.signalpanel.processor.compute_11(
            add_noise_to_signal,
            paramclass=test_data.GaussianNoiseParam,
            title=_("Add noise"),
        )

    def create_paracetamol_signal(self) -> None:
        """Create paracetamol signal"""
        obj = test_data.create_paracetamol_signal()
        self.proxy.add_object(obj)

    def create_noisy_signal(self) -> None:
        """Create noisy signal"""
        obj = self.signalpanel.new_object(add_to_panel=False)
        if obj is not None:
            noiseparam = test_data.GaussianNoiseParam(_("Noise"))
            self.signalpanel.processor.update_param_defaults(noiseparam)
            if noiseparam.edit(self.signalpanel):
                test_data.add_gaussian_noise_to_signal(obj, noiseparam)
                self.proxy.add_object(obj)

    # Image processing features ------------------------------------------------
    def add_noise_to_image(self) -> None:
        """Add noise to image"""
        self.imagepanel.processor.compute_11(
```

(suite sur la page suivante)

(suite de la page précédente)

```
            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)
        if obj is not None:
            param = test_data.PeakDataParam.create(size=max(obj.data.shape))
            self.imagepanel.processor.update_param_defaults(param)
            if param.edit(self.imagepanel):
                obj.data = test_data.get_peak2d_data(param)
                self.proxy.add_object(obj)

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

    def create_noisygauss_image(self) -> None:
        """Create 2D noisy gauss image"""
        newparam = self.edit_new_image_parameters(hide_image_type=True)
        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.edit_new_image_parameters(hide_image_type=True)
        if newparam is not None:
            obj = test_data.create_multigauss_image(newparam)
            self.proxy.add_object(obj)

    def create_2dstep_image(self) -> None:
        """Create 2D step image"""
        newparam = self.edit_new_image_parameters(hide_image_type=True)
        if newparam is not None:
            obj = test_data.create_2dstep_image(newparam)
            self.proxy.add_object(obj)

    def create_ring_image(self) -> None:
        """Create 2D ring image"""
        param = test_data.RingParam(_("Ring"))
        if param.edit(self.imagepanel):
            obj = test_data.create_ring_image(param)
            self.proxy.add_object(obj)

    def create_annotated_image(self) -> None:
        """Create annotated image"""
```

(suite sur la page suivante)

(suite de la page précédente)

```
        obj = test_data.create_annotated_image()
        self.proxy.add_object(obj)

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

(suite sur la page suivante)

(suite de la page précédente)

```
            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",
            )
```

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:
```

(suite sur la page suivante)

(suite de la page précédente)

```
        self.__debug = debug
        self.file_format = None  # long
        self.nbcols = None  # long
        self.nbrows = None  # long
        self.nbframes = None  # long
        self.pixeltype = None  # long
        self.quantlevels = None  # long
        self.maxlevel = None  # float
        self.minlevel = None  # float
        self.comment_length = None  # long
        self.fname = None
        self.data = None
        if fname is not None:
            self.load(fname)

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

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

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

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

(suite sur la page suivante)

(suite de la page précédente)

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


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

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

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

        Args:
            filename (str): path to FXD file

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


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


class XYZImageFormat(ImageFormatBase):
```

(suite sur la page suivante)

(suite de la page précédente)

```
    """Object representing Dürr NDT XYZ image file type"""

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

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

        Args:
            filename (str): path to XYZ file

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

    @staticmethod
    def write_data(filename: str, data: np.ndarray) -> None:
        """Write data to file

        Args:
            filename: File name
            data: Image array data
        """
        data = np.fliplr(data)
        with open(filename, "wb") as fdesc:
            fdesc.write(np.array(data.shape[1], dtype=np.uint16).tobytes())
            fdesc.write(np.array(data.shape[0], dtype=np.uint16).tobytes())
            fdesc.write(data.tobytes())
```

Autres exemples

D'autres exemples de plugins peuvent être trouvés dans le répertoire *plugins/examples* du code source de DataLab (explorez ici sur GitHub).

API publique

Système de plugins de DataLab

Le système de plugins de DataLab fournit un moyen d'étendre l'application avec de nouvelles fonctionnalités.

Les plugins sont des modules Python qui reposent sur deux classes :
- *PluginInfo*, qui stocke des informations sur le plugin
- *PluginBase*, qui est la classe de base pour tous les plugins

Les plugins peuvent également étendre les fonctionnalités d'entrée/sortie de DataLab en fournissant de nouveaux formats d'image ou de signal. Pour ce faire, ils doivent fournir une sous-classe de `ImageFormatBase` ou `SignalFormatBase`, dans laquelle les informations de format sont définies à l'aide de la classe `FormatInfo`.

class cdl.plugins.**PluginRegistry**(*name*, *bases*, *attrs*)

Métaclasse pour l'enregistrement des plugins

classmethod get_plugin_classes() → list[type[*PluginBase*]]

Retourne les classes de plugins

classmethod get_plugins() → list[*PluginBase*]

Retourne les instances de plugins

classmethod get_plugin(*name_or_class : str | type[*PluginBase*]*) → *PluginBase* | None

Retourne l'instance de plugin

classmethod register_plugin(*plugin :* PluginBase)

Enregistrer le plugin

classmethod unregister_plugin(*plugin :* PluginBase)

Désenregistrer le plugin

classmethod get_plugin_infos() → str

Retourne les informations sur les plugins (noms, versions, descriptions) au format html

class cdl.plugins.**PluginInfo**(*name : str = None*, *version : str = '0.0.0'*, *description : str = ''*, *icon : str = None*)

Informations sur le plugin

class cdl.plugins.**PluginBaseMeta**(*name*, *bases*, *namespace*, */*, ***kwargs*)

Métaclasse mixte pour éviter les conflits

class cdl.plugins.**PluginBase**

Classe de base du plugin

property signalpanel: SignalPanel

Retourne le panneau de signal

property imagepanel: ImagePanel

Retourne le panneau d'image

show_warning(*message : str*)

Afficher un message d'avertissement

show_error(*message : str*)

Afficher un message d'erreur

show_info(*message : str*)

Afficher un message d'information

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

Poser une question oui/non

edit_new_signal_parameters(*title : str | None = None*, *size : int | None = None*, *hide_signal_type : bool = True*) → *NewSignalParam*

Créer et éditer un nouveau jeu de paramètres de signal

Paramètres

- **title** – titre du nouveau signal
- **size** – taille du nouveau signal (par défaut : None, obtenue à partir du signal actuel)
- **hide_signal_type** – masquer le paramètre de type de signal (par défaut : True)

Renvoie

Nouveau jeu de paramètres de signal (ou None si annulé)

edit_new_image_parameters(*title : str | None = None*, *shape : tuple[int, int] | None = None*, *hide_image_type : bool = True*, *hide_image_dtype : bool = False*) → *NewImageParam* | None

Créer et éditer un nouveau jeu de paramètres d'image

Paramètres

- **title** – titre de la nouvelle image
- **shape** – dimensions de la nouvelle image (par défaut : None, obtenues à partir de l'image actuelle)
- **hide_image_type** – masquer le paramètre de type d'image (par défaut : True)
- **hide_image_dtype** – masquer le paramètre de type de données d'image (par défaut : False)

Renvoie

Nouveau jeu de paramètres d'image (ou None si annulé)

is_registered()

Retourne True si le plugin est enregistré

register(*main : main.CDLMainWindow*) → None

Enregistrer le plugin

unregister()

Désenregistrer le plugin

register_hooks()

Enregistrer les hooks du plugin

unregister_hooks()

Désenregistrer les hooks du plugin

abstract create_actions()

Créer des actions

cdl.plugins.**discover_plugins**() → list[type[*PluginBase*]]

Découvrir les plugins en utilisant la convention de nommage

Renvoie

Liste des plugins découverts (en tant que classes)

cdl.plugins.**get_available_plugins**() → list[*PluginBase*]

Instancier et obtenir les plugins disponibles

Renvoie

Liste des plugins disponibles (en tant qu'instances)

2.1.7 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)

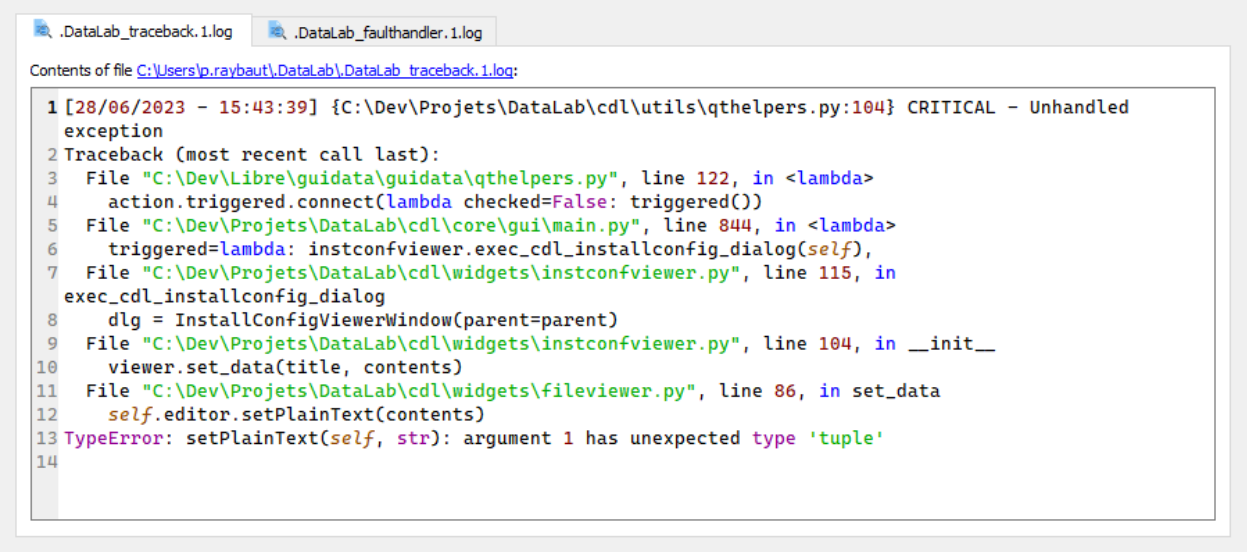


FIG. 6 – Journaux de bord DataLab (voir le menu « ? »)

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*).

2.1.8 Installation et configuration

En raison des multiples façons d'installer DataLab sur votre machine, comprendre l'origine d'un dysfontionnement 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*).

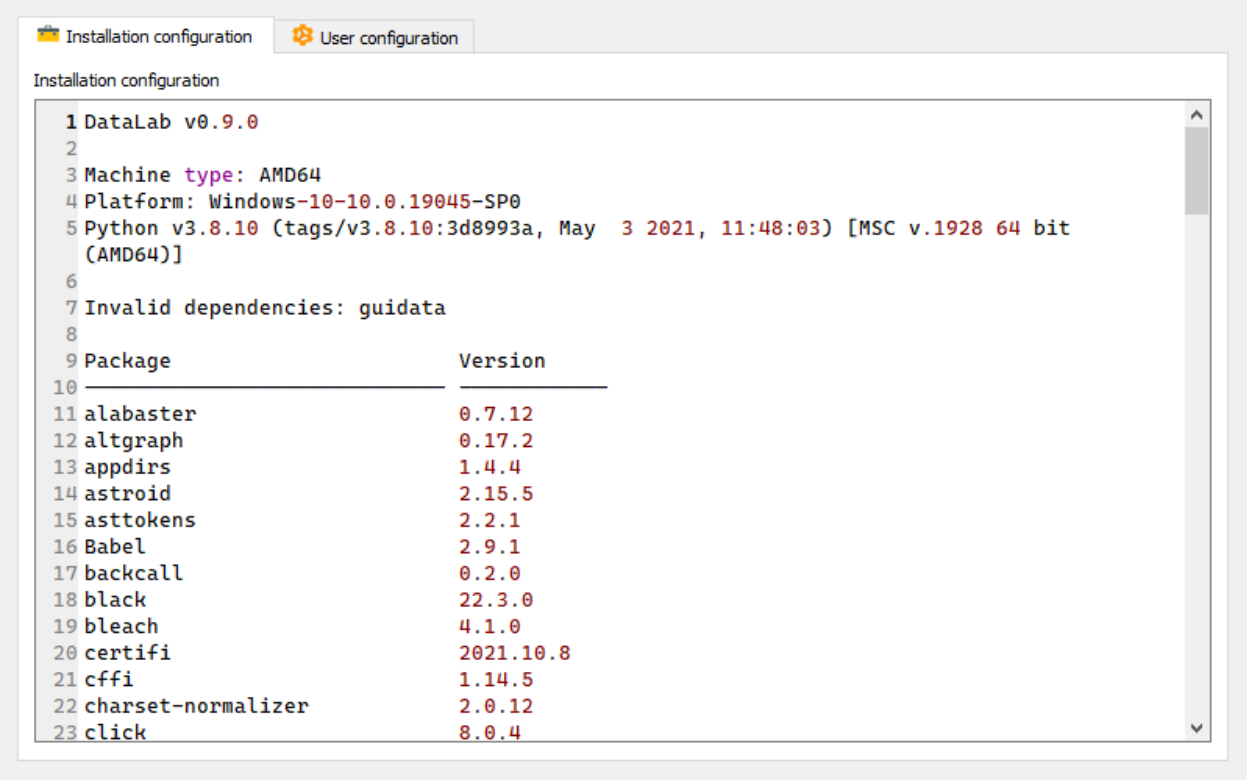


FIG. 7 – Installation et configuration (voir menu « ? »)

2.1.9 Ligne de commande

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
```

(suite sur la page suivante)

(suite de la page précédente)

```
                        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
```

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
```

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
```

Mode démonstration de DataLab

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

```
$ cdl-demo
```

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).

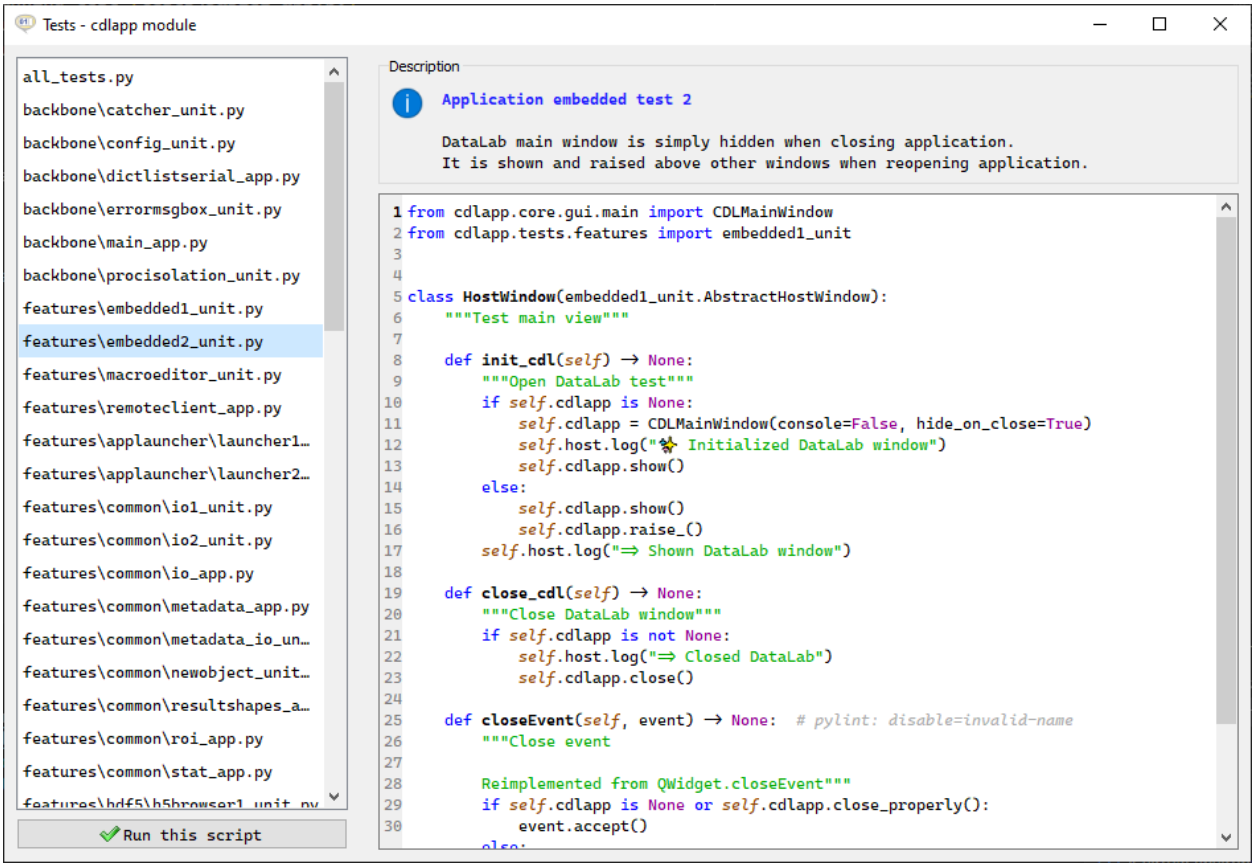
Pour exécuter l'ensemble des tests unitaires de DataLab, taper la commande suivante :

```
$ pytest
```

Exécuter les tests interactifs

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

```
$ cdl-tests
```



2.2 Traitement du signal

Cette section décrit les fonctionnalités spécifiques au panneau de traitement du signal. Le panneau de traitement du signal est le panneau par défaut lorsque DataLab est démarré.

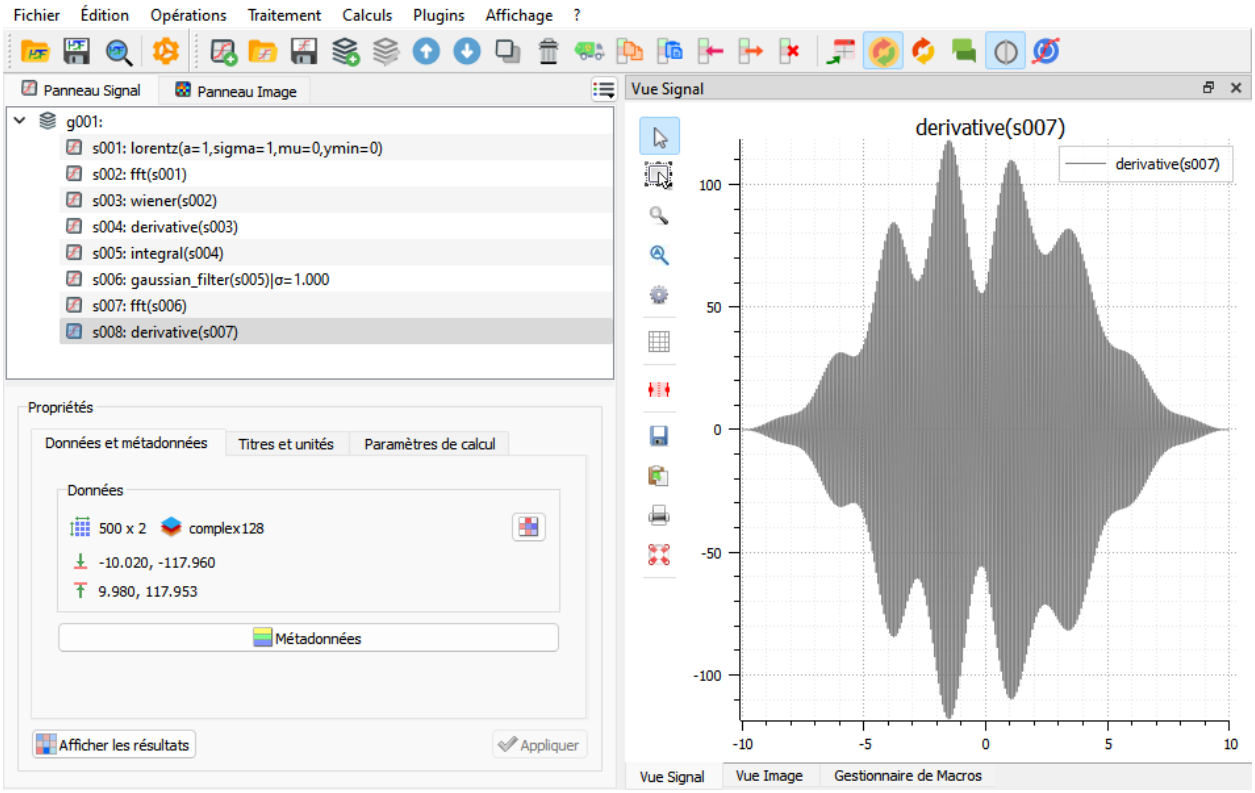


FIG. 8 – Fenêtre principale de DataLab : vue du traitement du signal

2.2.1 Menus

Cette section décrit les fonctionnalités liées aux signaux de DataLab, en présentant les différents menus et leurs entrées.

Menu « Fichier »

Le menu « Fichier » permet de créer, ouvrir, enregistrer et fermer des signaux. Il permet également d'importer et d'exporter des données depuis/vers des fichiers HDF5, et d'éditer les paramètres de la session courante.

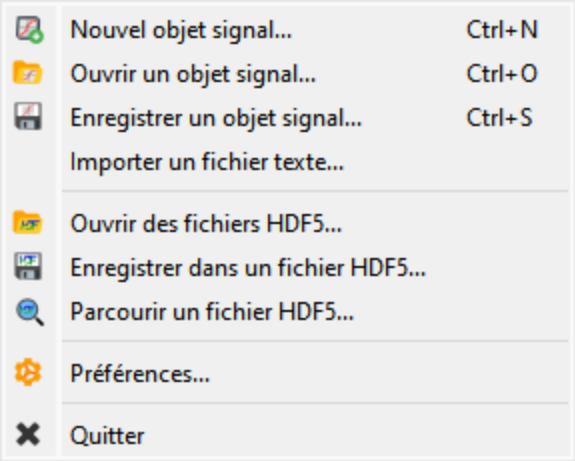


FIG. 9 – Capture d'écran du menu « Fichier ».

Nouveau signal

Crée un nouveau signal depuis différents modèles :

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

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

Importer un fichier texte

Importer les données depuis un fichier texte.

Voir aussi :

Voir la page *Importation d'un fichier texte signal* pour plus de détails sur l'importation de fichiers texte.

Ouvrir un fichier HDF5

Importer les données d'un fichier HDF5.

Enregistrer un fichier HDF5

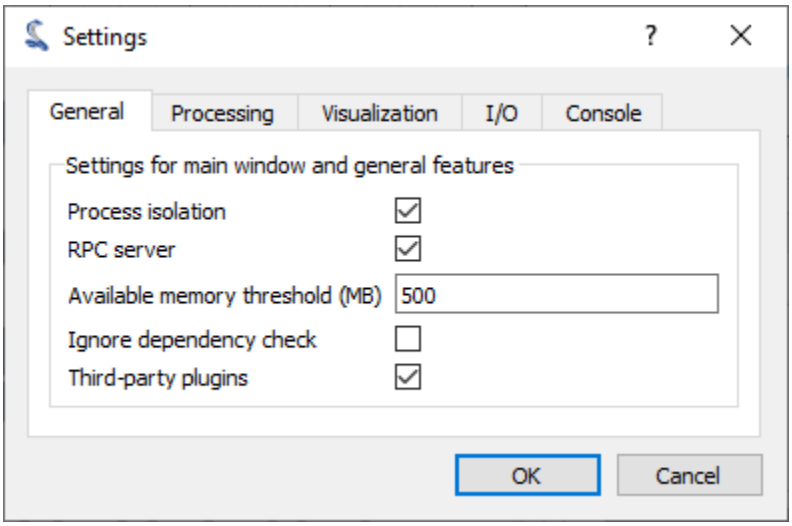
Exporter l'ensemble de la session DataLab (tous les signaux et images) vers un fichier HDF5.

Explorer un fichier HDF5

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

Préférences

Ouvrir la boîte de dialogue « Préférences ».



Menu « Edition »

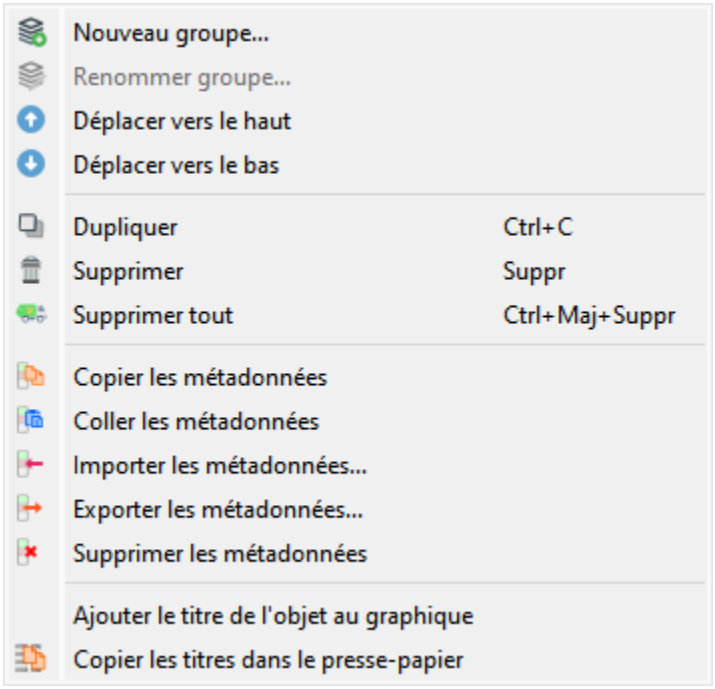


FIG. 10 – Capture d'écran du menu « Edition ».

Le menu « Edition » permet d'éditer le signal ou le groupe de signaux courant, en ajoutant, supprimant, renommant, déplaçant vers le haut ou vers le bas, ou dupliquant des signaux. Il permet également de manipuler les métadonnées, ou de gérer les titres des signaux.

Nouveau groupe

Crée un nouveau groupe de signaux. Les images peuvent être déplacées d'un groupe à un autre par glisser-déposer.

Renommer le groupe

Renomme le groupe sélectionné.

Déplacer vers le haut

Déplace la sélection vers le haut dans la liste (groupes ou signaux peuvent être sélectionnés). Si plusieurs objets sont sélectionnés, ils sont déplacés ensemble. Si un signal sélectionné est déjà en haut de son groupe, il est déplacé en bas du groupe précédent.

Déplacer vers le bas

Déplace la sélection vers le bas dans la liste (groupes ou signaux peuvent être sélectionnés). Si plusieurs objets sont sélectionnés, ils sont déplacés ensemble. Si un signal sélectionné est déjà en bas de son groupe, il est déplacé en haut du groupe suivant.

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 du signal sélectionné dans le presse-papier.

Coller les métadonnées

Coller les métadonnées depuis le presse-papier vers le signal sélectionné.

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.

Menu « Opérations »

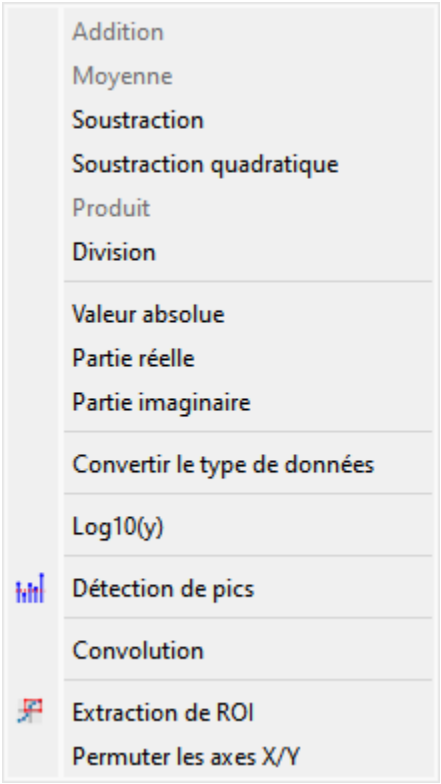


FIG. 11 – Capture d'écran du menu « Opérations ».

Le menu « Opérations » permet d'effectuer diverses opérations sur les signaux sélectionnés, telles que des opérations arithmétiques, la détection de pics, ou encore la convolution.

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 :

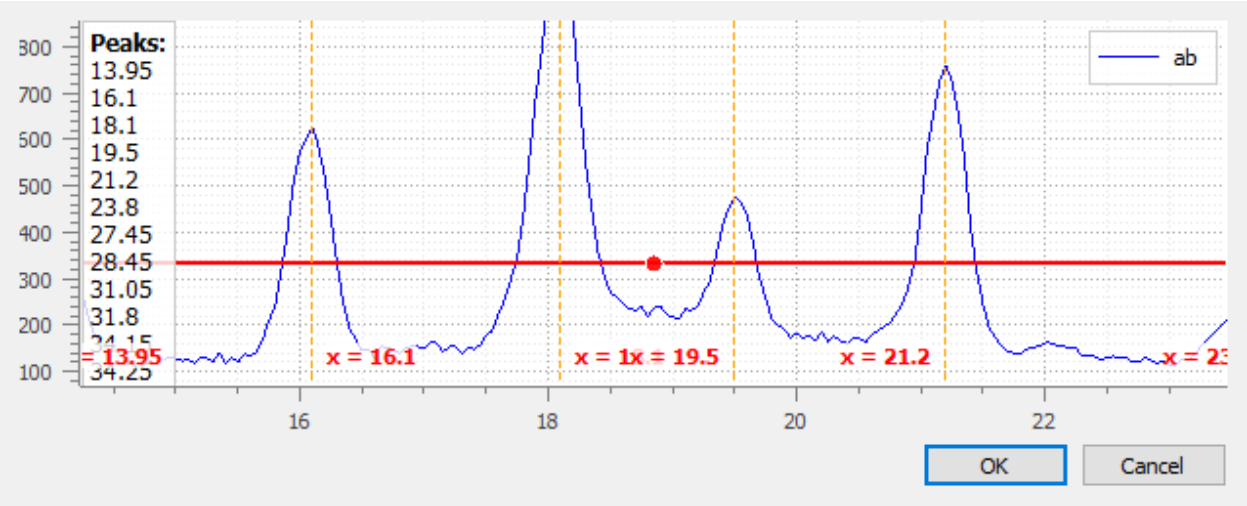


FIG. 12 – Boîte de dialogue de détection de pics : le seuil de détection est ajustable en déplaçant le curseur horizontal, les pics sont détectés automatiquement (des marqueurs verticaux indiquent les pics détectés avec leur position)

Convolution

Crée un signal à partir de la convolution de chaque signal sélectionné par rapport à un autre signal.

Cette fonctionnalité est basée sur la fonction scipy.signal.convolve de SciPy.

Extraction de ROI

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

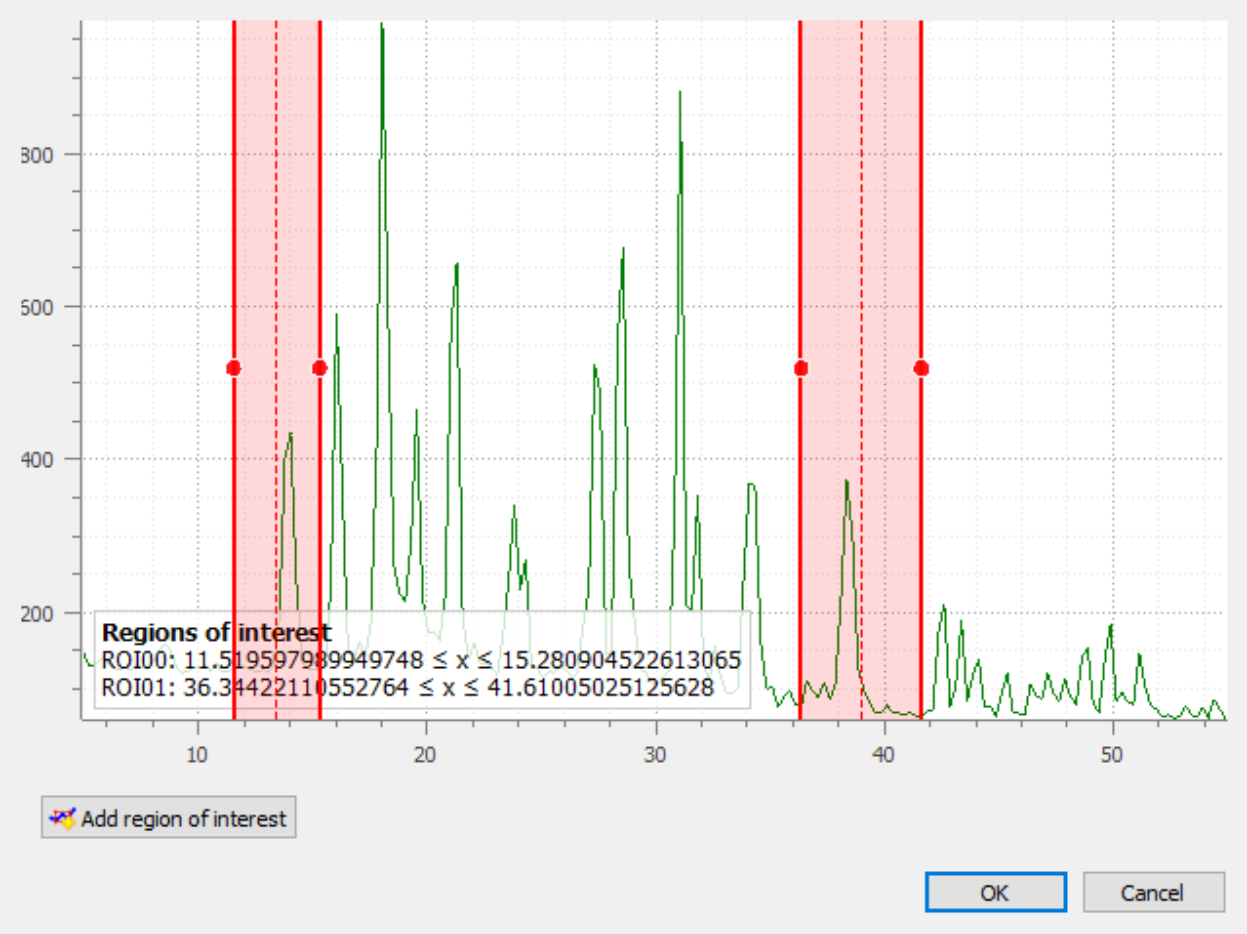


FIG. 13 – Boîte de dialogue d'extraction de ROI : la région d'intérêt (ROI) est définie en ajustant la position et la largeur de l'échelle horizontale de sélection.

Permuter les axes X/Y

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

Menu « Traitement »

FIG. 14 – Capture d'écran du menu « Traitement ».

Le menu « Traitement » permet d'effectuer divers traitements sur les signaux sélectionnés, tels que le lissage, la normalisation, ou encore l'interpolation.

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

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.

Interpolation

Crée un signal à partir de l'interpolation de chaque signal sélectionné, par rapport à l'axe X d'un second signal (qui peut être l'un des signaux sélectionnés).

Les méthodes d'interpolation suivantes sont disponibles :

Méthode	Description
Linéaire	Interpolation linéaire, utilisant la fonction interp de NumPy.
Spline	Interpolation par spline cubique, utilisant la fonction scipy.interpolate.splev de SciPy.
Quadratique	Interpolation quadratique, utilisant la fonction polyval de NumPy.
Cubique	Interpolation cubique, utilisant la classe Akima1DInterpolator de SciPy.
Barycentrique	Interpolation barycentrique, utilisant la classe BarycentricInterpolator de SciPy.
PCHIP	Interpolation par polynôme de Hermite cubique par morceaux (PCHIP), utilisant la classe PchipInterpolator de SciPy.

Rééchantillonnage

Crée un signal à partir du rééchantillonnage de chaque signal.

Les paramètres suivants sont disponibles :

Paramètre	Description
Méthode	Méthode d'interpolation (voir section précédente)
Valeur de remplissage	Valeur de remplissage pour l'interpolation (voir section précédente)
Xmin	Borne inférieure
Xmax	Borne supérieure
Mode	Mode de rééchantillonnage : pas ou nombre de points
Pas	Pas de rééchantillonnage
Nombre de points	Nombre de points de rééchantillonnage

Elimination de tendance

Crée un signal à partir de l'élimination de tendance de chaque signal. Cette fonctionnalité est basée sur la fonction scipy.signal.detrend de SciPy.

Les paramètres suivants sont disponibles :

Paramètre	Description
Méthode	Méthode d'élimination de tendance : "linéaire" ou "constante". Voir la fonction scipy.signal.detrend de SciPy.

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

Menu « Calculs »

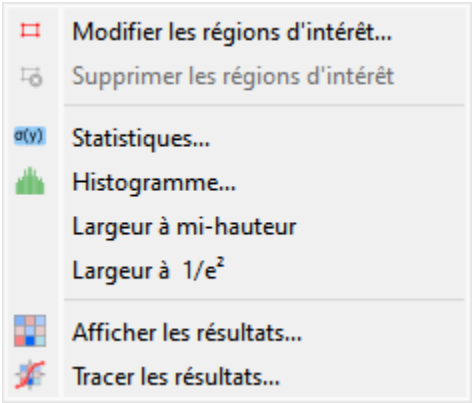


FIG. 15 – Capture d'écran du menu « Calculs ».

Le menu « Calculs » permet d'effectuer divers calculs sur les signaux sélectionnés, tels que des statistiques, la largeur à mi-hauteur, ou encore la largeur à $1/e^2$.

Note : Dans le vocabulaire de DataLab, un « calcul » est une fonctionnalité qui calcule un résultat scalaire à partir d'un signal. Ce résultat est stocké sous la forme de métadonnées ; il est donc attaché au signal. Cela diffère d'un « traitement »

qui crée un nouveau signal à partir d'un signal existant.

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 est ajustant la position et la largeur de l'échelle horizontale de sélection.

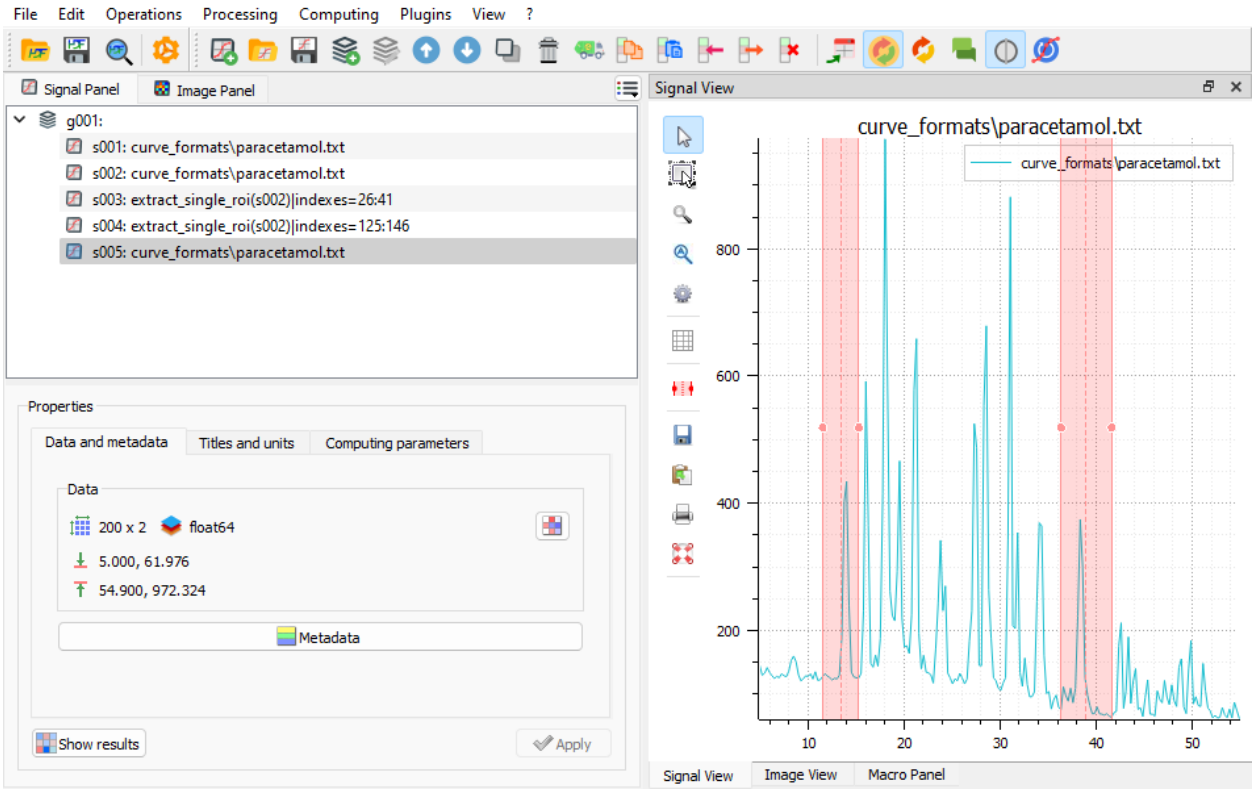


FIG. 16 – Un signal avec une ROI.

Supprimer les régions d'intérêt

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

Statistiques

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

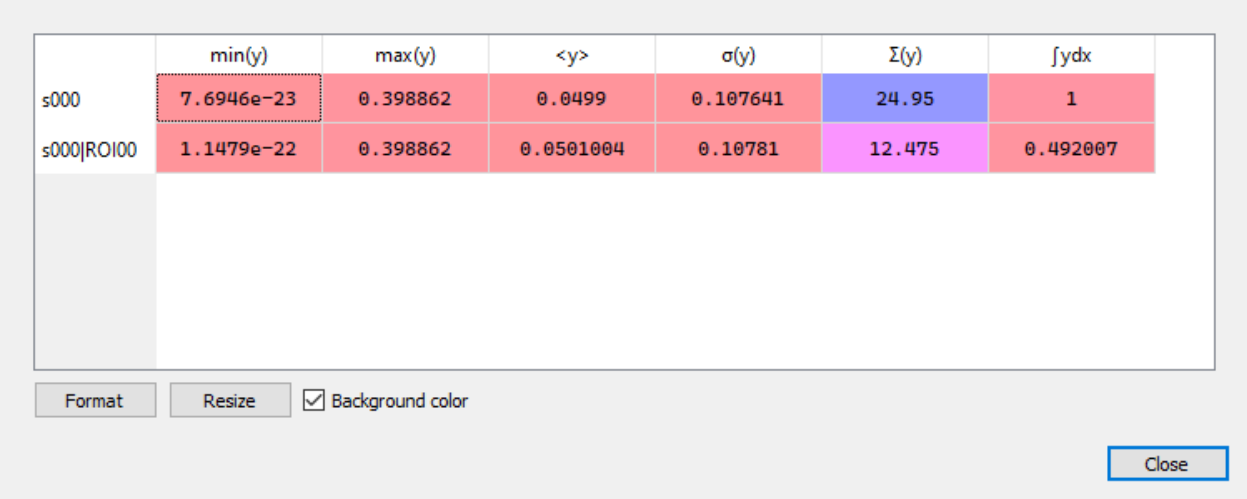


FIG. 17 – Exemple de tableau récapitulatif de statistiques : chaque ligne est associée à une ROI (à l'exception de la première qui correspond aux statistiques calculées sur la totalité des données).

Histogramme

Calcule l'histogramme du signal sélectionné et l'affiche.

Paramètres :

Paramètre	Description
Classes	Nombre de classes
Limite inférieure	Limite inférieure de l'histogramme
Limite supérieure	Limite supérieure de l'histogramme

Largeur à mi-hauteur

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

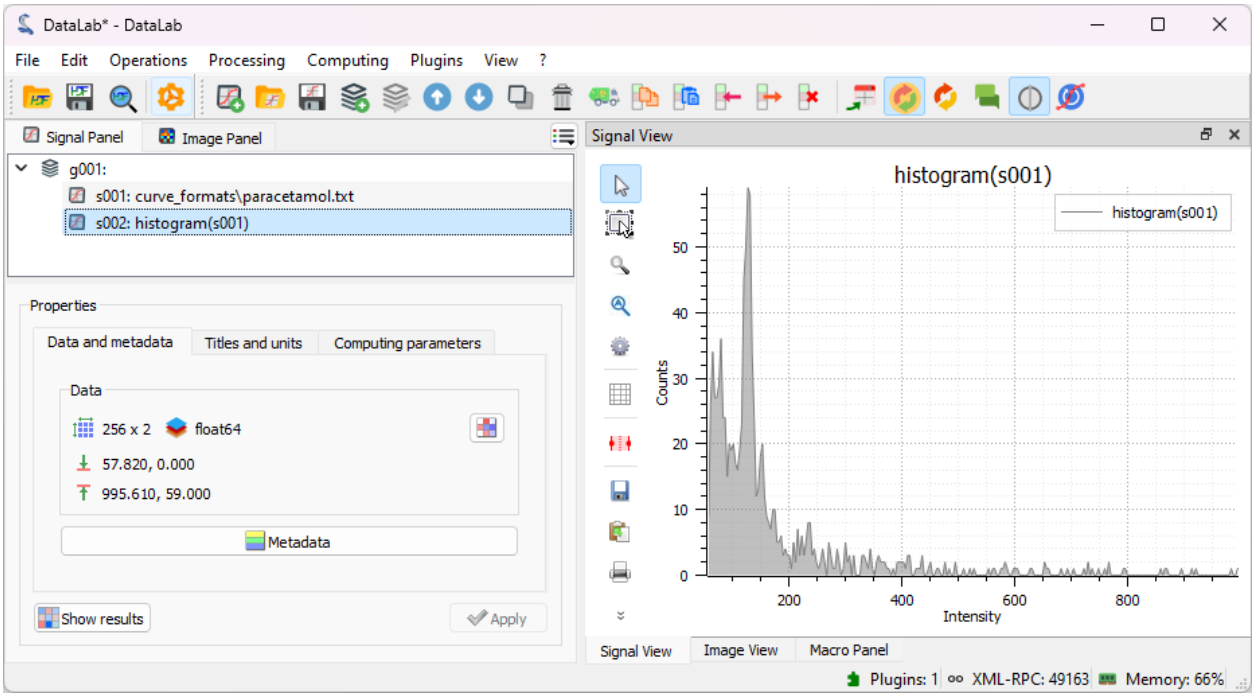


FIG. 18 – Exemple d'histogramme.

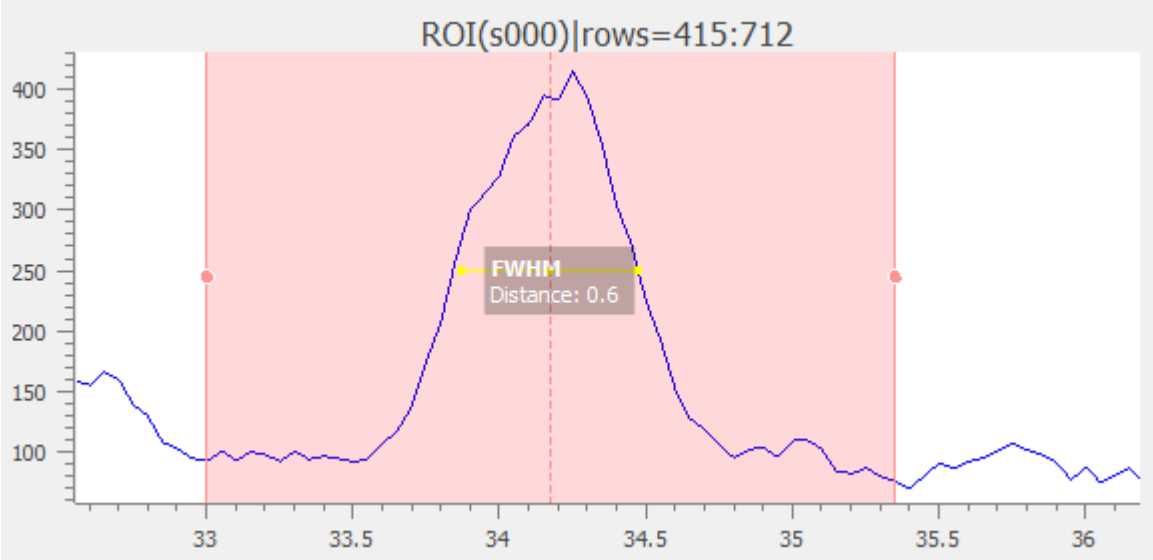


FIG. 19 – Le résultat du calcul est affiché sous la forme d'un segment annoté.

Largeur à 1/e²

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.

Afficher les résultats

Affiche les résultats de tous les calculs effectués sur les signaux sélectionnés. Cela affiche le même tableau que celui affiché après avoir effectué un calcul.

Tracer les résultats

Trace les résultats des calculs effectués sur les signaux sélectionnés, avec des axes X et Y définis par l'utilisateur (p.ex. trace la largeur à mi-hauteur en fonction de l'indice du signal).

Menu « Affichage »

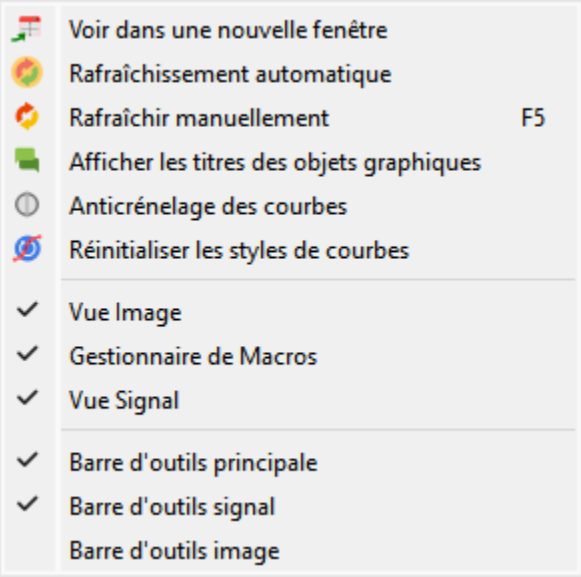


FIG. 20 – Capture d'écran du menu « Affichage ».

Le menu « Affichage » permet de visualiser le signal ou le groupe de signaux courant. Il permet également d'afficher/cacher les titres, d'activer/désactiver l'anticrénelage, ou encore de rafraîchir la visualisation.

Voir dans une nouvelle fenêtre

Ouvre une nouvelle fenêtre pour visualiser les signaux sélectionées.

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.

Réinitialiser les styles de courbes

Lors de l'affichage de courbes, DataLab attribue automatiquement une couleur et un style de ligne à chaque courbe. Les deux paramètres sont choisis dans une liste prédéfinie de couleurs et de styles de ligne, et sont attribués de manière cyclique.

Cette entrée de menu permet de réinitialiser les styles de courbes, de sorte que la prochaine fois que vous afficherez des courbes, la première courbe sera attribuée la première couleur et le premier style de ligne de la liste prédéfinie, et la boucle recommencera à partir de là.

Autres entrées du menu

Affiche/cache les panneaux et barres d'outils.

Menu « ? »

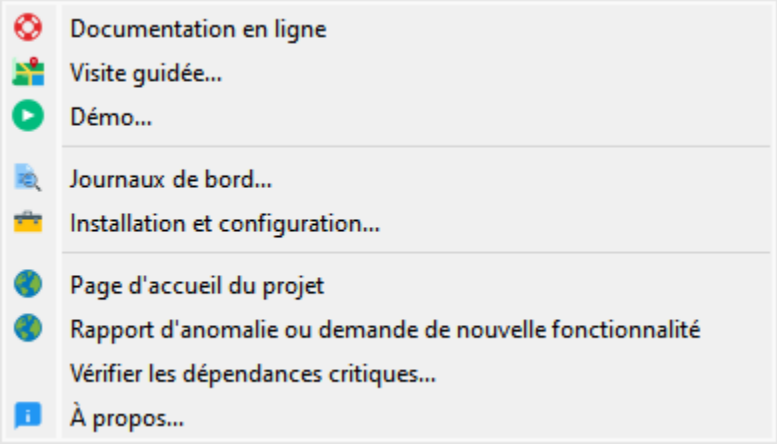
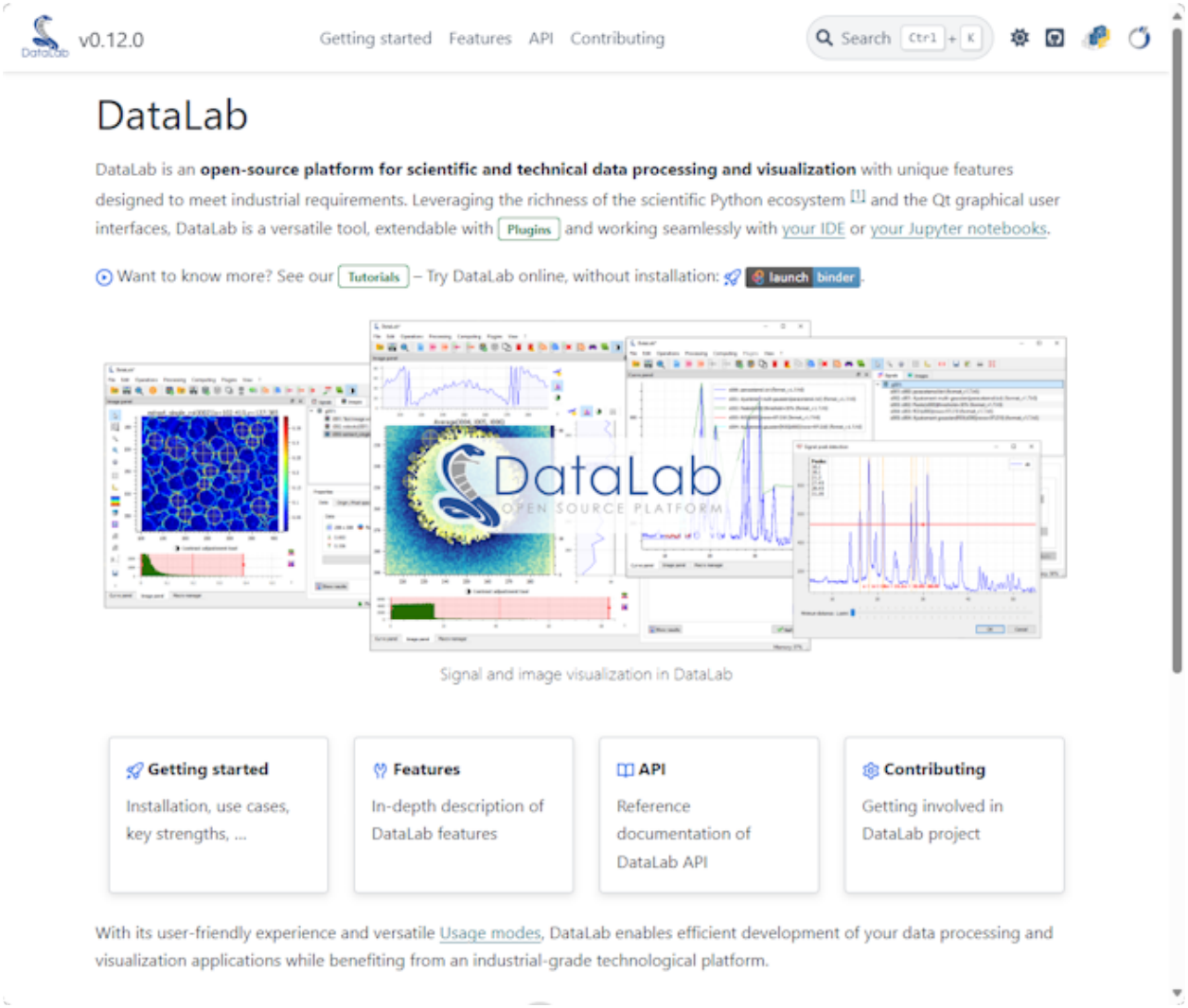


FIG. 21 – Capture d'écran du menu « ? ».

Le menu « ? » permet d'accéder à la documentation en ligne, d'afficher les journaux de bords, d'afficher des informations sur votre configuration d'installation de DataLab, et d'afficher la boîte de dialogue « A propos de DataLab ».

Documentation en ligne ou locale

Affiche la documentation en ligne ou locale :



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

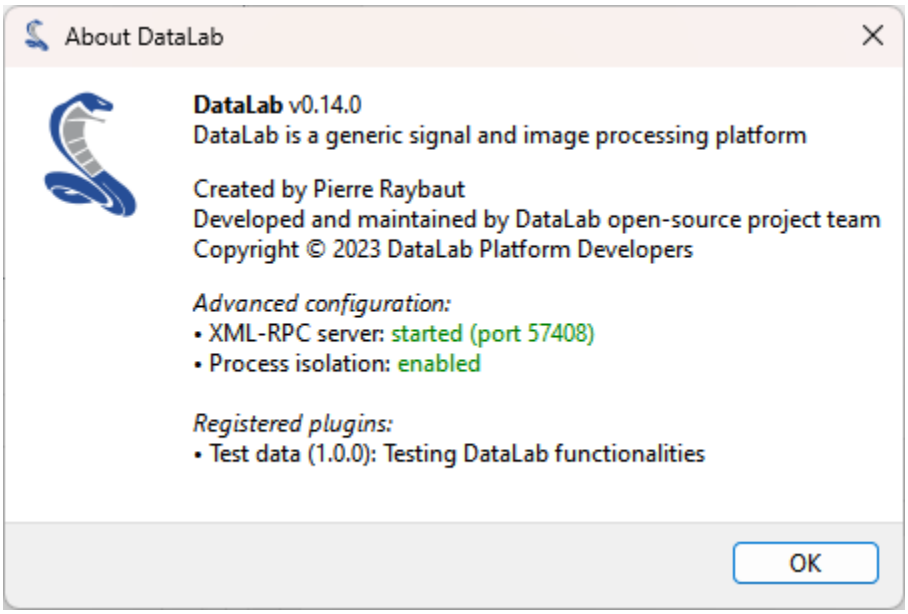
Affiche des informations sur votre configuration d'installation de DataLab (particulièrement utile pour signaler des anomalies de manière efficace).

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 »



2.2.2 Importation d'un fichier texte signal

DataLab peut importer nativement des fichiers de signaux (par exemple CSV, NPY, etc.). Cependant, certains formats de fichiers texte spécifiques peuvent ne pas être pris en charge. Dans ce cas, vous pouvez utiliser la fonctionnalité « Importer un fichier texte », qui vous permet d'importer un fichier texte et de le convertir en signal.

Cette fonctionnalité est accessible depuis le menu « Fichier », sous l'option « Importer un fichier texte ».

Il ouvre un assistant d'importation qui vous guide tout au long du processus d'importation du fichier texte.

Etape 1 : Sélectionner la source

La première étape consiste à sélectionner la source du fichier texte. Vous pouvez soit sélectionner un fichier de votre ordinateur, soit le presse-papiers si vous avez copié le texte à partir d'une autre application.

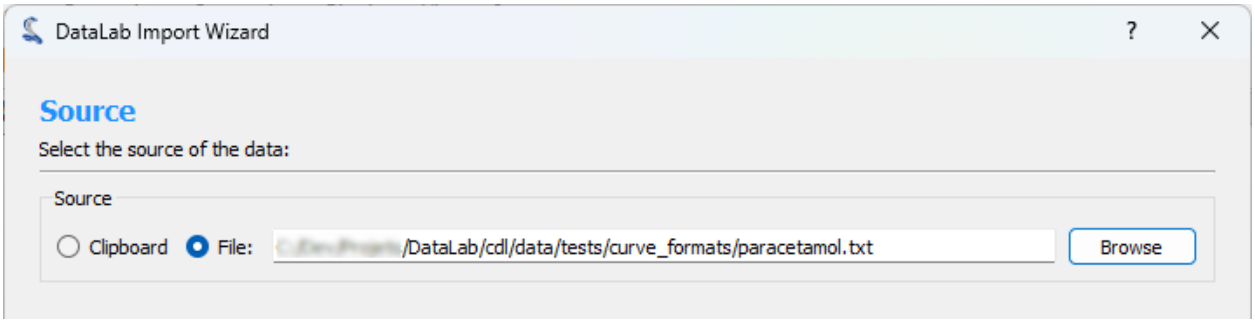


FIG. 22 – Etape 1 : Sélectionner la source

Etape 2 : Aperçu et configuration de l'importation

La deuxième étape consiste à configurer l'importation et à prévisualiser le résultat. Vous pouvez configurer les options suivantes :

- **Délimiteur** : Le caractère utilisé pour séparer les valeurs dans le fichier texte.
- **Commentaires** : Le caractère utilisé pour indiquer que la ligne est un commentaire et doit être ignorée.
- **Lignes à sauter** : Le nombre de lignes à sauter au début du fichier.
- **Nombre maximum de lignes** : Le nombre maximum de lignes à importer. Si le fichier contient plus de lignes, elles seront ignorées.
- **Transposer** : Si coché, les lignes et les colonnes seront transposées.
- **Type de données** : Le type de données de destination des données importées.
- **La première colonne est X** : Si coché, la première colonne sera utilisée comme axe X.

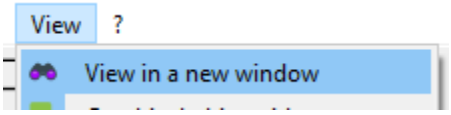
Lorsque vous avez terminé de configurer l'importation, cliquez sur le bouton « Appliquer » pour voir le résultat.

Etape 3 : Afficher la représentation graphique

La troisième étape montre une représentation graphique des données importées. Vous pouvez utiliser le bouton « Terminer » pour importer les données dans l'espace de travail de DataLab.

2.2.3 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.

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-desous).

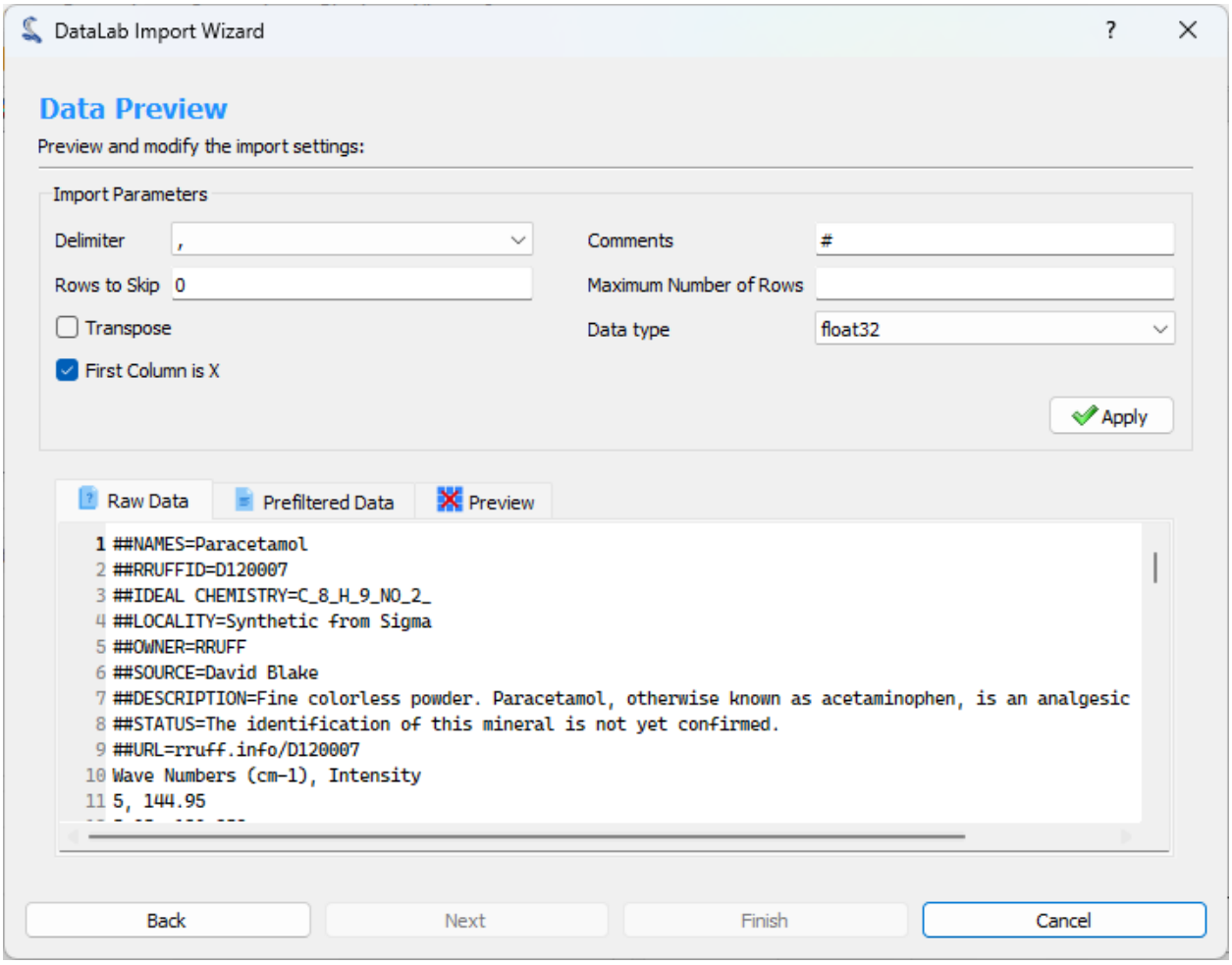


FIG. 23 – Etape 2 : Configurer l'importation

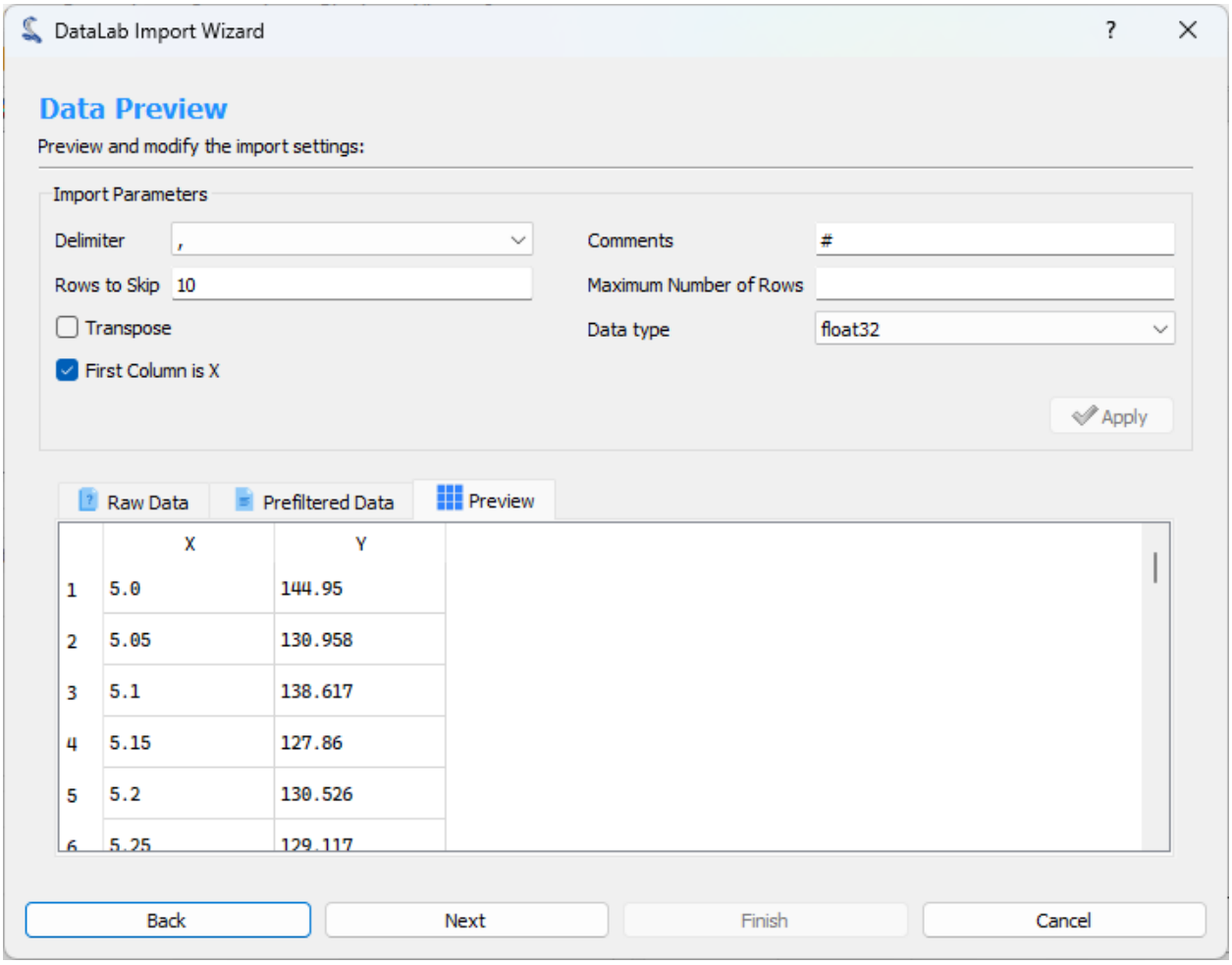


FIG. 24 – Etape 2 : Prévisualiser le résultat

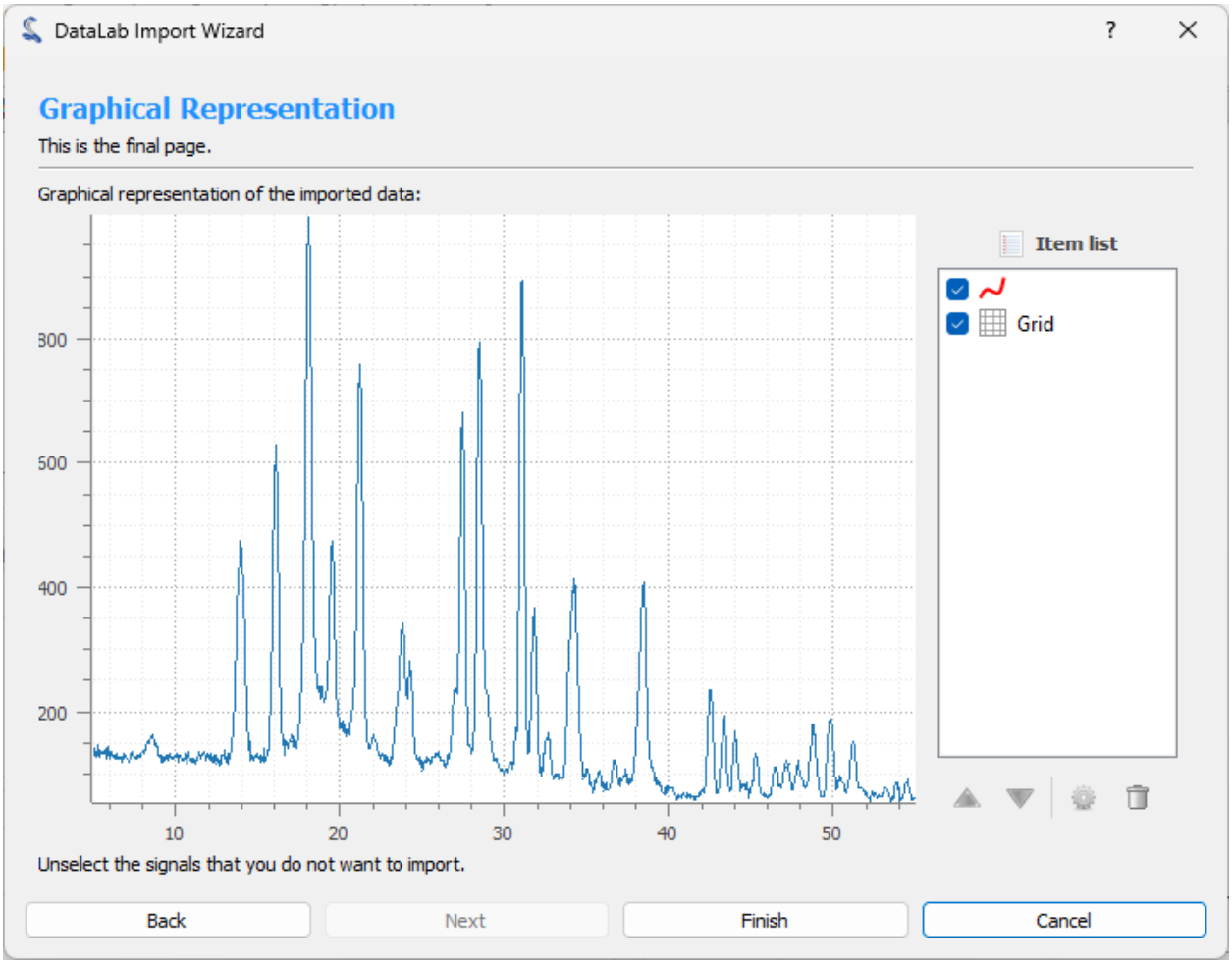


FIG. 25 – Etape 3 : Afficher la représentation graphique

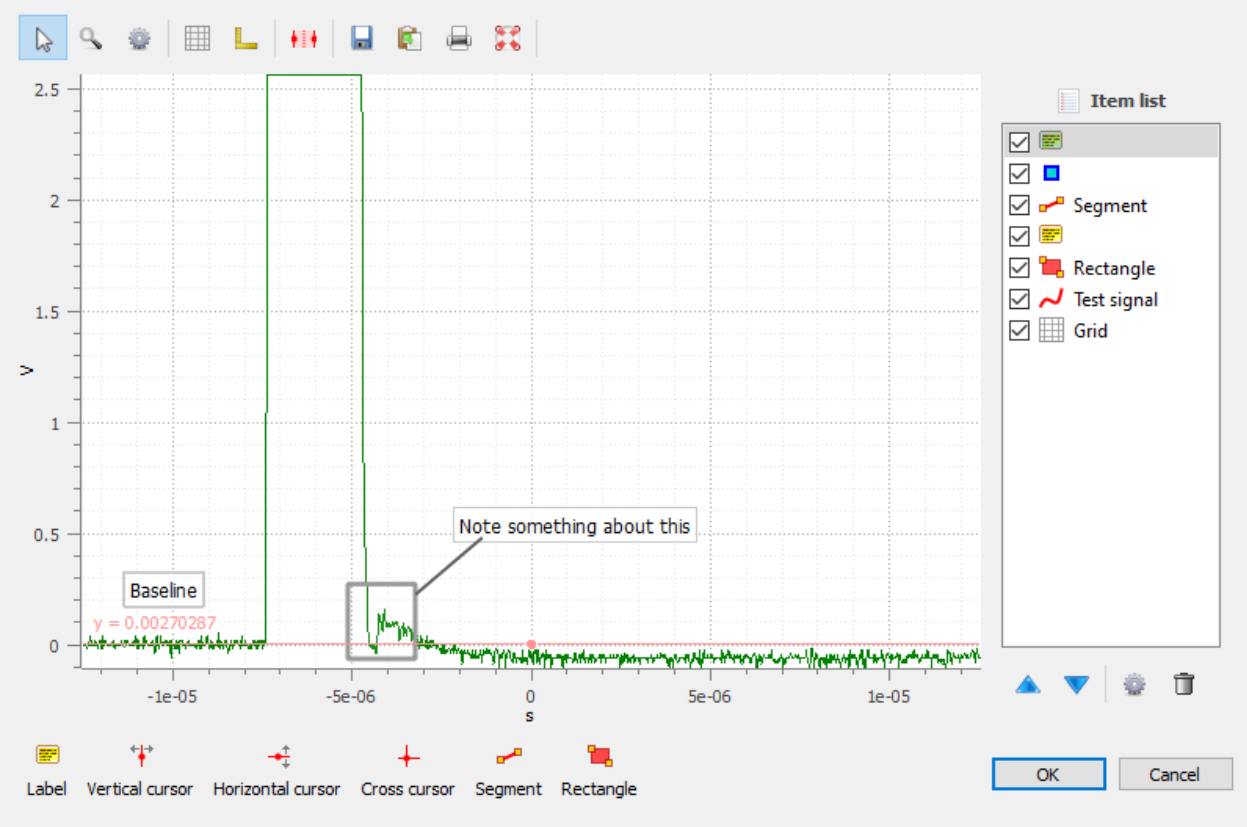


FIG. 26 – Les annotations peuvent être ajoutées dans la fenêtre séparée.

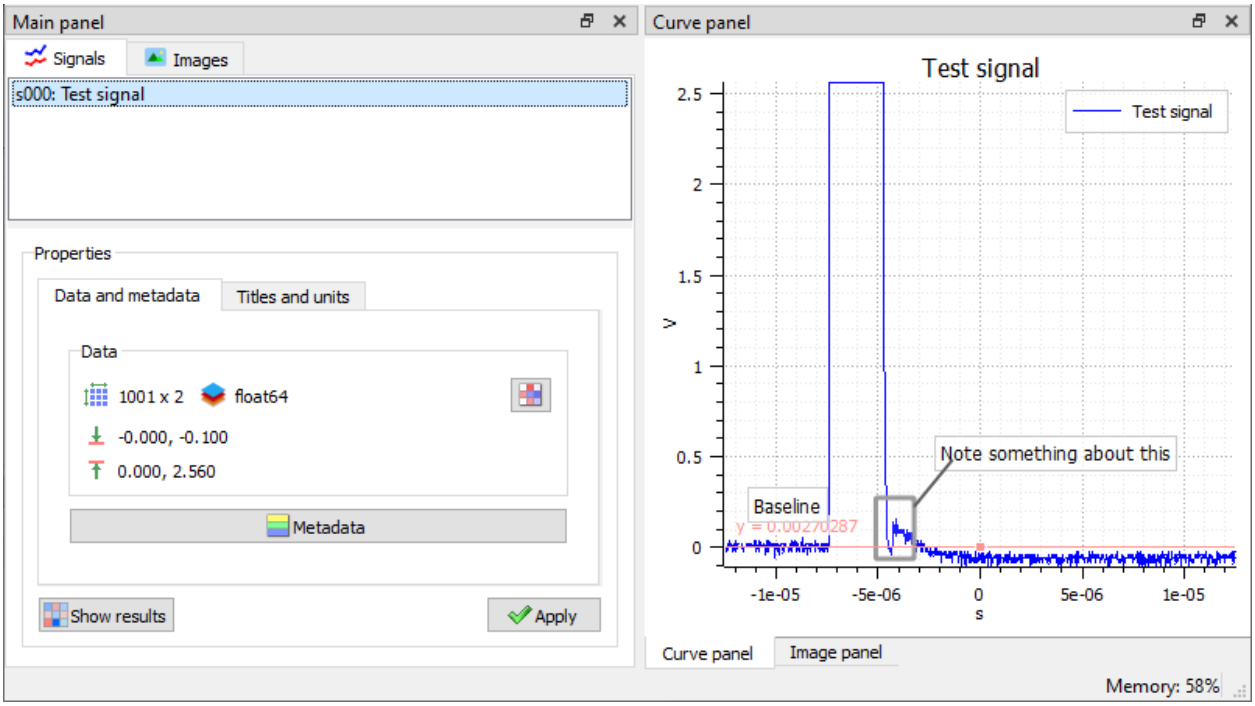


FIG. 27 – Les annotations font maintenant partie des métadonnées du signal.

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

2.3 Traitement d'image

Cette section décrit les fonctionnalités spécifiques au panneau de traitement d'image. Le panneau de traitement d'image peut être sélectionné en cliquant sur l'onglet « Images » en bas à droite de la fenêtre principale de DataLab.

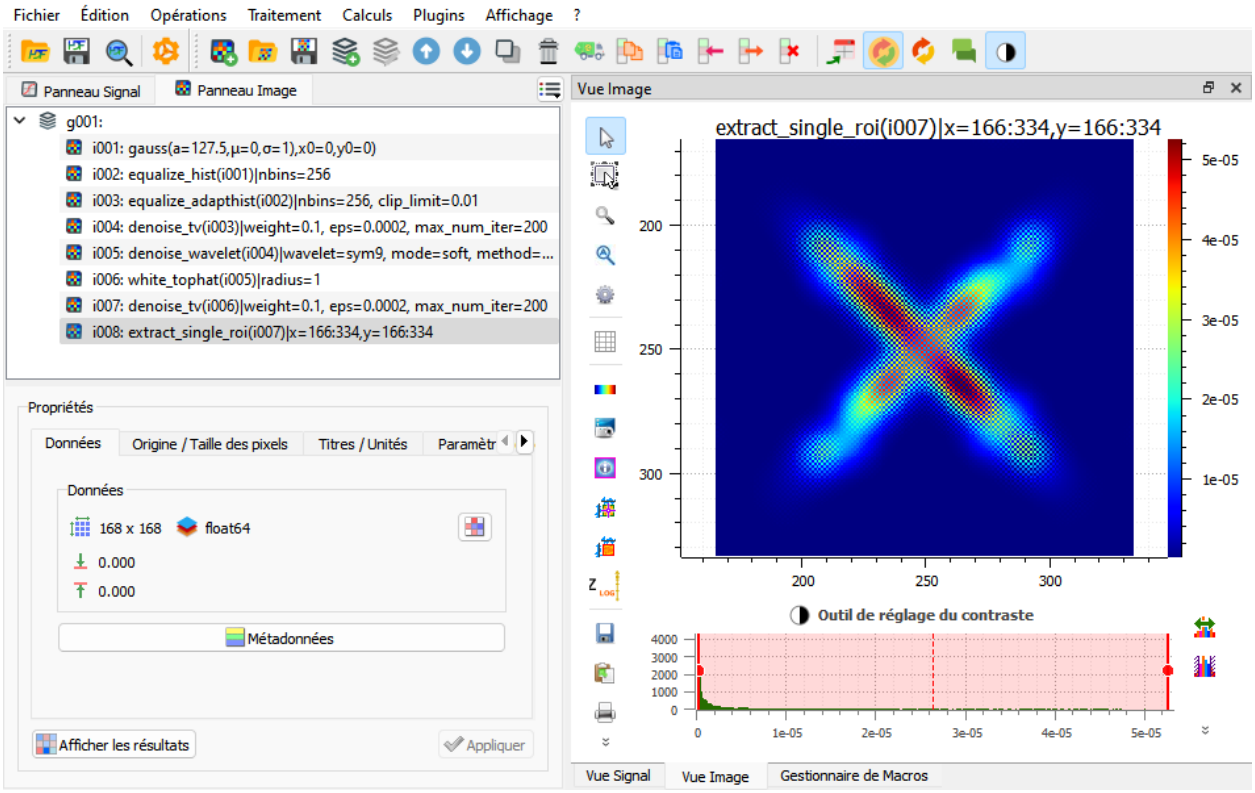


FIG. 28 – Fenêtre principale de DataLab : vue du traitement d'image

2.3.1 Menus

Cette section décrit les fonctionnalités liées aux images de DataLab, en présentant les différents menus et leurs fonctions associées.

Menu « Fichier »

Le menu « Fichier » permet de créer, ouvrir, enregistrer et fermer des images. Il permet également d'importer et d'exporter des données depuis/vers des fichiers HDF5, et d'éditer les paramètres de la session courante.

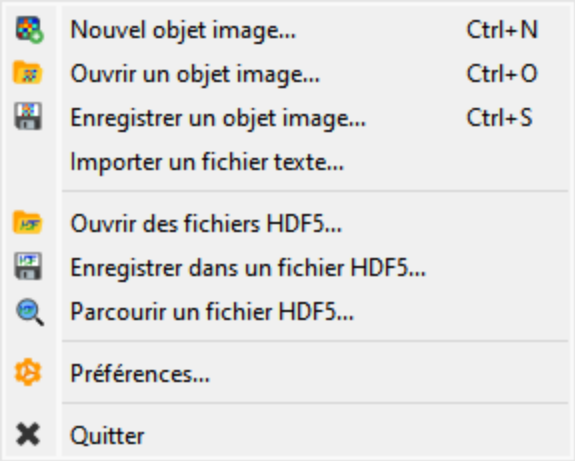


FIG. 29 – Capture d'écran du menu « Fichier ».

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 correpondant au type de données
Gaussienne 2D	$z = A.exp(-\frac{(\sqrt{(x-x0)^2+(y-y0)^2}-\mu)^2}{2\sigma^2})$

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

Enregister l'image

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

Importer un fichier texte

Importer les données depuis un fichier texte.

Voir aussi :

Voir la page *Importation d'un fichier texte image* pour plus de détails sur l'importation de fichiers texte.

Ouvrir un fichier HDF5

Importer les données d'un fichier HDF5.

Enregistrer un fichier HDF5

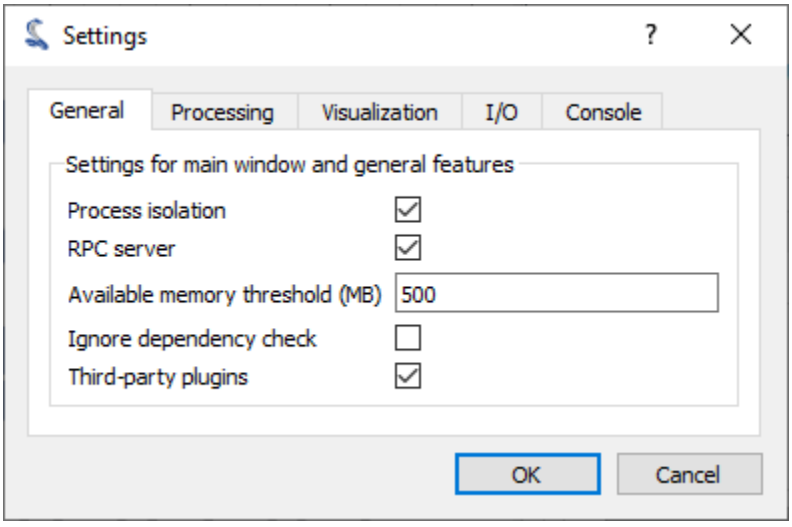
Exporter l'ensemble de la session DataLab (tous les signaux et images) vers un fichier HDF5.

Explorer un fichier HDF5

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

Préférences

Ouvrir la boîte de dialogue « Préférences ».



Menu « Edition »

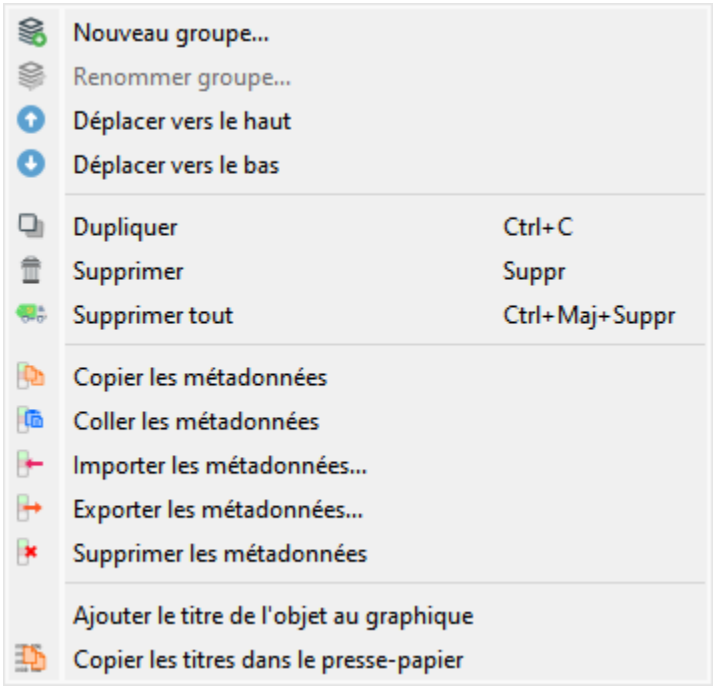


FIG. 30 – Capture d'écran du menu « Edition ».

Le menu « Edition » permet d'éditer l'image ou le groupe d'images courant, en ajoutant, supprimant, renommant, déplaçant vers le haut ou le bas, ou dupliquant des images. Il permet également de manipuler les métadonnées, ou de gérer les titres des images.

Nouveau groupe

Crée un nouveau groupe d'images. Les images peuvent être déplacées d'un groupe à un autre par glisser-déposer.

Renommer le groupe

Renomme le groupe sélectionné.

Déplacer vers le haut

Déplace la sélection vers le haut dans la liste (groupes ou images peuvent être sélectionnés). Si plusieurs objets sont sélectionnés, ils sont déplacés ensemble. Si une image sélectionnée est déjà en haut de son groupe, elle est déplacée en bas du groupe précédent.

Déplacer vers le bas

Déplace la sélection vers le bas dans la liste (groupes ou images peuvent être sélectionnés). Si plusieurs objets sont sélectionnés, ils sont déplacés ensemble. Si une image sélectionnée est déjà en bas de son groupe, elle est déplacée en haut du groupe suivant.

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.

Menu « Opérations »

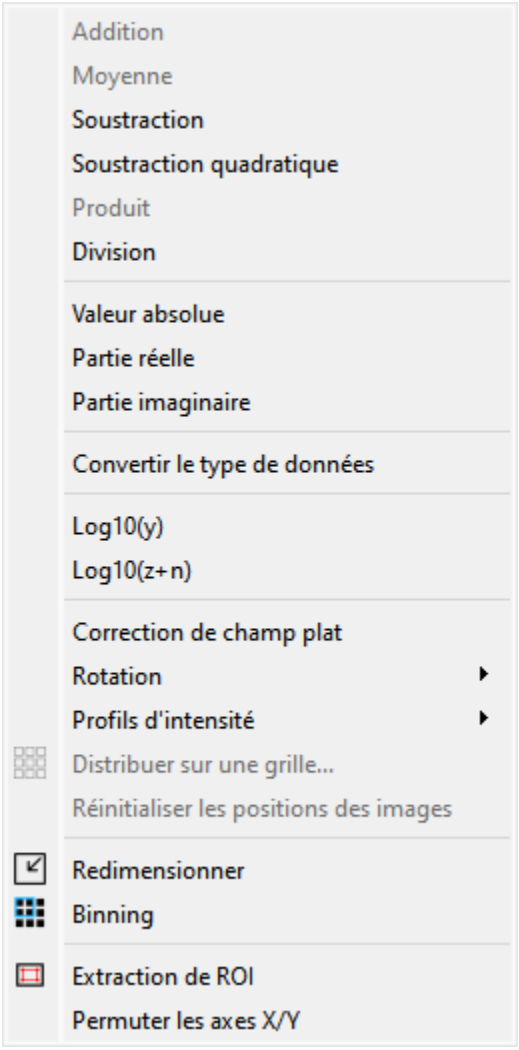


FIG. 31 – Capture d'écran du menu « Opérations ».

Le menu « Opérations » permet d'effectuer diverses opérations sur l'image ou le groupe d'images courant. Il permet également d'extraire des profils, de distribuer des images sur une grille, ou de redimensionner des images.

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

Créee 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 Log10(z+n) de chaque image sélectionnée (effet d'augmentation de la dynamique d'un logarithme en évitant de calculer Log10(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 :

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

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

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

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.

Profils d'intensité

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

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.

Extraire un profil radial
: Extraire un profil radial de chaque image sélectionnée et créer un nouveau signal à partir de ces profils.

Les paramètres suivants sont disponibles :

Paramètre	Description
Centre	Centre autour duquel le profil radial est calculé : centre de masse, centre de l'image, ou défini par l'utilisateur
X	Coordonnée X du centre (si défini par l'utilisateur), en pixels
Y	Coordonnée Y du centre (si défini par l'utilisateur), en pixels

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.

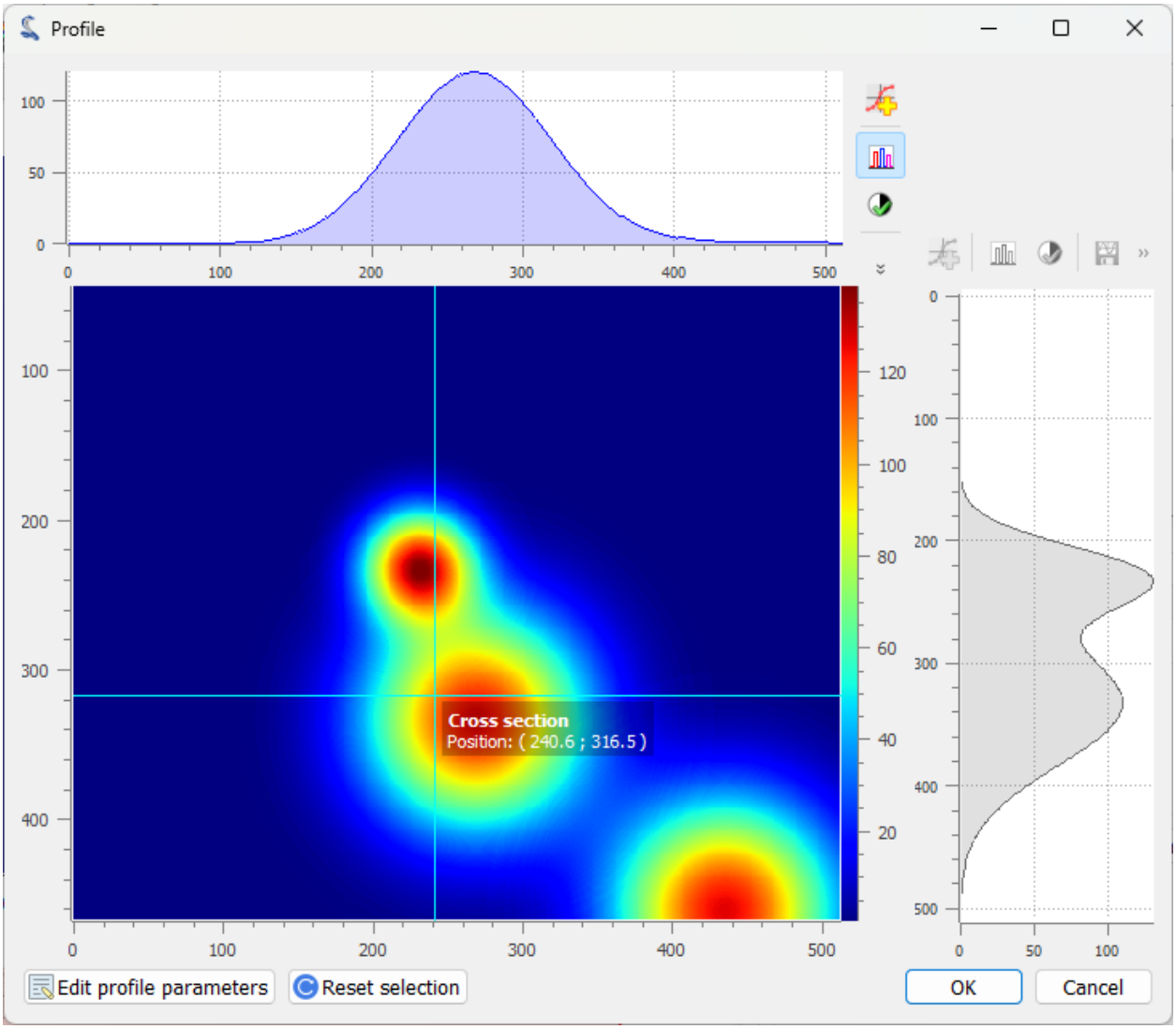


FIG. 32 – Boîte de dialogue d'extraction de profil. Les paramètres peuvent être également définis manuellement (bouton « Editer les paramètres du profil »).

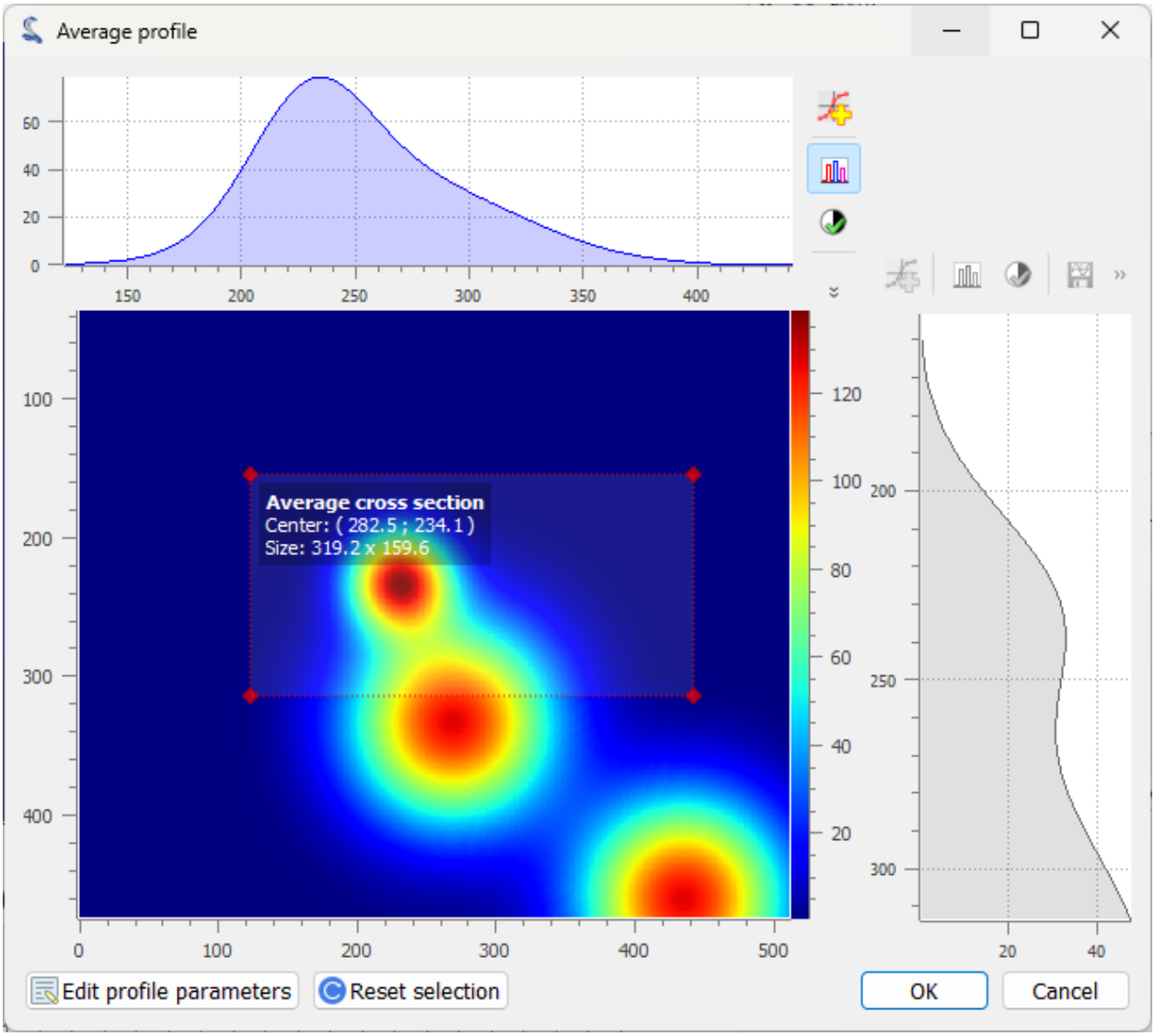


FIG. 33 – Boîte de dialogue d'extraction de profil moyen : la zone est définie par un rectangle. Les paramètres peuvent être également définis manuellement (bouton « Editer les paramètres du profil »).

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.

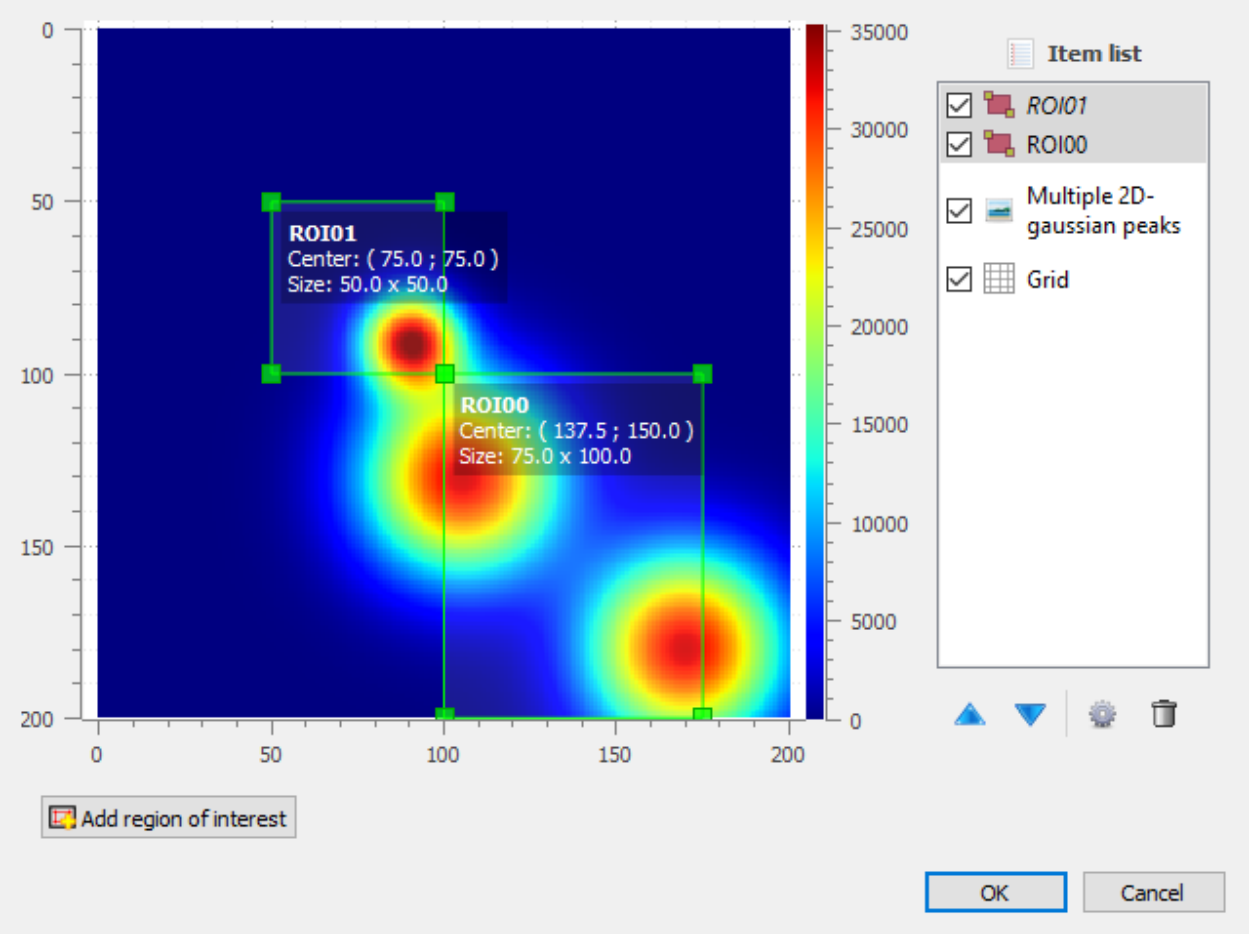


FIG. 34 – Boîte de dialogue d'extraction de ROI : la région d'intérêt (ROI) est définie en ajustant la position et la taille du rectangle de sélection.

Permuter les axes X/Y

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

Menu « Traitement »

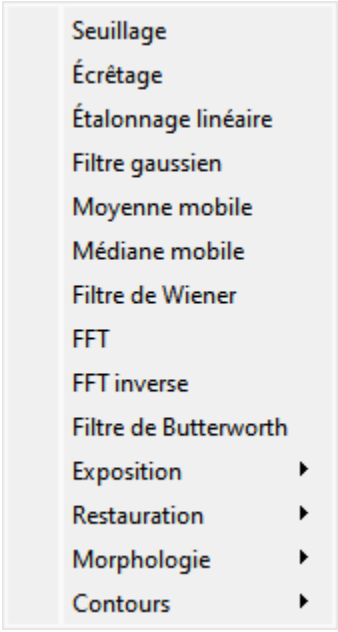


FIG. 35 – Capture d'écran du menu « Traitement ».

Le menu « Traitement » permet d'effectuer divers traitements sur l'image ou le groupe d'images courant : il permet d'appliquer des filtres, de corriger l'exposition, de réduire le bruit, d'effectuer des opérations morphologiques, etc.

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 paramétrabl

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)

Menu « Calculs »

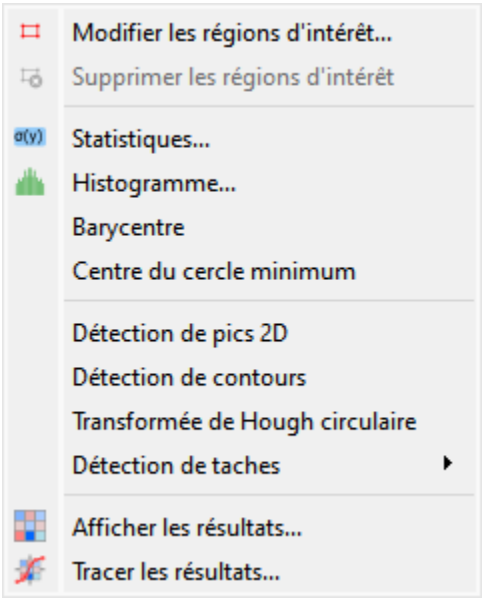


FIG. 36 – Capture d'écran du menu « Calculs ».

Le menu « Calculs » permet d'effectuer divers calculs sur l'image courante ou sur un groupe d'images. Il permet également de calculer des statistiques, le barycentre, de détecter des pics, des contours, etc.

Note : Dans le vocabulaire de DataLab, un « calcul » est une fonction qui calcule un résultat scalaire à partir d'une image. Ce résultat est stocké sous la forme de métadonnées, et donc attaché à l'image. C'est différent d'un « traitement » qui crée une nouvelle image à partir d'une image existante.

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).

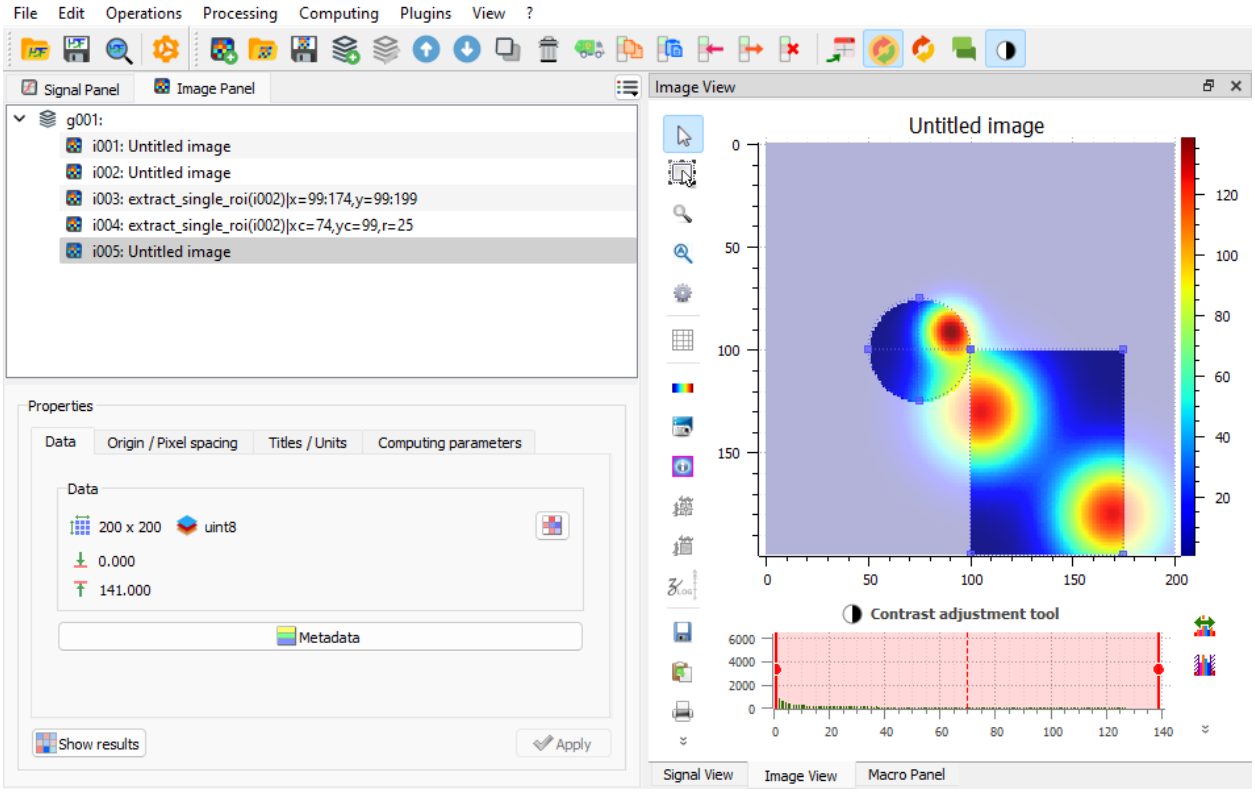


FIG. 37 – Une image avec une ROI.

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.

Histogramme

Calcule l'histogramme de l'image sélectionnée et l'affiche dans le panneau Signal.

Paramètres :

Paramètre	Description
Classes	Nombre de classes
Limite inférieure	Limite inférieure de l'histogramme
Limite supérieure	Limite supérieure de l'histogramme

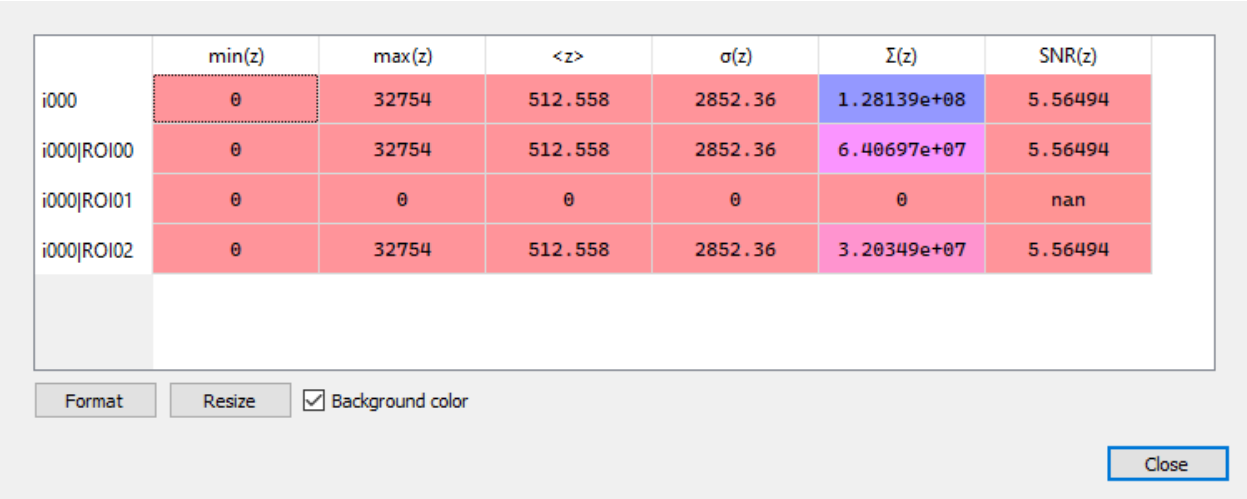


FIG. 38 – Exemple de tableau récapitulatif de statistiques : chaque ligne est associée à une ROI (à l'exception de la première qui correspond aux statistiques calculées sur la totalité des données).

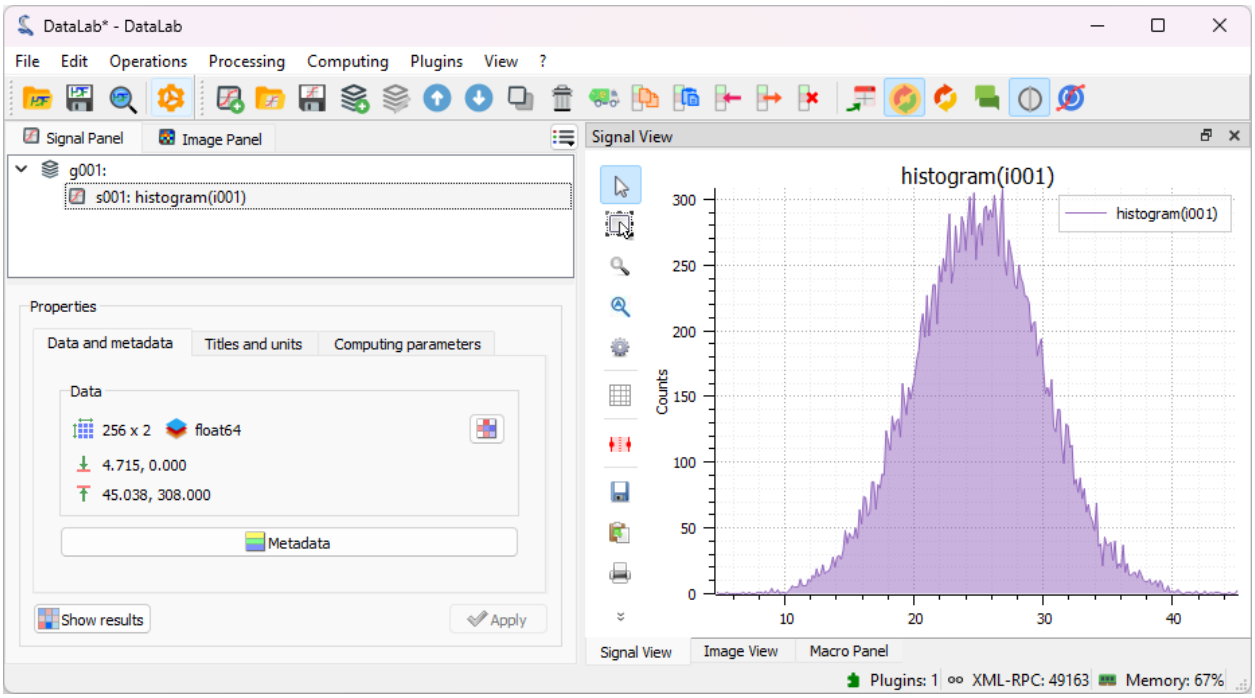


FIG. 39 – Exemple d'histogramme.

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.

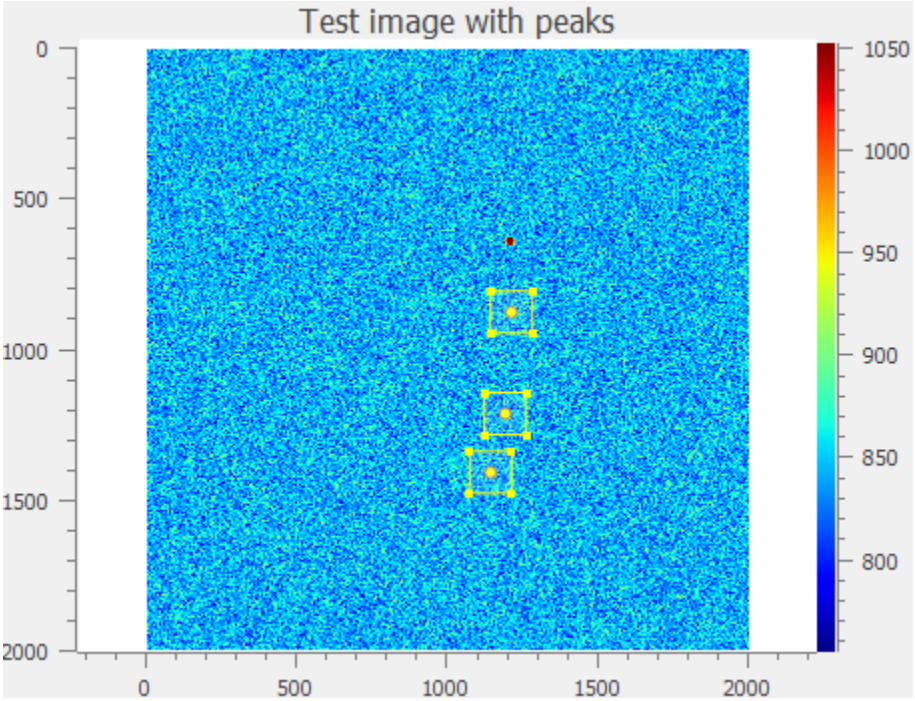


Fig. 40 – Exemple de détection de pics 2D.

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.

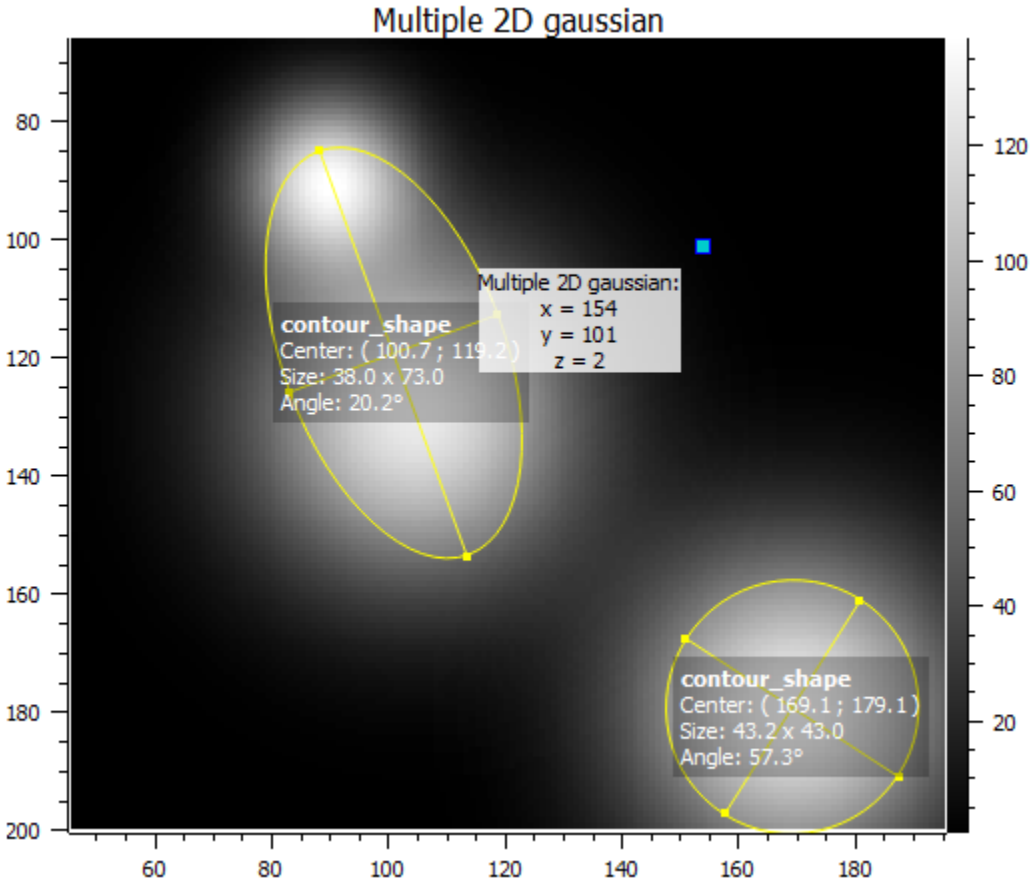


FIG. 41 – Exemple de détection de contours.

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 tranformée 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) (implementation basée sur skimage.feature.blob_dog).

Détection de taches (hessien)
: 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.

Afficher les résultats

Affiche les résultats des calculs effectués sur les images sélectionnées. Cela affiche le même tableau que celui affiché après avoir effectué un calcul.

Tracer les résultats

Trace les résultats des calculs effectués sur les images sélectionnées, avec des axes X et Y définis par l'utilisateur (p.ex. trace le rayon du cercle de contour en fonction du numéro de l'image).

Menu « Affichage »

Le menu « Affichage » permet de visualiser l'image courante ou un groupe d'images. Il permet également d'afficher/cacher les titres, d'afficher/cacher le panneau de contraste, de rafraîchir la visualisation, etc.

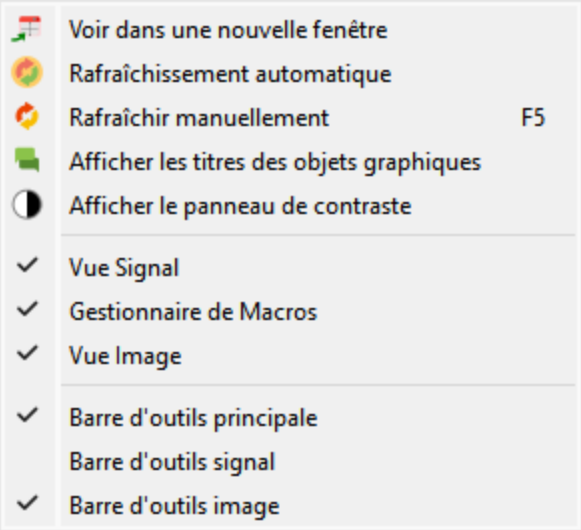


FIG. 42 – Capture d'écran du menu « Affichage ».

Voir dans une nouvelle fenêtre

Ouvre une nouvelle fenêtre pour visualiser les images sélectionées.

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.

Menu « ? »

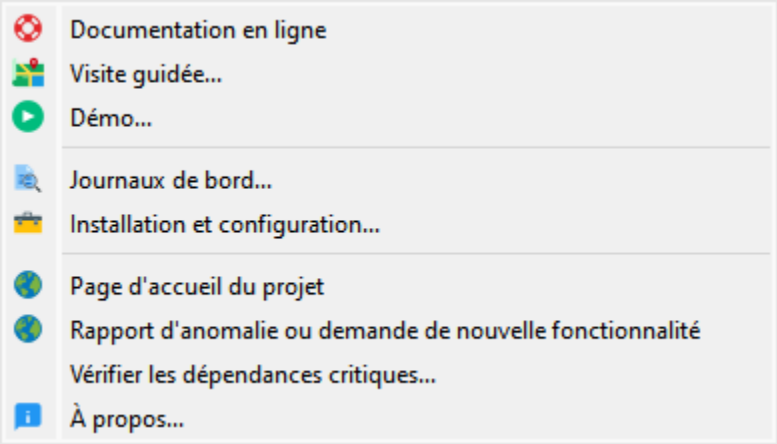
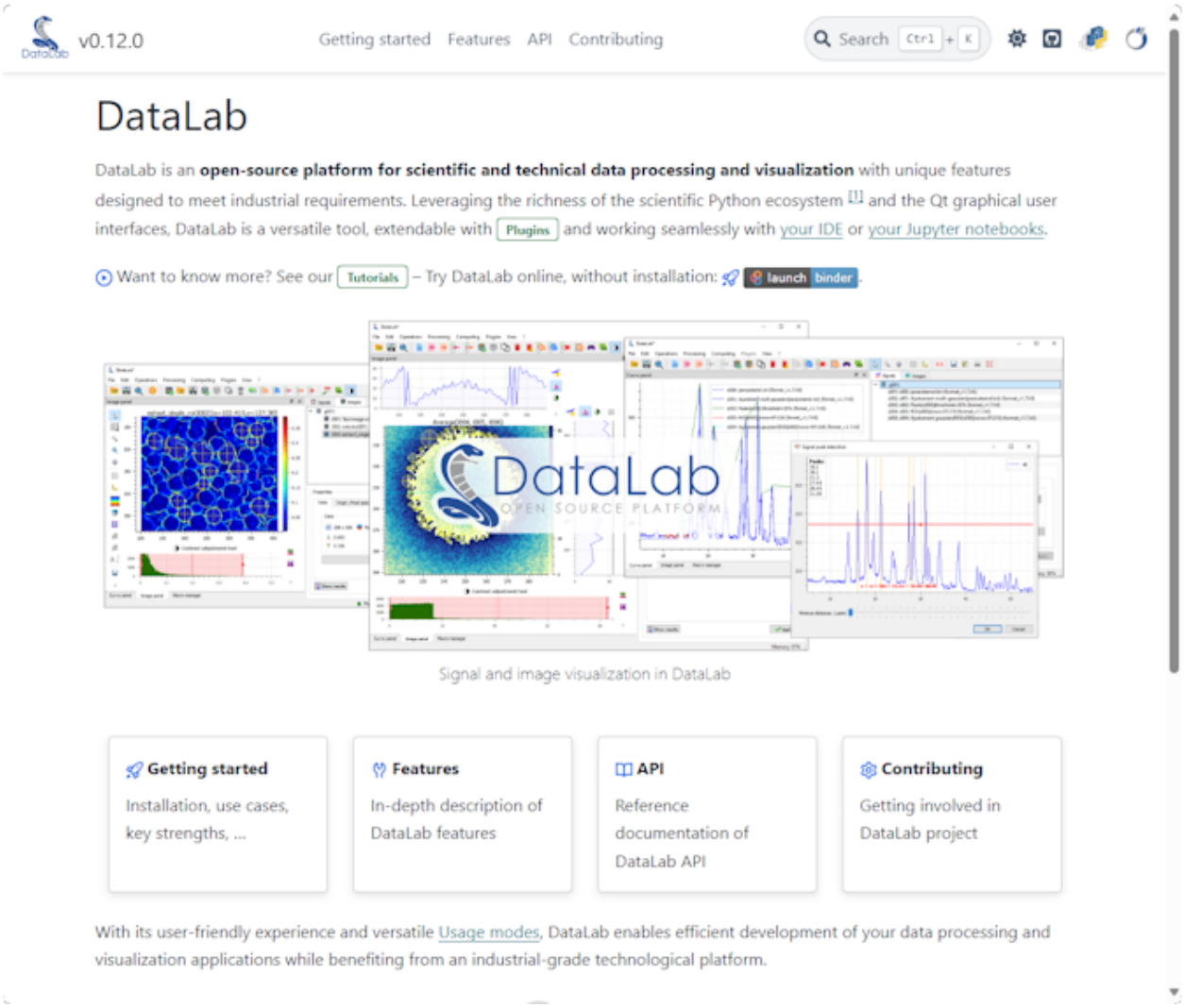


FIG. 43 – Capture d'écran du menu « ? ».

Le menu « ? » permet d'accéder à la documentation en ligne, d'afficher les journaux de bords, d'afficher des informations sur votre configuration d'installation de DataLab, et d'afficher la boîte de dialogue « A propos de DataLab ».

Documentation en ligne ou locale

Affiche la documentation en ligne ou locale :



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

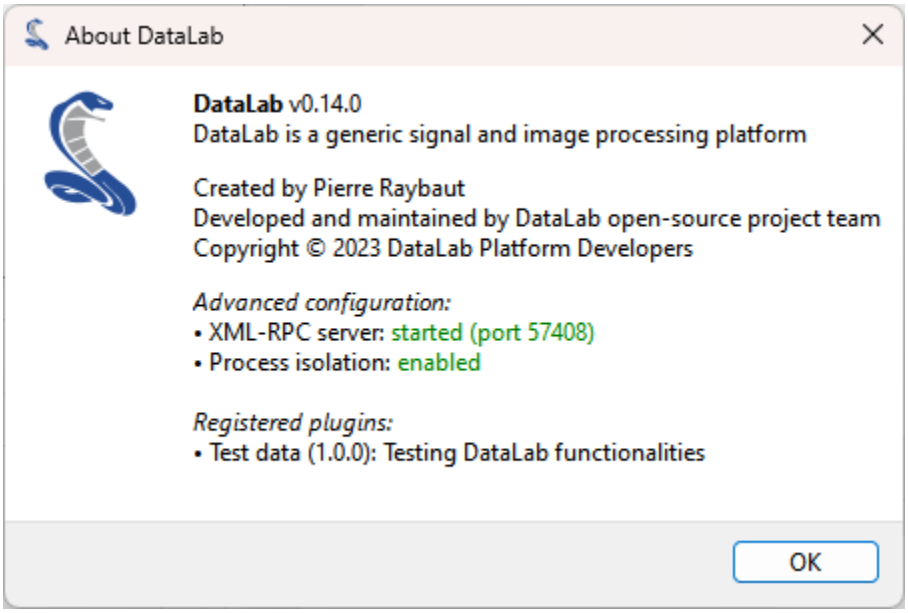
Affiche des informations sur votre configuration d'installation de DataLab (particulièrement utile pour signaler des anomalies de manière efficace).

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 »



2.3.2 Importation d'un fichier texte image

DataLab peut importer nativement de nombreux types de fichiers image (par exemple TIFF, JPEG, PNG, etc.). Cependant, certains formats de fichiers texte spécifiques peuvent ne pas être pris en charge. Dans ce cas, vous pouvez utiliser la fonctionnalité « Importer un fichier texte », qui vous permet d'importer un fichier texte et de le convertir en image.

Cette fonctionnalité est accessible depuis le menu « Fichier », sous l'option « Importer un fichier texte ».

Il ouvre un assistant d'importation qui vous guide tout au long du processus d'importation du fichier texte.

Etape 1 : Sélectionner la source

La première étape consiste à sélectionner la source du fichier texte. Vous pouvez soit sélectionner un fichier de votre ordinateur, soit le presse-papiers si vous avez copié le texte à partir d'une autre application.

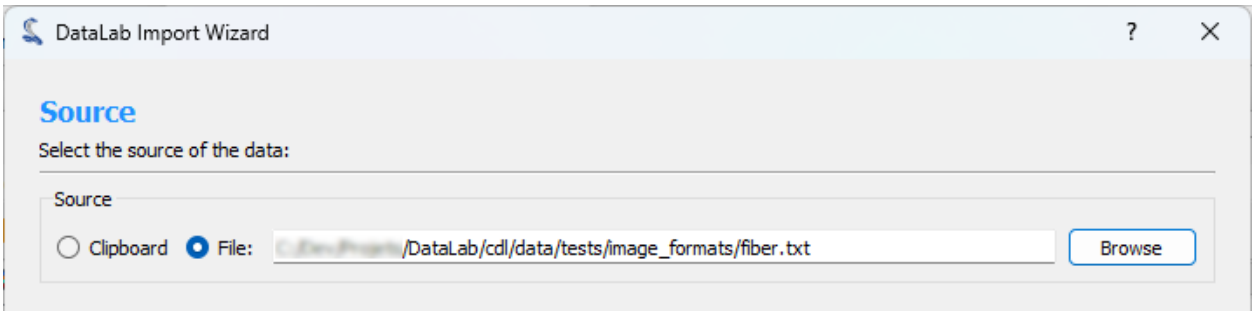


Fig. 44 – Etape 1 : Sélectionner la source

Etape 2 : Aperçu et configuration de l'importation

La deuxième étape consiste à configurer l'importation et à prévisualiser le résultat. Vous pouvez configurer les options suivantes :

- **Délimiteur** : Le caractère utilisé pour séparer les valeurs dans le fichier texte.
- **Commentaires** : Le caractère utilisé pour indiquer que la ligne est un commentaire et doit être ignorée.
- **Lignes à sauter** : Le nombre de lignes à sauter au début du fichier.
- **Nombre maximum de lignes** : Le nombre maximum de lignes à importer. Si le fichier contient plus de lignes, elles seront ignorées.
- **Transposer** : Si coché, les lignes et les colonnes seront transposées.
- **Type de données** : Le type de données de destination des données importées.

Lorsque vous avez terminé de configurer l'importation, cliquez sur le bouton « Appliquer » pour voir le résultat.

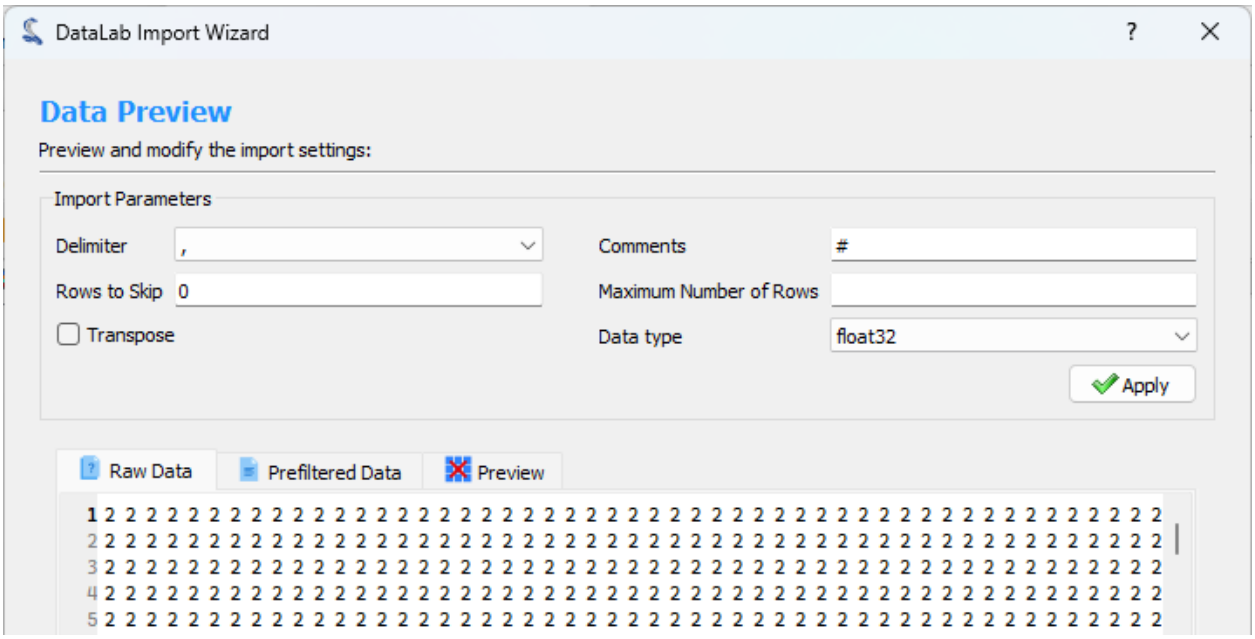


FIG. 45 – Etape 2 : Configurer l'importation

Etape 3 : Afficher la représentation graphique

La troisième étape montre une représentation graphique des données importées. Vous pouvez utiliser le bouton « Terminer » pour importer les données dans l'espace de travail de DataLab.

2.3.3 Détection de pics 2D

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

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).

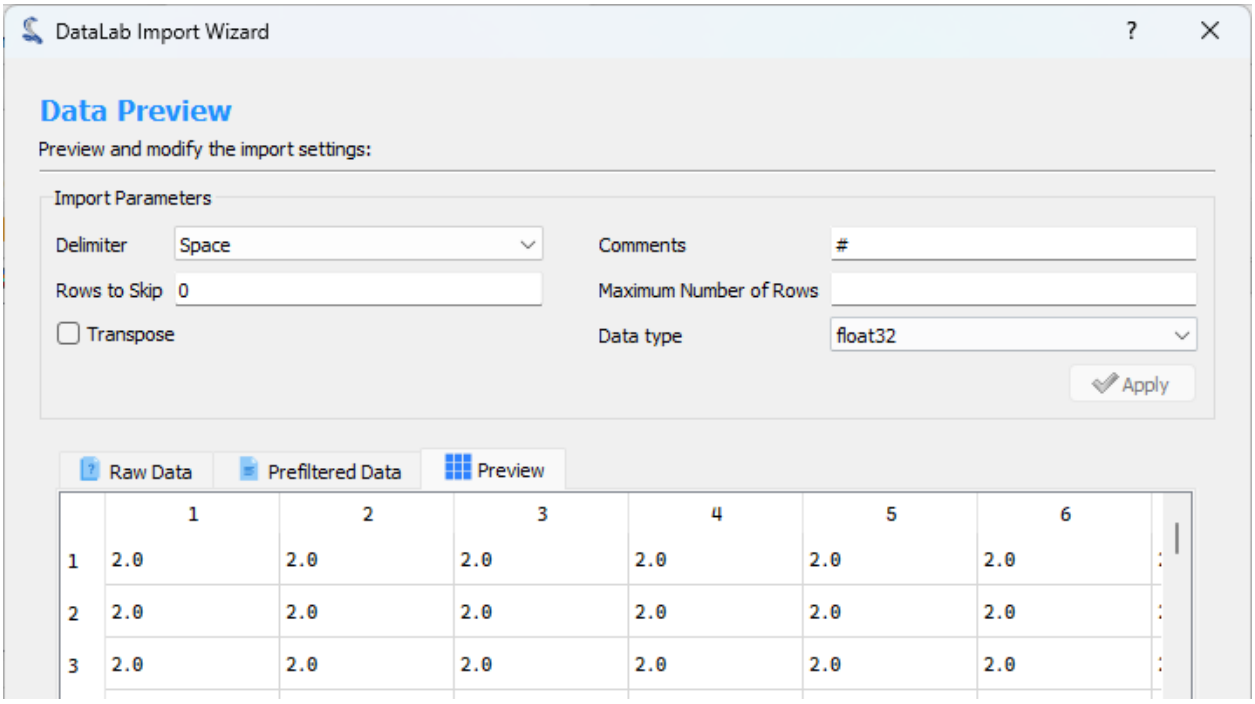


FIG. 46 – Etape 2 : Prévisualiser le résultat

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)
- 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 ensuites obtenues en calculant le centre de chaque étiquette
- Les doublons sont éventuellement supprimés

Les paramètrees 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

La fonctionnalité est basée sur la fonction `get_2d_peaks_coords` du module `cdl.algorithms` :

```
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.
```

(suite sur la page suivante)

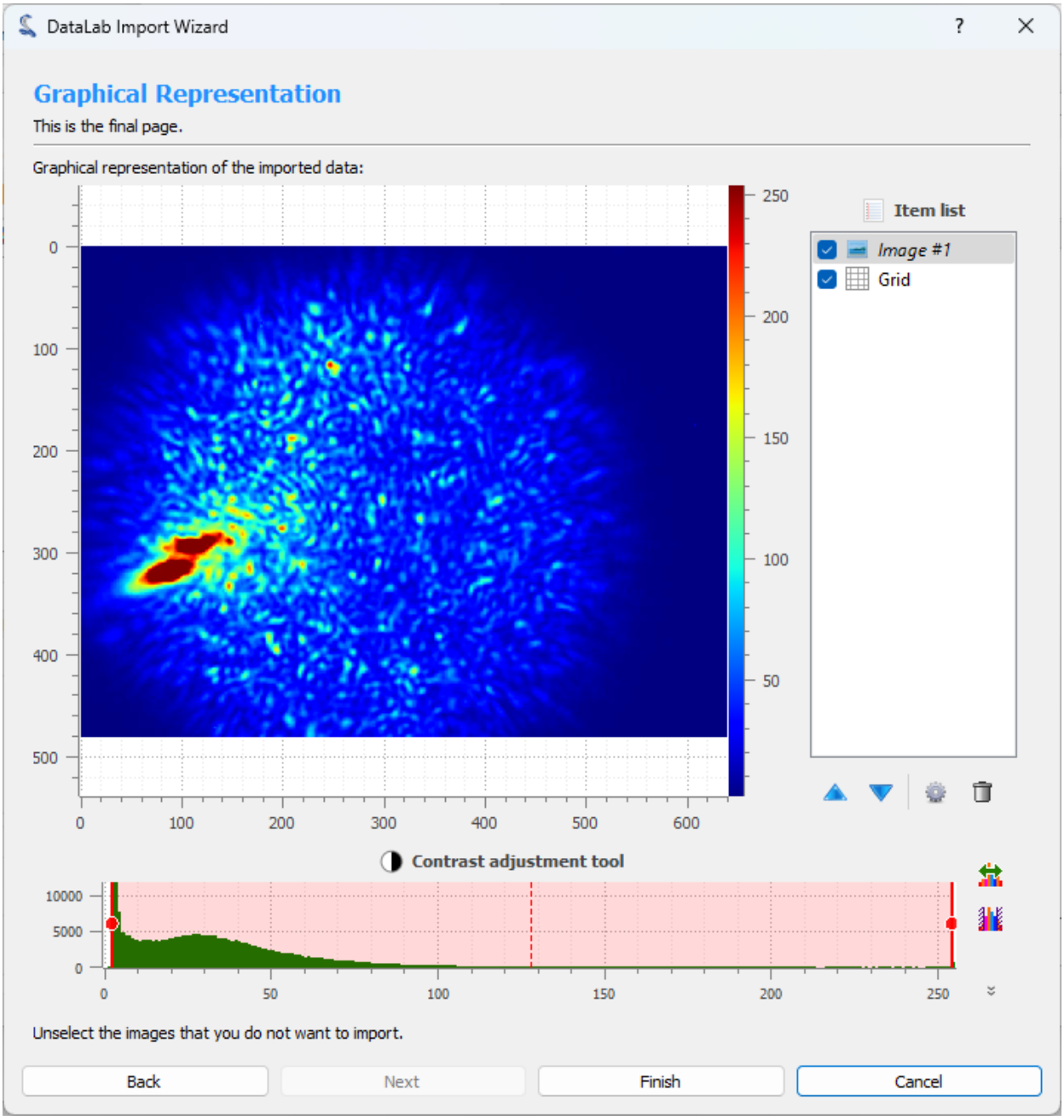


FIG. 47 – Etape 3 : Afficher la représentation graphique

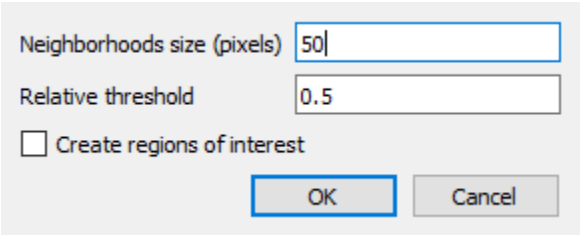


FIG. 48 – Paramètres de détection de pics 2D.

FIG. 49 – Résultats d'une détection de pics 2D (voir le test « peak2d_app.py »)

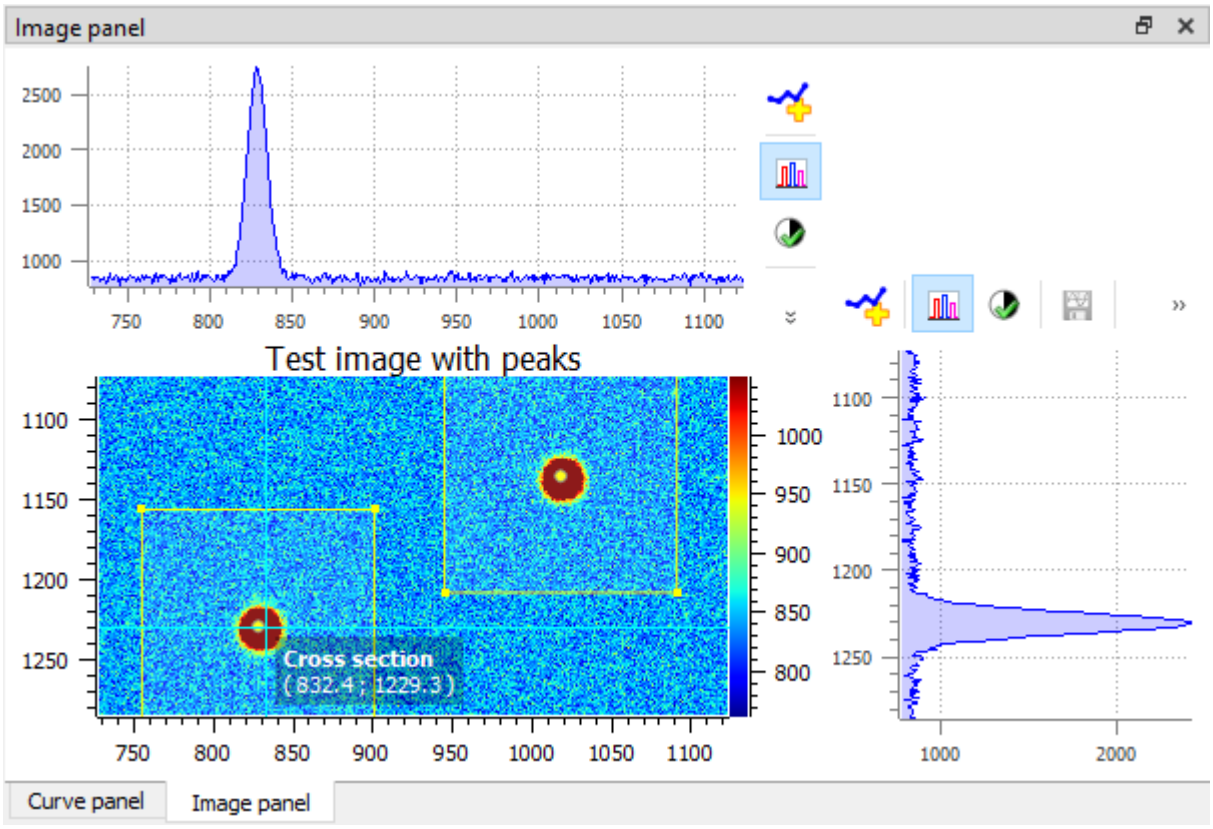


FIG. 50 – Exemple de détection de pics 2D.

(suite de la page précédente)

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

    Args:
        data: Input data
        size: Neighborhood size (default: None)
        level: Relative level (default: 0.5)

    Returns:
        Coordinates of peaks
    """
    if size is None:
        size = max(min(data.shape) // 40, 50)
    data_max = spi.maximum_filter(data, size)
    data_min = spi.minimum_filter(data, size)
    data_diff = data_max - data_min
    diff = (data_max - data_min) > get_absolute_level(data_diff, level)
    maxima = data == data_max
    maxima[diff == 0] = 0
    labeled, _num_objects = spi.label(maxima)
    slices = spi.find_objects(labeled)
    coords = []
    for dy, dx in slices:
        x_center = int(0.5 * (dx.start + dx.stop - 1))
        y_center = int(0.5 * (dy.start + dy.stop - 1))
        coords.append((x_center, y_center))
    if len(coords) > 1:
        # Eventually removing duplicates
        dist = distance_matrix(coords)
        for index in reversed(np.unique(np.where((dist < size) & (dist >␣
↪0))[1])):
            coords.pop(index)
    return np.array(coords)
```

2.3.4 Détection de contours

DataLab fournit une fonctionnalté de « Détection de contours » qui est basée sur l'algorithme des marching cubes

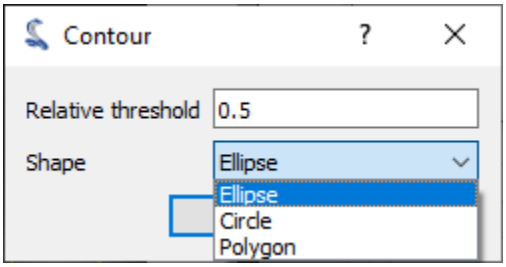


FIG. 51 – Paramètres de détection de contours.

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 »)

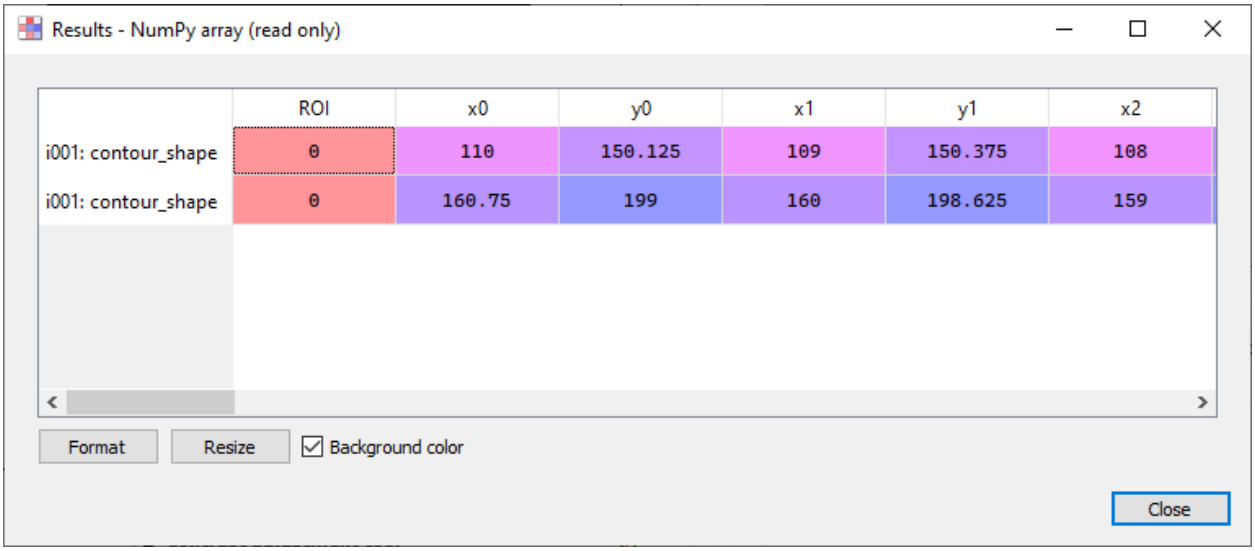


FIG. 52 – Résultats de la détection de contours (cf. test « contour_app.py »)

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` :

```
def get_contour_shapes(
    data: np.ndarray | ma.MaskedArray, 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)
```

(suite sur la page suivante)

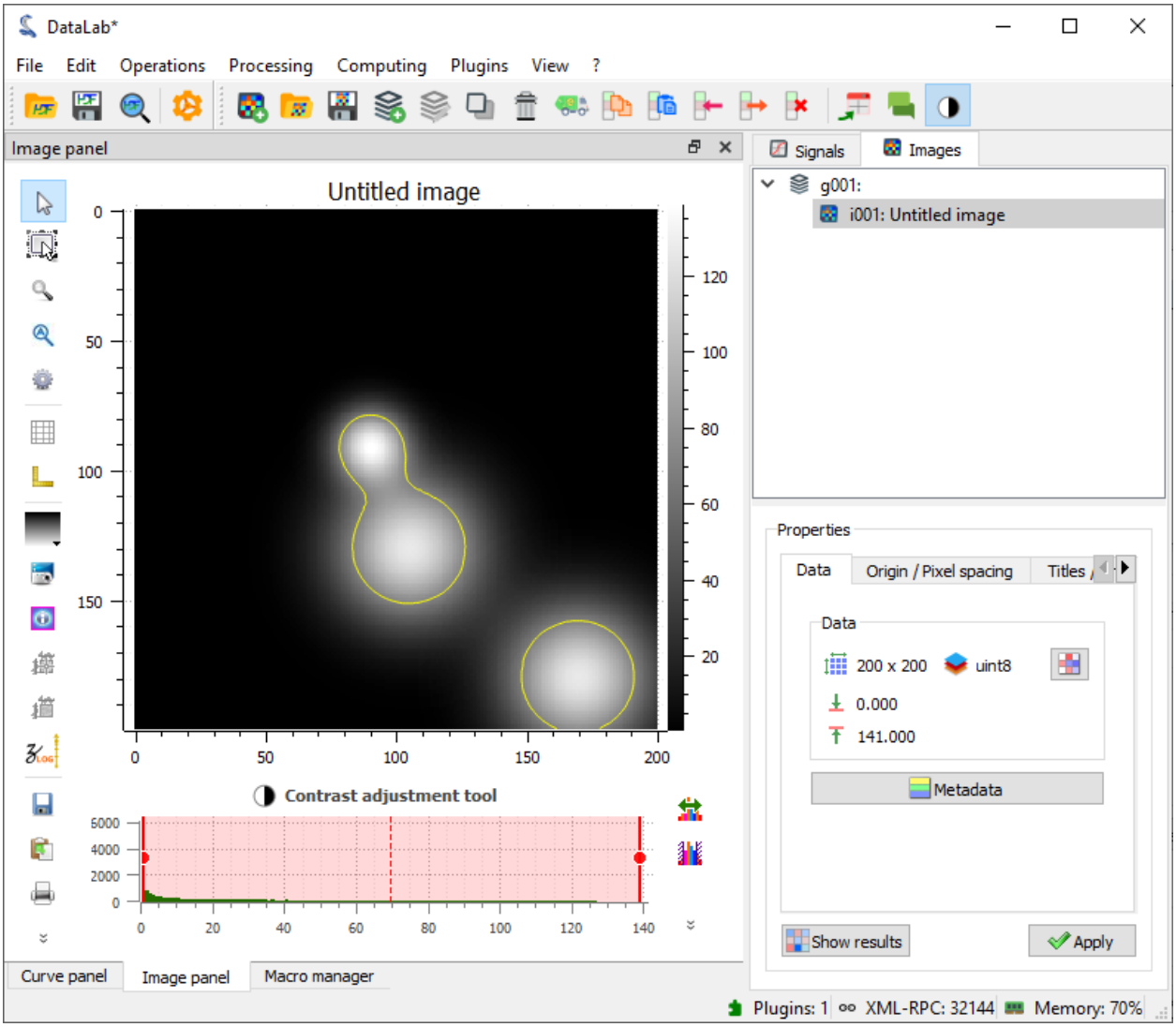


FIG. 53 – Exemple de détection de contours.

(suite de la page précédente)

```
    Returns:
        Coordinates of shapes
    """
    # pylint: disable=too-many-locals
    assert shape in ("circle", "ellipse", "polygon")
    contours = measure.find_contours(data, level=get_absolute_level(data,␣
↪level))
    coords = []
    for contour in contours:
        # `contour` is a (N, 2) array (rows, cols): we need to check if all␣
↪those
        # coordinates are masked: if so, we skip this contour
        if isinstance(data, ma.MaskedArray) and np.all(
            data.mask[contour[:, 0].astype(int), contour[:, 1].astype(int)]
        ):
            continue
        if shape == "circle":
            model = measure.CircleModel()
            if model.estimate(contour):
                yc, xc, r = model.params
                if r <= 1.0:
                    continue
                coords.append([xc, yc, r])
        elif shape == "ellipse":
            model = measure.EllipseModel()
            if model.estimate(contour):
                yc, xc, b, a, theta = model.params
                if a <= 1.0 or b <= 1.0:
                    continue
                coords.append([xc, yc, a, b, theta])
        elif shape == "polygon":
            # `contour` is a (N, 2) array (rows, cols): we need to convert it
            # to a list of x, y coordinates flattened in a single list
            coords.append(contour[:, ::-1].flatten())
        else:
            raise NotImplementedError(f"Invalid contour model {model}")
    if shape == "polygon":
        # `coords` is a list of arrays of shape (N, 2) where N is the number of␣
↪points
        # that can vary from one array to another, so we need to padd with␣
↪NaNs each
        # array to get a regular array:
        max_len = max(coord.shape[0] for coord in coords)
        arr = np.full((len(coords), max_len), np.nan)
        for i_row, coord in enumerate(coords):
            arr[i_row, : coord.shape[0]] = coord
        return arr
    return np.array(coords)
```

2.3.5 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-desous).

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

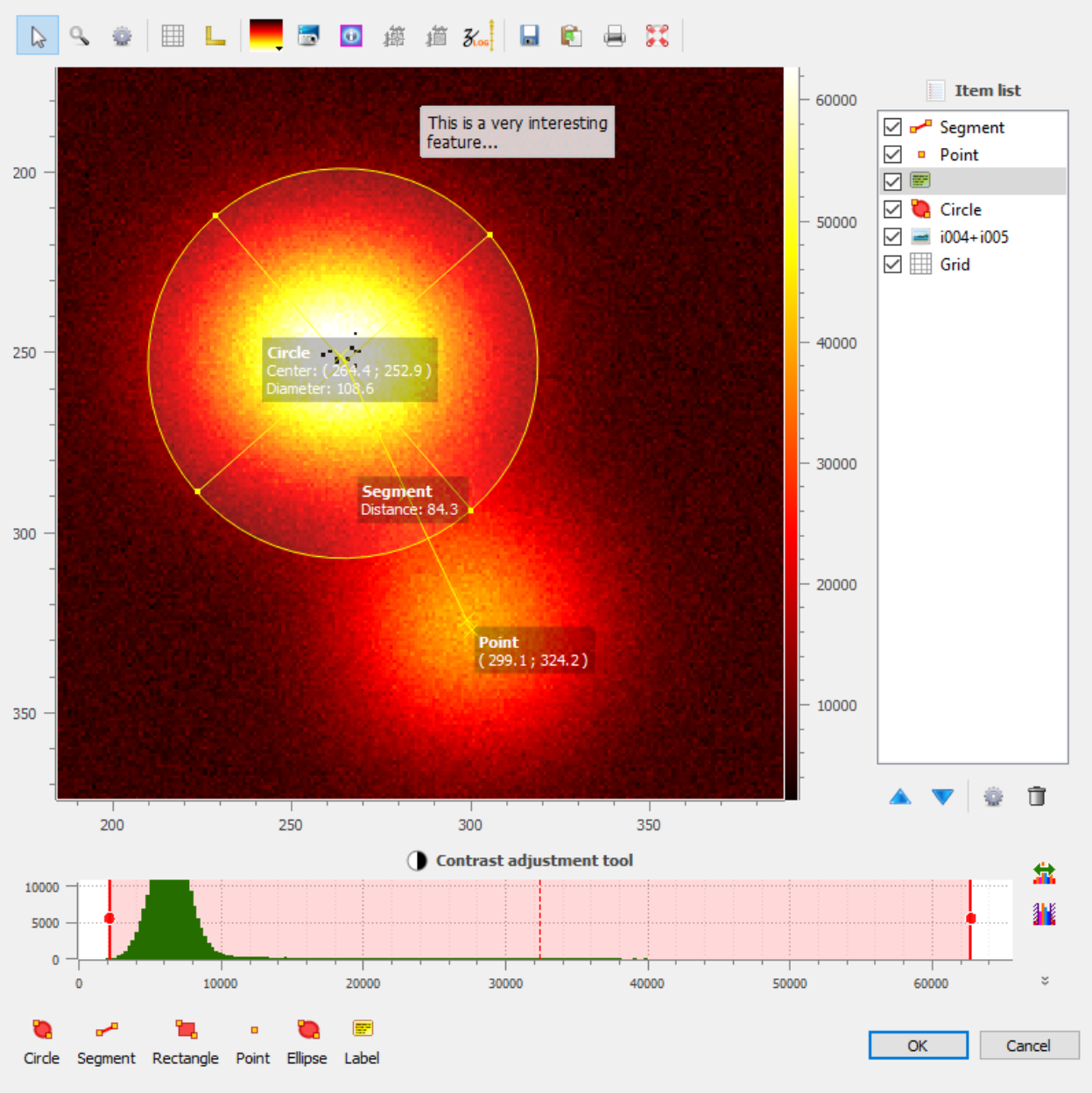


FIG. 54 – Les annotations peuvent être ajoutées dans la fenêtre séparée.

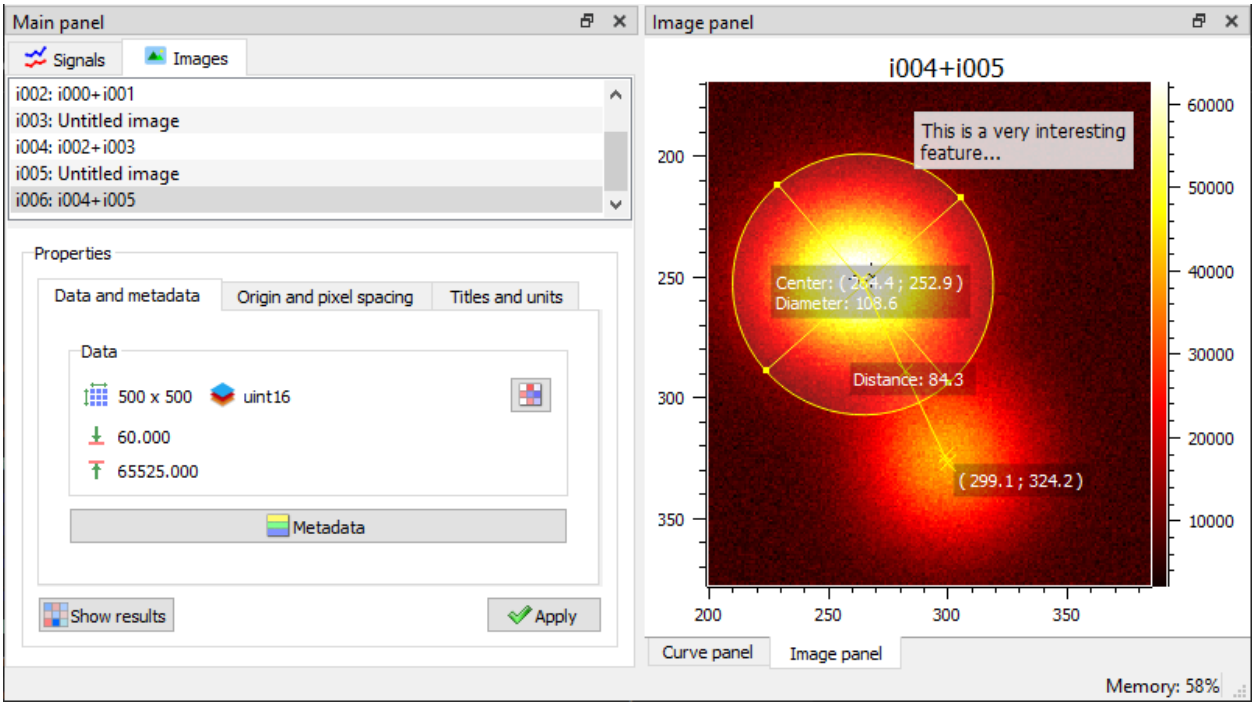


FIG. 55 – Les annotations font maintenant partie des métadonnées de l'image.

CHAPITRE 3

API

L'interface de programmation d'application (API) publique de DataLab offre un ensemble de fonctions pour accéder aux fonctionnalités de DataLab. Ces fonctions sont disponibles dans divers sous-modules du package *cdl*. Le tableau suivant répertorie les sous-modules disponibles et leurs contenus respectifs :

Sous-module	Contenu
cdl.algorithms	Algorithmes pour l'analyse de données, qui opèrent sur des tableaux NumPy
cdl.param	Module de commodité pour accéder aux ensembles de paramètres DataLab (instances d'objets guidata.dataset.DataSet)
cdl.obj	Module de commodité pour accéder aux objets DataLab (*cdl.obj.SignalObj* ou *cdl.obj.ImageObj*) et fonctions associées
cdl.core.computation	Fonctions de calcul, qui opèrent sur les objets DataLab (*cdl.obj.SignalObj* ou *cdl.obj.ImageObj*)
cdl.proxy	Objets proxy pour accéder à l'interface DataLab à partir d'un script Python ou d'une application distante

3.1 Algorithmes (cdl.algorithms)

Ce paquet contient les algorithmes utilisés par le projet DataLab. Ces algorithmes opèrent directement sur des tableaux NumPy et sont conçus pour être utilisés dans le pipeline DataLab, mais peuvent également être utilisés indépendamment.

Voir aussi :

Le module *cdl.algorithms* est le point d'entrée principal pour les algorithmes DataLab lors de la manipulation de tableaux NumPy. Voir le module *cdl.core.computation* pour les algorithmes qui opèrent directement sur les objets DataLab (c'est-à-dire *cdl.obj.SignalObj* et *cdl.obj.ImageObj*).

Les algorithmes sont organisés en sous-paquets en fonction de leur objectif. Les sous-paquets suivants sont disponibles :

- *cdl.algorithms.signal* : Algorithmes de traitement du signal
- *cdl.algorithms.image* : Algorithmes de traitement d'image

— *cdl.algorithms.datatypes* : Algorithmes de conversion de type de données
— *cdl.algorithms.coordinates* : Algorithmes de conversion de coordonnées
— *cdl.algorithms.fit* : Algorithmes de régression

3.1.1 Algorithmes de traitement du signal

cdl.algorithms.signal.**moving_average**(*y : ndarray*, *n : int*) → ndarray

Calculer la moyenne mobile.

Paramètres
— **y** (*numpy.ndarray*) – Tableau d'entrée
— **n** (*int*) – Taille de la fenêtre

Renvoie
Moyenne mobile

Type renvoyé
np.ndarray

cdl.algorithms.signal.**derivative**(*x : ndarray*, *y : ndarray*) → ndarray

Calculer la dérivée numérique.

Paramètres
— **x** (*numpy.ndarray*) – Données X
— **y** (*numpy.ndarray*) – Données Y

Renvoie
Dérivée numérique

Type renvoyé
np.ndarray

cdl.algorithms.signal.**normalize**(*yin : ndarray*, *parameter : str = 'maximum'*) → ndarray

Normaliser le tableau d'entrée à un paramètre donné.

Paramètres
— **yin** (*numpy.ndarray*) – Tableau d'entrée
— **parameter** (*str | None*) – Normalisation du paramètre. Par défaut : « maximum ». Valeurs prises en charge : "maximum", "amplitude", "sum", "energy"

Renvoie
Tableau normalisé

Type renvoyé
np.ndarray

cdl.algorithms.signal.**xy_fft**(*x : ndarray*, *y : ndarray*, *shift : bool = True*) → tuple[ndarray, ndarray]

Calculer la FFT sur les données X,Y.

Paramètres
— **x** (*numpy.ndarray*) – Données X
— **y** (*numpy.ndarray*) – Données Y
— **shift** (*bool | None*) – Décaler la fréquence nulle vers le centre du spectre. Par défaut : True.

Renvoie
Données X,Y

Type renvoyé
tuple[np.ndarray, np.ndarray]

cdl.algorithms.signal.**xy_ifft**(*x : ndarray*, *y : ndarray*, *shift : bool = True*) → tuple[ndarray, ndarray]

Calculer la iFFT sur les données X,Y.

Paramètres
- **x** (*numpy.ndarray*) – Données X
- **y** (*numpy.ndarray*) – Données Y
- **shift** (*bool | None*) – Décaler la fréquence nulle vers le centre du spectre. Par défaut : True.

Renvoie
Données X,Y

Type renvoyé
tuple[np.ndarray, np.ndarray]

cdl.algorithms.signal.**peak_indexes**(*y*, *thres : float = 0.3*, *min_dist : int = 1*, *thres_abs : bool = False*) → ndarray

Routine de détection de pics.

Recherche l'index numérique des pics dans *y* en prenant sa première différence d'ordre. En utilisant les paramètres *thres* et *min_dist*, il est possible de réduire le nombre de pics détectés. *y* doit être signé.

Paramètres
- **y** (*ndarray (signed)*) – Données d'amplitude 1D à rechercher pour les pics.
- **thres** (*float between [0., 1.]*) – Seuil normalisé. Seuls les pics dont l'amplitude est supérieure au seuil seront détectés.
- **min_dist** (*int*) – Distance minimale entre chaque pic détecté. Le pic avec l'amplitude la plus élevée est préféré pour satisfaire cette contrainte.
- **thres_abs** (*boolean*) – Si True, la valeur thres sera interprétée comme une valeur absolue, au lieu d'un seuil normalisé.

Renvoie
Tableau contenant les index numériques des pics qui ont été détectés

Type renvoyé
ndarray

cdl.algorithms.signal.**xpeak**(*x : ndarray*, *y : ndarray*) → float

Retourne la position X du pic par défaut (en supposant un seul pic).

Paramètres
- **x** (*numpy.ndarray*) – Données X
- **y** (*numpy.ndarray*) – Données Y

Renvoie
Position X du pic

Type renvoyé
float

cdl.algorithms.signal.**interpolate**(*x : ndarray*, *y : ndarray*, *xnew : ndarray*, *method : str*, *fill_value : float | None = None*) → ndarray

Interpoler les données.

Paramètres
- **x** (*numpy.ndarray*) – Données X
- **y** (*numpy.ndarray*) – Données Y
- **xnew** (*numpy.ndarray*) – Nouvelles données X
- **method** (*str*) – Méthode d'interpolation. Les valeurs valides sont "linear", "spline", "quadratic", "cubic", "barycentric", "pchip"
- **fill_value** (*float | None*) – Valeur de remplissage. Par défaut : None. Cette valeur est utilisée pour remplir les points demandés en dehors de la plage de données X. Elle n'est utilisée que si l'argument de méthode est "linear", "cubic" ou "pchip".

3.1.2 Algorithmes de traitement d'image

cdl.algorithms.image.**scale_data_to_min_max**(*data : ndarray*, *zmin : float | int*, *zmax : float | int*) → ndarray

Mettre à l'échelle le tableau *data* pour s'adapter à la plage dynamique [zmin, zmax]

Paramètres

— **data** – Données d'entrée
— **zmin** – Valeur minimale des données de sortie
— **zmax** – Valeur maximale des données de sortie

Renvoie

Données mises à l'échelle

cdl.algorithms.image.**z_fft**(*z : ndarray*, *shift : bool = True*) → ndarray

Calculer la FFT du tableau complexe *z*

Paramètres

— **z** – Données d'entrée
— **shift** – Décaler la fréquence nulle vers le centre (par défaut : True)

Renvoie

FFT des données d'entrée

cdl.algorithms.image.**z_ifft**(*z : ndarray*, *shift : bool = True*) → ndarray

Calculer la FFT inverse du tableau complexe *z*

Paramètres

— **z** – Données d'entrée
— **shift** – Décaler la fréquence nulle vers le centre (par défaut : True)

Renvoie

FFT inverse des données d'entrée

cdl.algorithms.image.**binning**(*data : ndarray*, *binning_x : int*, *binning_y : int*, *operation : str*, *dtype=None*) → ndarray

Effectuer le binning des pixels de l'image

Paramètres

— **data** – Données d'entrée
— **binning_x** – Facteur de binning le long de l'axe x
— **binning_y** – Facteur de binning le long de l'axe y
— **operation** – Opération de binning (sum, average, median, min, max)
— **dtype** – Type de données de sortie (par défaut : None, c'est-à-dire identique à l'entrée)

Renvoie

Données binnées

cdl.algorithms.image.**flatfield**(*rawdata : ndarray*, *flatdata : ndarray*, *threshold : float | None = None*) → ndarray

Calculer la correction du champ plat

Paramètres

— **rawdata** – Données brutes
— **flatdata** – Données de champ plat
— **threshold** – Seuil de correction du champ plat (par défaut : None)

Renvoie

Données corrigées du champ plat

cdl.algorithms.image.**get_centroid_fourier**(*data : ndarray*) → tuple[float, float]

Retourne le centre de gravité de l'image en utilisant l'algorithme de Fourier

Paramètres

data – Données d'entrée

Renvoie
Coordonnées du centre de gravité (ligne, colonne)

cdl.algorithms.image.**get_absolute_level**(*data : ndarray*, *level : float*) → float

Retourne le niveau absolu

Paramètres
— **data** – Données d'entrée
— **level** – Niveau relatif (0.0 à 1.0)

Renvoie
Niveau absolu

cdl.algorithms.image.**get_enclosing_circle**(*data : ndarray*, *level : float = 0.5*) → tuple[int, int, float]

Retourne (x, y, rayon) pour le contour du cercle contenant les valeurs d'image au-dessus du niveau relatif du seuil (.5 signifie FWHM)

Paramètres
— **data** – Données d'entrée
— **level** – Niveau relatif (par défaut : 0.5)

Renvoie
Un tuple (x, y, rayon)

Lève
ValueError – Aucun contour n'a été trouvé

cdl.algorithms.image.**get_radial_profile**(*data : ndarray*, *center : tuple[int, int]*) → tuple[ndarray, ndarray]

Retourne le profil radial des données d'image

Paramètres
— **data** – Tableau d'entrée (2D)
— **center** – Coordonnées du centre du profil (x, y)

Renvoie
Profil radial (X, Y) où X est la distance du centre (tableau 1D) et Y est la valeur moyenne des pixels à cette distance (tableau 1D)

cdl.algorithms.image.**distance_matrix**(*coords : list*) → ndarray

Retourne la matrice de distance à partir des coordonnées

Paramètres
coords – Liste des coordonnées

Renvoie
Matrice de distance

cdl.algorithms.image.**get_2d_peaks_coords**(*data : ndarray*, *size : int | None = None*, *level : float = 0.5*) → ndarray

Détecte les pics dans les données d'image, retourne les coordonnées.

Si la taille du voisinage est None, la valeur par défaut est la plus élevée entre 50 pixels et le 1/40ème de la plus petite dimension de l'image.

Le niveau de seuil de détection est relatif à la différence entre les valeurs maximales et minimales des données.

Paramètres
— **data** – Données d'entrée
— **size** – Taille du voisinage (par défaut : None)
— **level** – Niveau relatif (par défaut : 0.5)

Renvoie
Coordonnées des pics

cdl.algorithms.image.**get_contour_shapes**(*data : ndarray | MaskedArray*, *shape : str = 'ellipse'*, *level : float = 0.5*) → ndarray

Trouve les contours de valeur iso dans un tableau 2D, au-dessus du niveau relatif (.5 signifie FWHM), puis ajuste les contours avec la forme ("ellipse" ou "cercle")

Paramètres

- **data** – Données d'entrée
- **shape** – Forme à ajuster. Valeurs valides : "circle", "ellipse", "polygon". (par défaut : "ellipse")
- **level** – Niveau relatif (par défaut : 0.5)

Renvoie

Coordonnées des formes

cdl.algorithms.image.**get_hough_circle_peaks**(*data : ndarray*, *min_radius : float | None = None*, *max_radius : float | None = None*, *nb_radius : int | None = None*, *min_distance : int = 1*) → ndarray

Détecte les pics dans l'image à partir de la transformée de Hough du cercle, retourne les coordonnées du cercle.

Paramètres

- **data** – Données d'entrée
- **min_radius** – Rayon minimum (par défaut : None)
- **max_radius** – Rayon maximum (par défaut : None)
- **nb_radius** – Nombre de rayons (par défaut : None)
- **min_distance** – Distance minimale entre les cercles (par défaut : 1)

Renvoie

Coordonnées des cercles

cdl.algorithms.image.**find_blobs_dog**(*data : ndarray*, *min_sigma : float = 1*, *max_sigma : float = 30*, *overlap : float = 0.5*, *threshold_rel : float = 0.2*, *exclude_border : bool = True*) → ndarray

Recherche les taches dans l'image en niveaux de gris donnée en utilisant la méthode de différence de Gauss (DoG).

Paramètres

- **data** – L'image d'entrée en niveaux de gris.
- **min_sigma** – Le rayon minimum de la tache en pixels.
- **max_sigma** – Le rayon maximum de la tache en pixels.
- **overlap** – Le chevauchement minimum entre deux taches en pixels. Par exemple, si deux taches sont détectées avec des rayons de 10 et 12 respectivement, et que le chevauchement est défini sur 0,5, alors la zone de la plus petite tache sera ignorée et seule la zone de la plus grande tache sera retournée.
- **threshold_rel** – La limite inférieure absolue pour les maxima de l'espace d'échelle. Les maxima locaux plus petits que `threshold_rel` sont ignorés. Réduisez ceci pour détecter les taches avec moins d'intensité.
- **exclude_border** – Si `True`, exclure les taches de la détection si elles sont trop proches du bord de l'image. La taille de la bordure est `min_sigma`.

Renvoie

Coordonnées des taches

cdl.algorithms.image.**find_blobs_doh**(*data : ndarray*, *min_sigma : float = 1*, *max_sigma : float = 30*, *overlap : float = 0.5*, *log_scale : bool = False*, *threshold_rel : float = 0.2*) → ndarray

Recherche les taches dans l'image en niveaux de gris donnée en utilisant la méthode du déterminant de Hessian (DoH).

Paramètres

- **data** – L'image d'entrée en niveaux de gris.

— **min_sigma** – Le rayon minimum de la tache en pixels.
— **max_sigma** – Le rayon maximum de la tache en pixels.
— **overlap** – Le chevauchement minimum entre deux taches en pixels. Par exemple, si deux taches sont détectées avec des rayons de 10 et 12 respectivement, et que le chevauchement est défini sur 0,5, alors la zone de la plus petite tache sera ignorée et seule la zone de la plus grande tache sera retournée.
— **log_scale** – Si `True`, le rayon de chaque tache est retourné comme `sqrt(sigma)` pour chaque tache détectée.
— **threshold_rel** – La limite inférieure absolue pour les maxima de l'espace d'échelle. Les maxima locaux plus petits que `threshold_rel` sont ignorés. Réduisez ceci pour détecter les taches avec moins d'intensité.

Renvoie
Coordonnées des taches

`cdl.algorithms.image.`**find_blobs_log**(*data : ndarray*, *min_sigma : float = 1*, *max_sigma : float = 30*, *overlap : float = 0.5*, *log_scale : bool = False*, *threshold_rel : float = 0.2*, *exclude_border : bool = True*) → ndarray

Recherche les taches dans l'image en niveaux de gris donnée en utilisant la méthode du Laplacien de Gauss (LoG).

Paramètres
— **data** – L'image d'entrée en niveaux de gris.
— **min_sigma** – Le rayon minimum de la tache en pixels.
— **max_sigma** – Le rayon maximum de la tache en pixels.
— **overlap** – Le chevauchement minimum entre deux taches en pixels. Par exemple, si deux taches sont détectées avec des rayons de 10 et 12 respectivement, et que le chevauchement est défini sur 0,5, alors la zone de la plus petite tache sera ignorée et seule la zone de la plus grande tache sera retournée.
— **log_scale** – Si `True`, le rayon de chaque tache est retourné comme `sqrt(sigma)` pour chaque tache détectée.
— **threshold_rel** – La limite inférieure absolue pour les maxima de l'espace d'échelle. Les maxima locaux plus petits que `threshold_rel` sont ignorés. Réduisez ceci pour détecter les taches avec moins d'intensité.
— **exclude_border** – Si `True`, exclure les taches de la détection si elles sont trop proches du bord de l'image. La taille de la bordure est `min_sigma`.

Renvoie
Coordonnées des taches

`cdl.algorithms.image.`**remove_overlapping_disks**(*coords : ndarray*) → ndarray

Supprimer les disques qui se chevauchent parmi les coordonnées

Paramètres
coords – Les coordonnées des disques

Renvoie
Les coordonnées des disques avec les disques qui se chevauchent supprimés

cdl.algorithms.image.**find_blobs_opencv**(*data : ndarray*, *min_threshold : float | None = None*, *max_threshold : float | None = None*, *min_repeatability : int | None = None*, *min_dist_between_blobs : float | None = None*, *filter_by_color : bool | None = None*, *blob_color : int | None = None*, *filter_by_area : bool | None = None*, *min_area : float | None = None*, *max_area : float | None = None*, *filter_by_circularity : bool | None = None*, *min_circularity : float | None = None*, *max_circularity : float | None = None*, *filter_by_inertia : bool | None = None*, *min_inertia_ratio : float | None = None*, *max_inertia_ratio : float | None = None*, *filter_by_convexity : bool | None = None*, *min_convexity : float | None = None*, *max_convexity : float | None = None*) → ndarray

Recherche les taches dans l'image en niveaux de gris donnée en utilisant le détecteur de taches simples d'OpenCV.

Paramètres

- **data** – L'image d'entrée en niveaux de gris.
- **min_threshold** – L'intensité minimale de la tache.
- **max_threshold** – L'intensité maximale de la tache.
- **min_repeatability** – Le nombre minimum de fois qu'une tache est détectée avant qu'elle ne soit signalée.
- **min_dist_between_blobs** – La distance minimale entre les taches.
- **filter_by_color** – Si True, les taches sont filtrées par couleur.
- **blob_color** – La couleur des taches à filtrer par.
- **filter_by_area** – Si True, les taches sont filtrées par zone.
- **min_area** – La zone minimale de la tache.
- **max_area** – La zone maximale de la tache.
- **filter_by_circularity** – Si True, les taches sont filtrées par circularité.
- **min_circularity** – La circularité minimale de la tache.
- **max_circularity** – La circularité maximale de la tache.
- **filter_by_inertia** – Si True, les taches sont filtrées par inertie.
- **min_inertia_ratio** – Le rapport d'inertie minimale de la tache.
- **max_inertia_ratio** – Le rapport d'inertie maximale de la tache.
- **filter_by_convexity** – Si True, les taches sont filtrées par convexité.
- **min_convexity** – La convexité minimale de la tache.
- **max_convexity** – La convexité maximale de la tache.

Renvoie

Coordonnées des taches

3.1.3 Algorithmes de conversion de type de données

cdl.algorithms.datatypes.**is_integer_dtype**(*dtype : dtype*) → bool

Retourne True si le type de données est un type entier

Paramètres

dtype – Type de données à vérifier

Renvoie

True si le type de données est un type entier

cdl.algorithms.datatypes.**is_complex_dtype**(*dtype : dtype*) → bool

Retourne True si le type de données est un type complexe

Paramètres

dtype – Type de données à vérifier

Renvoie

True si le type de données est un type complexe

3.1.4 Algorithmes de conversion de coordonnées

cdl.algorithms.coordinates.**circle_to_diameter**(*xc : float*, *yc : float*, *r : float*) → tuple[float, float, float, float]

Convertit le centre et le rayon du cercle en coordonnées de diamètre X

Paramètres
- **xc** – Abscisse du centre du cercle
- **yc** – Ordonnée du centre du cercle
- **r** – Rayon du cercle

Renvoie
Coordonnées de diamètre X du cercle

Type renvoyé
tuple

cdl.algorithms.coordinates.**array_circle_to_diameter**(*data : ndarray*) → ndarray

Convertit le centre et le rayon du cercle en coordonnées de diamètre X (version tableau)

Paramètres
data – Centre et rayon du cercle, sous la forme d'un tableau 2D (N, 3)

Renvoie
Coordonnées de diamètre X du cercle, sous la forme d'un tableau 2D (N, 4)

cdl.algorithms.coordinates.**circle_to_center_radius**(*x0 : float*, *y0 : float*, *x1 : float*, *y1 : float*) → tuple[float, float, float]

Convertit les coordonnées de diamètre X du cercle en centre et rayon

Paramètres
- **x0** – Abscisse de début de diamètre du cercle
- **y0** – Ordonnée de début de diamètre du cercle
- **x1** – Abscisse de fin de diamètre du cercle
- **y1** – Ordonnée de fin de diamètre du cercle

Renvoie
Centre et rayon du cercle

Type renvoyé
tuple

cdl.algorithms.coordinates.**array_circle_to_center_radius**(*data : ndarray*) → ndarray

Convertit les coordonnées de diamètre X du cercle en centre et rayon (version tableau)

Paramètres
data – Coordonnées de diamètre X du cercle, sous la forme d'un tableau 2D (N, 4)

Renvoie
Centre et rayon du cercle, sous la forme d'un tableau 2D (N, 3)

cdl.algorithms.coordinates.**ellipse_to_diameters**(*xc : float*, *yc : float*, *a : float*, *b : float*, *theta : float*) → tuple[float, float, float, float, float, float, float, float]

Convertit le centre, les axes et l'angle de l'ellipse en coordonnées de diamètres X/Y

Paramètres
- **xc** – Abscisse du centre de l'ellipse
- **yc** – Ordonnée du centre de l'ellipse
- **a** – Demi grand axe de l'ellipse
- **b** – Demi petit axe de l'ellipse
- **theta** – Angle de l'ellipse

Renvoie
Coordonnées des diamètres X/Y (grands/petits axes) de l'ellipse

cdl.algorithms.coordinates.**array_ellipse_to_diameters**(*data : ndarray*) → ndarray

Convertit le centre, les axes et l'angle de l'ellipse en coordonnées de diamètres X/Y (version tableau)

Paramètres
data – Centre, axes et angle de l'ellipse, sous la forme d'un tableau 2D (N, 5)

Renvoie
Coordonnées des diamètres X/Y de l'ellipse (grands/petits axes), sous la forme d'un tableau 2D (N, 8)

cdl.algorithms.coordinates.**ellipse_to_center_axes_angle**(*x0 : float*, *y0 : float*, *x1 : float*, *y1 : float*, *x2 : float*, *y2 : float*, *x3 : float*, *y3 : float*) → tuple[float, float, float, float, float]

Convertit les coordonnées de diamètres X/Y de l'ellipse en centre, axes et angle

Paramètres
- **x0** – abscisse de début du grand axe
- **y0** – ordonnée de début du grand axe
- **x1** – abscisse de fin du grand axe
- **y1** – ordonnée de fin du grand axe
- **x2** – abscisse de début du petit axe
- **y2** – ordonnée de début du petit axe
- **x3** – abscisse de fin du petit axe
- **y3** – ordonnée de fin du petit axe

Renvoie
Centre, axes et angle de l'ellipse

cdl.algorithms.coordinates.**array_ellipse_to_center_axes_angle**(*data : ndarray*) → ndarray

Convertit les coordonnées de diamètres X/Y de l'ellipse en centre, axes et angle (version tableau)

Paramètres
data – Coordonnées de diamètres X/Y de l'ellipse, sous la forme d'un tableau 2D (N, 8)

Renvoie
Centre, axes et angle de l'ellipse, sous la forme d'un tableau 2D (N, 5)

3.1.5 Algorithmes de régression

class cdl.algorithms.fit.**FitModel**

Classe de base du modèle de régression

abstract classmethod func(*x*, *amp*, *sigma*, *x0*, *y0*)

Retourne la fonction de régression

classmethod get_amp_from_amplitude(*amplitude*, *sigma*)

Retourne amp à partir de l'amplitude de la fonction et de sigma

classmethod amplitude(*amp*, *sigma*)

Retourne l'amplitude de la fonction

abstract classmethod fwhm(*amp*, *sigma*)

Retourne la LMH de la fonction

classmethod half_max_segment(*amp*, *sigma*, *x0*, *y0*)

Retourne les coordonnées du segment pour l'intersection y=half-maximum

class cdl.algorithms.fit.**GaussianModel**

Modèle de régression gaussien à 1 dimension

classmethod func(*x*, *amp*, *sigma*, *x0*, *y0*)
Retourne la fonction de régression

classmethod get_amp_from_amplitude(*amplitude*, *sigma*)
Retourne amp à partir de l'amplitude de la fonction et de sigma

classmethod amplitude(*amp*, *sigma*)
Retourne l'amplitude de la fonction

classmethod fwhm(*amp*, *sigma*)
Retourne la LMH de la fonction

class cdl.algorithms.fit.**LorentzianModel**
Modèle de régression lorentzien à 1 dimension

classmethod func(*x*, *amp*, *sigma*, *x0*, *y0*)
Retourne la fonction de régression

classmethod get_amp_from_amplitude(*amplitude*, *sigma*)
Retourne amp à partir de l'amplitude de la fonction et de sigma

classmethod amplitude(*amp*, *sigma*)
Retourne l'amplitude de la fonction

classmethod fwhm(*amp*, *sigma*)
Retourne la LMH de la fonction

class cdl.algorithms.fit.**VoigtModel**
Modèle de régression Voigt à 1 dimension

classmethod func(*x*, *amp*, *sigma*, *x0*, *y0*)
Retourne la fonction de régression

classmethod fwhm(*amp*, *sigma*)
Retourne la LMH de la fonction

3.2 Paramètres (cdl.param)

Le module *cdl.param* fournit tous les paramètres de jeu de données utilisés par les modules *cdl.core.computation* et cdl.core.gui.processor.

Ces jeux de données (sous-classes de guidata.dataset.datatypes.Dataset) sont définis dans d'autres modules :
- *cdl.core.computation.base*
- *cdl.core.computation.image*
- *cdl.core.computation.signal*

Le module *cdl.param* est donc un moyen pratique d'importer tous les ensembles de paramètres en une seule fois.

En fait, l'instruction d'importation suivante est équivalente à la précédente :

```
# Original import statement
from cdl.core.computation.base import MovingAverageParam
from cdl.core.computation.signal import PolynomialFitParam
from cdl.core.computation.image.exposure import EqualizeHistParam

# Equivalent import statement
from cdl.param import MovingAverageParam, PolynomialFitParam, EqualizeHistParam
```

3.2.1 Paramètres communs

class cdl.param.**ClipParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres d'écrêtage

class cdl.param.**FFTParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres FFT

class cdl.param.**GaussianParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres de filtre gaussien

class cdl.param.**MovingAverageParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres de moyenne mobile

class cdl.param.**MovingMedianParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres de médiane mobile

class cdl.param.**ROIDataParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Données de l'éditeur de ROI

classmethod create(*roidata : ndarray | None = None*, *singleobj : bool | None = None*)

Créer une instance de ROIDataParam

property is_empty: bool

Retourne True s'il n'y a pas de ROI

class cdl.param.**ThresholdParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres de seuillage

3.2.2 Paramètres des signaux

class cdl.param.**DataTypeSParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Convertir les paramètres de type de données du signal

class cdl.param.**FWHMParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres de LMH

class cdl.param.**NormalizeYParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres de normalisation

class cdl.param.**PeakDetectionParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres de détection de pics

class cdl.param.**PolynomialFitParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres de régression polynomiale

class cdl.param.**XYCalibrateParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres de calibration du signal

class cdl.param.**InterpolationParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres d'interpolation

class cdl.param.**ResamplingParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres de rééchantillonnage

class cdl.param.**DetrendingParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Elimination de tendance

3.2.3 Paramètres des images

Paramètres de base des images

class cdl.param.**AverageProfileParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres de profil horizontal ou vertical moyen

class cdl.param.**RadialProfileParam**(**args*, ***kwargs*)

Paramètres de profil radial

update_from_image(*obj : ImageObj*) → None

Mettre à jour les paramètres à partir de l'image

choice_callback(*item*, *value*)

Callback pour l'item de choix

class cdl.param.**BinningParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres de binning

class cdl.param.**ButterworthParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres de filtre Butterworth

class cdl.param.**DataTypeIParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Convertir les paramètres de type de données de l'image

class cdl.param.**FlatFieldParam**(*title=None*, *comment=None*, *icon=''*)

Paramètres de correction de champ plat

class cdl.param.**GridParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres de grille

class cdl.param.**HoughCircleParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres de transformation de Hough circulaire

class cdl.param.**LogP1Param**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres de log10

class cdl.param.**ProfileParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres de profil horizontal ou vertical

class cdl.param.**ResizeParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres de redimensionnement

class cdl.param.**RotateParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres de rotation

class cdl.param.**ZCalibrateParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres de calibration linéaire de l'image

Paramètres de correction d'exposition

class cdl.param.**AdjustGammaParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres d'ajustement gamma

class cdl.param.**AdjustLogParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres d'ajustement logarithmique

class cdl.param.**AdjustSigmoidParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres d'ajustement sigmoïde

class cdl.param.**EqualizeAdaptHistParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres d'égalisation d'histogramme adaptatif

class cdl.param.**EqualizeHistParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres d'égalisation d'histogramme

class cdl.param.**RescaleIntensityParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres de rééchelonnement d'intensité

Paramètres de restauration

class cdl.param.**DenoiseBilateralParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres de débruitage par filtre bilatéral

class cdl.param.**DenoiseTVParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres de débruitage par variation totale

class cdl.param.**DenoiseWaveletParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres de débruitage par ondelettes

Paramètres morphologiques

class cdl.param.**MorphologyParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres de White Top-Hat

Paramètres de détection de contours

class cdl.param.**CannyParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres de filtre Canny

Paramètres de détection

class cdl.param.**BlobDOGParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Détection de taches par méthode de différence de gaussiennes

class cdl.param.**BlobDOHParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Détection de taches par méthode du déterminant du Hessien

class cdl.param.**BlobLOGParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Détection de taches par méthode du Laplacien de gaussiennes

class cdl.param.**BlobOpenCVParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Détection de taches par OpenCV

class cdl.param.**ContourShapeParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres de forme de contour

class cdl.param.**Peak2DDetectionParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres de détection de pics

3.3 Modèle de données (cdl.obj)

Le module *cdl.obj* fournit les classes et fonctions nécessaires pour créer et manipuler des objets signal et image.

Ces classes et fonctions sont définies dans d'autres modules :

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

Le module *cdl.obj* est donc un moyen pratique d'importer tous les objets en une seule fois. En fait, l'instruction d'importation suivante est équivalente à la précédente :

```
# Original import statement
from cdl.core.model.signal import SignalObj
from cdl.core.model.image import ImageObj

# Equivalent import statement
from cdl.obj import SignalObj, ImageObj
```

3.3.1 Objets communs

class cdl.obj.**ResultShape**(*shapetype :* ShapeTypes, *array : ndarray*, *label : str = ''*)

Objet représentant une forme géométrique sérialisable dans les métadonnées du signal/image.

Le tableau de résultats *array* est un tableau NumPy 2-D : chaque ligne est un résultat, éventuellement associé à une ROI (valeur de la première colonne).

L'index de la ROI commence à 0 (ou est simplement 0 s'il n'y a pas de ROI).

Paramètres

- **shapetype** – type de forme
- **array** – coordonnées de la forme (plusieurs formes : une forme par ligne), la première colonne est l'index de la ROI (0 s'il n'y a pas de ROI)
- **label** – étiquette de la forme

Lève

AssertionError – argument invalide

classmethod label_shapetype_from_key(*key : str*)

Retourne l'étiquette de la forme et le type de forme à partir de la clé des métadonnées

classmethod from_metadata_entry(*key*, *value*) → *ResultShape* | None

Crée un objet de forme de métadonnées à partir de l'entrée de métadonnées (clé, valeur)

classmethod match(*key*, *value*) → bool

Retourne True si l'entrée du dictionnaire de métadonnées (clé, valeur) est un résultat de métadonnées

property key: str

Retourne la clé de métadonnées associée au résultat

property shown_xlabels: tuple[str]

Retourne les étiquettes pour les colonnes du tableau de résultats

property shown_array: ndarray

Retourne le tableau des résultats affichés, c'est-à-dire incluant le tableau complémentaire

Renvoie

Tableau des résultats affichés

add_to(*obj : BaseObj*)

Ajoute une forme de métadonnées à l'objet (signal/image)

merge_with(*obj : BaseObj*, *other_obj : BaseObj | None = None*)

Fusionne la forme résultat de l'objet avec celle d'un autre : obj <– other_obj ou fusionne simplement cette forme résultat avec obj si other_obj est None

property data_colnb

Retourne le numéro de colonne des résultats de données brutes

is_first_column_roi_index() → bool

Retourne True si la première colonne est l'index de la ROI

property data

Retourne les données brutes (tableau sans informations sur la ROI)

transform_coordinates(*func : Callable[[ndarray], None]*) → None

Transforme les coordonnées de la forme.

Paramètres
func – fonction pour transformer les coordonnées

check_array()

Vérifie si le tableau est valide

iterate_plot_items(*fmt : str*, *lbl : bool*, *option : str*) → Iterable

Itère sur les éléments de tracé de la forme de métadonnées.

Paramètres
— **fmt** – format numérique (par exemple « %.3f »)
— **lbl** – si True, affiche les étiquettes de forme
— **option** – option de style de forme (par exemple « shape/drag »)

Produit (*yield*)
Item graphique

create_plot_item(*args : ndarray*, *fmt : str*, *lbl : bool*, *option : str*)

Crée un élément de tracé.

Paramètres
— **args** – données de forme
— **fmt** – format numérique (par exemple « %.3f »)
— **lbl** – si True, affiche les étiquettes de forme
— **option** – option de style de forme (par exemple « shape/drag »)

Renvoie
Item graphique

make_marker_item(*args*, *fmt*)

Crée un élément de marqueur

class cdl.obj.**ShapeTypes**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Types de formes pour les métadonnées d'image

class cdl.obj.**UniformRandomParam**(*title=None*, *comment=None*, *icon=''*)

Paramètres de signal/image aléatoire de loi uniforme

apply_integer_range(*vmin*, *vmax*)

Fait quelque chose en cas de plage min-max entière

class cdl.obj.**NormalRandomParam**(*title=None*, *comment=None*, *icon=''*)

Paramètres de signal/image aléatoire de loi normale

apply_integer_range(*vmin*, *vmax*)

Fait quelque chose en cas de plage min-max entière

3.3.2 Modèle de signal

class cdl.obj.**SignalObj**(*title=None*, *comment=None*, *icon=''*)

Objet signal

regenerate_uuid()

Regénère l'UUID

Cette méthode est utilisée pour régénérer l'UUID après avoir chargé l'objet à partir d'un fichier. Cela est nécessaire pour éviter les conflits d'UUID lors du chargement d'objets à partir d'un fichier sans effacer l'espace de travail au préalable.

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

Copie l'objet.

Paramètres

— **title** – titre

— **dtype** – type de données

Renvoie

Objet copié

set_data_type(*dtype : dtype*) → None

Change le type de données.

Paramètres

type (*Data*) –

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

Définit les données xy

Paramètres

— **x** – données x

— **y** – données y

— **dx** – données dx (optionnel : barres d'erreur)

— **dy** – données dy (optionnel : barres d'erreur)

property x: ndarray | None

Récupère les données x

property y: ndarray | None

Récupère les données y

property data: ndarray | None

Récupère les données y

property dx: ndarray | None

Récupère les données dx

property dy: ndarray | None

Récupère les données dy

get_data(*roi_index : int | None = None*) → ndarray

Renvoie les données originales (si la ROI n'est pas définie ou si l'*roi_index* est None), ou les données de la ROI (si la ROI et l'*roi_index* sont définis).

Paramètres
roi_index – index de la ROI

Renvoie
Données

update_plot_item_parameters(*item : CurveItem*) → None

Met à jour les paramètres de l'élément de tracé à partir des données/métadonnées de l'objet

Prend en compte un sous-ensemble de paramètres d'élément de tracé. Ces paramètres peuvent avoir été remplacés par des entrées de métadonnées d'objet ou d'autres données d'objet. Le but est de mettre à jour l'élément de tracé en conséquence.

C'est *presque* l'opération inverse de *update_metadata_from_plot_item*.

Paramètres
item – élément de tracé

update_metadata_from_plot_item(*item : CurveItem*) → None

Met à jour les métadonnées à partir de l'élément de tracé.

Prend en compte un sous-ensemble de paramètres d'élément de tracé. Ces paramètres peuvent avoir été modifiés par l'utilisateur via l'interface graphique de l'élément de tracé. Le but est de mettre à jour les métadonnées en conséquence.

C'est *presque* l'opération inverse de *update_plot_item_parameters*.

Paramètres
item – élément de tracé

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

Crée un élément de tracé à partir des données.

Paramètres
update_from – élément de tracé à mettre à jour à partir de

Renvoie
Item graphique

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

Met à jour l'élément de tracé à partir des données.

Paramètres
— **item** – élément de tracé
— **data_changed** – si True, les données ont changé

roi_coords_to_indexes(*coords : list*) → ndarray

Convertit les coordonnées de la ROI en index.

Paramètres
coords – coordonnées

Renvoie
Indices

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

Renvoie le jeu de données de paramètres de la ROI.

Paramètres
— **title** – titre
— ***defaults** – valeurs par défaut

static params_to_roidata(*params : DataSetGroup*) → ndarray

Convertit le groupe de données de la ROI en données de tableau de la ROI.

Paramètres
params – groupe de données de la ROI

Renvoie
données de tableau de la ROI

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

Renvoyer un nouvel élément ROI à partir de zéro

Paramètres
— **fmt** – chaîne de format
— **lbl** – si True, ajouter une étiquette
— **editable** – si True, ROI est modifiable

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

Créer un élément de tracé représentant une région d'intérêt.

Paramètres
— **fmt** – chaîne de format
— **lbl** – si True, ajouter une étiquette
— **editable** – si True, ROI est modifiable

Produit (*yield*)
Item graphique

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

Ajouter une étiquette avec une annotation de titre

Paramètres
title – titre (si None, utiliser le titre du signal)

accept(*vis : object*) → None

Fonction d'aide qui transmet le visiteur aux méthodes accept des éléments dans cet ensemble de données

Paramètres
vis (*object*) – objet visiteur

add_annotations_from_file(*filename : str*) → None

Ajouter des annotations d'objet à partir d'un fichier (JSON).

Paramètres
filename – nom du fichier

add_annotations_from_items(*items : list*) → None

Ajouter des annotations d'objet (éléments de tracé d'annotation).

Paramètres
items – éléments de tracé d'annotation

add_resultshape(*label : str*, *shapetype :* ShapeTypes, *array : ndarray*, *param : DataSet | None = None*) → *ResultShape*

Ajouter une forme géométrique en tant qu'entrée de métadonnées et renvoyer une instance ResultShape.

Paramètres
— **label** – étiquette
— **shapetype** – type de forme
— **array** – tableau
— **param** – paramètres

Renvoie
Forme résultat

property annotations: str

Obtenir les annotations d'objet (chaîne JSON décrivant les éléments de tracé d'annotation)

check() → list[str]

Vérifier les valeurs des éléments de l'ensemble de données

Renvoie
liste des erreurs

Type renvoyé
list[str]

check_data()

Vérifier si les données sont valides, lever une exception si ce n'est pas le cas

Lève
TypeError – si le type de données n'est pas pris en charge

classmethod create(***kwargs*) → DataSet

Créer une nouvelle instance de la classe DataSet

Paramètres
kwargs – arguments clés pour définir les valeurs de DataItem

Renvoie
instance de DataSet

deserialize(*reader : HDF5Reader | JSONReader | INIReader*) → None

Désérialiser l'ensemble de données

Paramètres
reader (*HDF5Reader | JSONReader | INIReader*) – objet lecteur

edit(*parent : QWidget | None = None*, *apply : Callable | None = None*, *wordwrap : bool = True*, *size : QSize | tuple[int, int] | None = None*) → DataSetEditDialog

Ouvrir une boîte de dialogue pour modifier l'ensemble de données

Paramètres
— **parent** – widget parent (par défaut, None signifie pas de parent)
— **apply** – callback d'application (par défaut, None)
— **wordwrap** – si True, le texte du commentaire est enveloppé
— **size** – taille de la boîte de dialogue (objet QSize ou tuple d'entiers (largeur, hauteur))

export_metadata_to_file(*filename : str*) → None

Exporter les métadonnées de l'objet vers un fichier (JSON).

Paramètres
filename – nom du fichier

get_comment() → str | None

Renvoyer le commentaire de l'ensemble de données

Renvoie
commentaire

Type renvoyé
str | None

get_icon() → str | None

Renvoyer l'icône de l'ensemble de données

Renvoie
icône

Type renvoyé
str | None

`get_items`(*copy=False*) → list[DataItem]

Renvoie tous les objets DataItem de l'instance DataSet. Ignore les objets privés qui ont un nom commençant par un tiret bas (par exemple "_private_item = ...")

Paramètres
— **copy** – Si True, effectue une copie profonde de la liste DataItem, sinon renvoie l'original.
— **False.** (*Defaults to*) –

Renvoie
description

`get_metadata_option`(*name : str*) → Any

Renvoyer la valeur de l'option de métadonnées

Une option de métadonnées est une entrée de métadonnées commençant par un underscore. C'est un moyen de stocker des options spécifiques à l'application dans les métadonnées de l'objet.

Paramètres
name – nom de l'option

Renvoie
Valeur de l'option

Noms d'options valides :
"format" : chaîne de format "showlabel" : afficher l'étiquette

`get_title`() → str

Renvoyer le titre de l'ensemble de données

Renvoie
titre

Type renvoyé
str

`classmethod get_valid_dtypenames`() → list[str]

Obtenir les noms de types de données valides

Renvoie
Noms de types de données valides pris en charge par cette classe

`import_metadata_from_file`(*filename : str*) → None

Importer les métadonnées de l'objet à partir d'un fichier (JSON).

Paramètres
filename – nom du fichier

`iterate_resultshapes`()

Itérer sur les formes de résultat de l'objet.

Produit (*yield*)
Forme résultat

`iterate_roi_indexes`()

Itérer sur les index de ROI de l'objet ([0] s'il n'y a pas de ROI)

`iterate_shape_items`(*editable : bool = False*)

Itérer sur les éléments de calcul encodés dans les métadonnées (s'il y en a).

Paramètres
editable – si True, ROI est modifiable

Produit (*yield*)
Item graphique

metadata_to_html() → str

Convertir les métadonnées en chaîne lisible par l'homme.

Renvoie
chaîne HTML

property number: int

Renvoyer le numéro de l'objet (utilisé pour l'ID court)

read_config(*conf : UserConfig*, *section : str*, *option : str*) → None

Lire la configuration à partir d'une instance UserConfig

Paramètres
— **conf** (*UserConfig*) – instance UserConfig
— **section** (*str*) – nom de la section
— **option** (*str*) – nom de l'option

remove_all_shapes() → None

Supprimer les formes de métadonnées et les ROI

reset_metadata_to_defaults() → None

Réinitialiser les métadonnées aux valeurs par défaut

property roi: ndarray | None

Renvoie le tableau des régions d'intérêt de l'objet (une ROI par ligne).

Renvoie
Tableau des régions d'intérêt

roi_has_changed() → bool

Renvoie True si la ROI a changé depuis le dernier appel à cette méthode.

Le premier appel à cette méthode renverra True si la ROI n'a pas encore été définie, ou si la ROI a été définie et a changé depuis le dernier appel à cette méthode. Le prochain appel à cette méthode renverra toujours False si la ROI n'a pas changé entre-temps.

Renvoie
True si la ROI a changé

roidata_to_params(*roidata : ndarray*) → DataSetGroup

Convertit les données du tableau ROI en groupe de jeu de données ROI.

Paramètres
roidata – données de tableau de la ROI

Renvoie
groupe de données de la ROI

serialize(*writer : HDF5Writer | JSONWriter | INIWriter*) → None

Sérialiser le jeu de données

Paramètres
writer (*HDF5Writer | JSONWriter | INIWriter*) – objet écrivain

set_annotations_from_file(*filename : str*) → None

Définir les annotations de l'objet à partir du fichier (JSON).

Paramètres
filename – nom du fichier

set_defaults() → None

Définir les valeurs par défaut

classmethod set_global_prop(*realm : str*, ***kwargs*) → None

Définir les propriétés globales pour tous les éléments de données du jeu de données

Paramètres
- **realm** (*str*) – nom du domaine
- **kwargs** (*dict*) – propriétés à définir

set_metadata_option(*name : str*, *value : Any*) → None

Définir la valeur de l'option de métadonnées

Une option de métadonnées est une entrée de métadonnées commençant par un underscore. C'est un moyen de stocker des options spécifiques à l'application dans les métadonnées de l'objet.

Paramètres
- **name** – nom de l'option
- **value** – valeur de l'option

Noms d'options valides :
"format" : chaîne de format "showlabel" : afficher l'étiquette

property short_id

ID court de l'objet

text_edit() → None

Modifier le jeu de données avec une saisie de texte uniquement

to_string(*debug : bool | None = False*, *indent : str | None = None*, *align : bool | None = False*, *show_hidden : bool | None = True*) → str

Renvoie une représentation lisible du jeu de données. Si debug est True, ajoute plus de détails sur les éléments de données

Paramètres
- **debug** (*bool*) – si True, ajoute plus de détails sur les éléments de données
- **indent** (*str*) – chaîne d'indentation (par défaut, None signifie aucune indentation)
- **align** (*bool*) – si True, aligne les éléments de données (par défaut, False)
- **show_hidden** (*bool*) – si True, affiche les éléments de données masqués (par défaut, True)

Renvoie
représentation sous forme de chaîne du jeu de données

Type renvoyé
str

transform_shapes(*orig*, *func*, *param=None*)

Appliquer la fonction de transformation aux coordonnées de la forme / des annotations du résultat.

Paramètres
- **orig** – objet d'origine
- **func** – fonction de transformation
- **param** – paramètre de la fonction de transformation

update_metadata_view_settings() → None

Mettre à jour les paramètres d'affichage des métadonnées à partir de Conf.view

update_resultshapes_from(*other : BaseObj*) → None

Mettre à jour la forme géométrique à partir d'un autre objet (fusionner les métadonnées).

Paramètres
other – autre objet, à partir duquel mettre à jour cet objet

view(*parent : QWidget | None = None*, *wordwrap : bool = True*, *size : QSize | tuple[int, int] | None = None*) → None

Ouvrir une boîte de dialogue pour afficher le jeu de données

Paramètres

— **parent** – widget parent (par défaut, None signifie pas de parent)
— **wordwrap** – si True, le texte du commentaire est enveloppé
— **size** – taille de la boîte de dialogue (objet QSize ou tuple d'entiers (largeur, hauteur))

write_config(*conf : UserConfig*, *section : str*, *option : str*) → None

Écrire la configuration dans une instance UserConfig

Paramètres

— **conf** (*UserConfig*) – instance UserConfig
— **section** (*str*) – nom de la section
— **option** (*str*) – nom de l'option

cdl.obj.**read_signal**(*filename : str*) → *SignalObj*

Lire un signal à partir d'un fichier.

Paramètres

filename (*str*) – Nom de fichier.

Renvoie

Signal.

Type renvoyé

Signal

cdl.obj.**create_signal**(*title : str*, *x : ndarray | None = None*, *y : ndarray | None = None*, *dx : ndarray | None = None*, *dy : ndarray | None = None*, *metadata : dict | None = None*, *units : tuple[str, str] | None = None*, *labels : tuple[str, str] | None = None*) → *SignalObj*

Créer un nouvel objet Signal.

Paramètres

— **title** – titre du signal
— **x** – données X
— **y** – données Y
— **dx** – données dX (facultatif : barres d'erreur)
— **dy** – données dY (facultatif : barres d'erreur)
— **metadata** – métadonnées du signal
— **units** – unités X, Y (tuple de chaînes)
— **labels** – étiquettes X, Y (tuple de chaînes)

Renvoie

Objet signal

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

Créer un nouvel objet Signal à partir d'une boîte de dialogue.

Paramètres

— **newparam** – nouveaux paramètres du signal
— **addparam** – paramètres supplémentaires
— **edit** – Ouvrir une boîte de dialogue pour modifier les paramètres (par défaut : False)
— **parent** – widget parent

Renvoie

Objet Signal ou None si annulé

cdl.obj.**new_signal_param**(*title : str | None = None*, *stype : str | None = None*, *xmin : float | None = None*, *xmax : float | None = None*, *size : int | None = None*) → *NewSignalParam*

Créer une nouvelle instance de jeu de données Signal.

Paramètres
- **title** – titre du jeu de données (par défaut : None, utilise le titre par défaut)
- **stype** – type de signal (par défaut : None, utilise le type par défaut)
- **xmin** – X min (par défaut : None, utilise la valeur par défaut)
- **xmax** – X max (par défaut : None, utilise la valeur par défaut)
- **size** – taille du signal (par défaut : None, utilise la valeur par défaut)

Renvoie
nouvelle instance de jeu de données signal

Type renvoyé
NewSignalParam

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

Types de signal

class cdl.obj.**NewSignalParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Nouveau jeu de données signal

class cdl.obj.**GaussLorentzVoigtParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres pour les fonctions gaussiennes et lorentziennes

class cdl.obj.**StepParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres pour la fonction d'étape

class cdl.obj.**PeriodicParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres pour les fonctions périodiques

get_frequency_in_hz()

Renvoie la fréquence en Hz

3.3.3 Modèle d'image

class cdl.obj.**ImageObj**(*title=None*, *comment=None*, *icon=''*)

Objet image

regenerate_uuid()

Regénère l'UUID

Cette méthode est utilisée pour régénérer l'UUID après avoir chargé l'objet à partir d'un fichier. Cela est nécessaire pour éviter les conflits d'UUID lors du chargement d'objets à partir d'un fichier sans effacer l'espace de travail au préalable.

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

Définir les métadonnées à partir de l'objet : semblable à un dictionnaire

Paramètres
obj – objet

property dicom_template

Obtenir le modèle DICOM

property xc: float

Renvoie la coordonnée X du centre de l'image

property yc: float

Renvoie la coordonnée Y du centre de l'image

get_data(*roi_index : int | None = None*) → ndarray

Renvoie les données originales (si la ROI n'est pas définie ou si l'*roi_index* est None), ou les données de la ROI (si la ROI et l'*roi_index* sont définis).

Paramètres
roi_index – index de la ROI

Renvoie
Données masquées

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

Copie l'objet.

Paramètres
— **title** – titre
— **dtype** – type de données

Renvoie
Objet copié

set_data_type(*dtype : dtype*) → None

Change le type de données.

Paramètres
type (*Data*) –

update_plot_item_parameters(*item : MaskedImageItem*) → None

Met à jour les paramètres de l'élément de tracé à partir des données/métadonnées de l'objet

Prend en compte un sous-ensemble de paramètres d'élément de tracé. Ces paramètres peuvent avoir été remplacés par des entrées de métadonnées d'objet ou d'autres données d'objet. Le but est de mettre à jour l'élément de tracé en conséquence.

C'est *presque* l'opération inverse de *update_metadata_from_plot_item*.

Paramètres
item – élément de tracé

update_metadata_from_plot_item(*item : MaskedImageItem*) → None

Met à jour les métadonnées à partir de l'élément de tracé.

Prend en compte un sous-ensemble de paramètres d'élément de tracé. Ces paramètres peuvent avoir été modifiés par l'utilisateur via l'interface graphique de l'élément de tracé. Le but est de mettre à jour les métadonnées en conséquence.

C'est *presque* l'opération inverse de *update_plot_item_parameters*.

Paramètres
item – élément de tracé

make_item(*update_from : MaskedImageItem | None = None*) → MaskedImageItem

Crée un élément de tracé à partir des données.

Paramètres
update_from – mettre à jour à partir de l'élément de tracé

Renvoie
Item graphique

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

Met à jour l'élément de tracé à partir des données.

Paramètres
— **item** – élément de tracé
— **data_changed** – si True, les données ont changé

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

Renvoie le jeu de données de paramètres de la ROI.

Paramètres
— **title** – titre
— ***defaults** – valeurs par défaut

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

Convertit le groupe de données de la ROI en données de tableau de la ROI.

Paramètres
params – groupe de données de la ROI

Renvoie
données de tableau de la ROI

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

Renvoyer un nouvel élément ROI à partir de zéro

Paramètres
— **fmt** – chaîne de format
— **lbl** – si True, ajouter une étiquette
— **editable** – si True, ROI est modifiable
— **geometry** – géométrie de la ROI

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

Convertit les coordonnées de la ROI en index.

Paramètres
coords – coordonnées

Renvoie
Indices

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

Créer un élément de tracé représentant une région d'intérêt.

Paramètres
— **fmt** – chaîne de format
— **lbl** – si True, ajouter une étiquette
— **editable** – si True, ROI est modifiable

Produit (*yield*)
Item graphique

`property maskdata: ndarray`

Renvoie les données masquées (zones en dehors des régions d'intérêt définies)

Renvoie
Données masquées

`get_masked_view`() → MaskedArray

Retourne une vue masquée des données

Renvoie
Vue masquée

invalidate_maskdata_cache() → None

Invalider le cache des données masquées : force la reconstruction

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

Ajouter une étiquette avec une annotation de titre

Paramètres
title – titre (si None, utilise le titre de l'image)

accept(*vis : object*) → None

Fonction d'aide qui transmet le visiteur aux méthodes accept des éléments dans cet ensemble de données

Paramètres
vis (*object*) – objet visiteur

add_annotations_from_file(*filename : str*) → None

Ajouter des annotations d'objet à partir d'un fichier (JSON).

Paramètres
filename – nom du fichier

add_annotations_from_items(*items : list*) → None

Ajouter des annotations d'objet (éléments de tracé d'annotation).

Paramètres
items – éléments de tracé d'annotation

add_resultshape(*label : str*, *shapetype :* ShapeTypes, *array : ndarray*, *param : DataSet | None = None*) → *ResultShape*

Ajouter une forme géométrique en tant qu'entrée de métadonnées et renvoyer une instance ResultShape.

Paramètres
— **label** – étiquette
— **shapetype** – type de forme
— **array** – tableau
— **param** – paramètres

Renvoie
Forme résultat

property annotations: str

Obtenir les annotations d'objet (chaîne JSON décrivant les éléments de tracé d'annotation)

check() → list[str]

Vérifier les valeurs des éléments de l'ensemble de données

Renvoie
liste des erreurs

Type renvoyé
list[str]

check_data()

Vérifier si les données sont valides, lever une exception si ce n'est pas le cas

Lève
TypeError – si le type de données n'est pas pris en charge

classmethod create(***kwargs*) → DataSet

Créer une nouvelle instance de la classe DataSet

Paramètres
kwargs – arguments clés pour définir les valeurs de DataItem

Renvoie
instance de DataSet

deserialize(*reader : HDF5Reader | JSONReader | INIReader*) → None

Désérialiser l'ensemble de données

Paramètres
reader (*HDF5Reader | JSONReader | INIReader*) – objet lecteur

edit(*parent : QWidget | None = None*, *apply : Callable | None = None*, *wordwrap : bool = True*, *size : QSize | tuple[int, int] | None = None*) → DataSetEditDialog

Ouvrir une boîte de dialogue pour modifier l'ensemble de données

Paramètres
— **parent** – widget parent (par défaut, None signifie pas de parent)
— **apply** – callback d'application (par défaut, None)
— **wordwrap** – si True, le texte du commentaire est enveloppé
— **size** – taille de la boîte de dialogue (objet QSize ou tuple d'entiers (largeur, hauteur))

export_metadata_to_file(*filename : str*) → None

Exporter les métadonnées de l'objet vers un fichier (JSON).

Paramètres
filename – nom du fichier

get_comment() → str | None

Renvoyer le commentaire de l'ensemble de données

Renvoie
commentaire

Type renvoyé
str | None

get_icon() → str | None

Renvoyer l'icône de l'ensemble de données

Renvoie
icône

Type renvoyé
str | None

get_items(*copy=False*) → list[DataItem]

Renvoie tous les objets DataItem de l'instance DataSet. Ignore les objets privés qui ont un nom commençant par un tiret bas (par exemple "_private_item = ...")

Paramètres
— **copy** – Si True, effectue une copie profonde de la liste DataItem, sinon renvoie l'original.
— **False.** (*Defaults to*) –

Renvoie
description

get_metadata_option(*name : str*) → Any

Renvoyer la valeur de l'option de métadonnées

Une option de métadonnées est une entrée de métadonnées commençant par un underscore. C'est un moyen de stocker des options spécifiques à l'application dans les métadonnées de l'objet.

Paramètres
name – nom de l'option

Renvoie
Valeur de l'option

Noms d'options valides :
"format" : chaîne de format "showlabel" : afficher l'étiquette

`get_title()` → str

Renvoyer le titre de l'ensemble de données

Renvoie
titre

Type renvoyé
str

`classmethod get_valid_dtypenames()` → list[str]

Obtenir les noms de types de données valides

Renvoie
Noms de types de données valides pris en charge par cette classe

`import_metadata_from_file(`*filename : str*`)` → None

Importer les métadonnées de l'objet à partir d'un fichier (JSON).

Paramètres
filename – nom du fichier

`iterate_resultshapes()`

Itérer sur les formes de résultat de l'objet.

Produit (*yield*)
Forme résultat

`iterate_roi_indexes()`

Itérer sur les index de ROI de l'objet ([0] s'il n'y a pas de ROI)

`iterate_shape_items(`*editable : bool = False*`)`

Itérer sur les éléments de calcul encodés dans les métadonnées (s'il y en a).

Paramètres
editable – si True, ROI est modifiable

Produit (*yield*)
Item graphique

`metadata_to_html()` → str

Convertir les métadonnées en chaîne lisible par l'homme.

Renvoie
chaîne HTML

`property number: int`

Renvoyer le numéro de l'objet (utilisé pour l'ID court)

`read_config(`*conf : UserConfig*, *section : str*, *option : str*`)` → None

Lire la configuration à partir d'une instance UserConfig

Paramètres
— **conf** (*UserConfig*) – instance UserConfig
— **section** (*str*) – nom de la section
— **option** (*str*) – nom de l'option

remove_all_shapes() → None

Supprimer les formes de métadonnées et les ROI

reset_metadata_to_defaults() → None

Réinitialiser les métadonnées aux valeurs par défaut

property roi: ndarray | None

Renvoie le tableau des régions d'intérêt de l'objet (une ROI par ligne).

Renvoie
Tableau des régions d'intérêt

roi_has_changed() → bool

Renvoie True si la ROI a changé depuis le dernier appel à cette méthode.

Le premier appel à cette méthode renverra True si la ROI n'a pas encore été définie, ou si la ROI a été définie et a changé depuis le dernier appel à cette méthode. Le prochain appel à cette méthode renverra toujours False si la ROI n'a pas changé entre-temps.

Renvoie
True si la ROI a changé

roidata_to_params(*roidata : ndarray*) → DataSetGroup

Convertit les données du tableau ROI en groupe de jeu de données ROI.

Paramètres
roidata – données de tableau de la ROI

Renvoie
groupe de données de la ROI

serialize(*writer : HDF5Writer | JSONWriter | INIWriter*) → None

Sérialiser le jeu de données

Paramètres
writer (*HDF5Writer | JSONWriter | INIWriter*) – objet écrivain

set_annotations_from_file(*filename : str*) → None

Définir les annotations de l'objet à partir du fichier (JSON).

Paramètres
filename – nom du fichier

set_defaults() → None

Définir les valeurs par défaut

classmethod set_global_prop(*realm : str*, ***kwargs*) → None

Définir les propriétés globales pour tous les éléments de données du jeu de données

Paramètres
— **realm** (*str*) – nom du domaine
— **kwargs** (*dict*) – propriétés à définir

set_metadata_option(*name : str*, *value : Any*) → None

Définir la valeur de l'option de métadonnées

Une option de métadonnées est une entrée de métadonnées commençant par un underscore. C'est un moyen de stocker des options spécifiques à l'application dans les métadonnées de l'objet.

Paramètres
— **name** – nom de l'option
— **value** – valeur de l'option

Noms d'options valides :
"format" : chaîne de format "showlabel" : afficher l'étiquette

property short_id

ID court de l'objet

text_edit() → None

Modifier le jeu de données avec une saisie de texte uniquement

to_string(*debug : bool | None = False*, *indent : str | None = None*, *align : bool | None = False*, *show_hidden : bool | None = True*) → str

Renvoie une représentation lisible du jeu de données. Si debug est True, ajoute plus de détails sur les éléments de données

Paramètres
- **debug** (*bool*) – si True, ajoute plus de détails sur les éléments de données
- **indent** (*str*) – chaîne d'indentation (par défaut, None signifie aucune indentation)
- **align** (*bool*) – si True, aligne les éléments de données (par défaut, False)
- **show_hidden** (*bool*) – si True, affiche les éléments de données masqués (par défaut, True)

Renvoie
représentation sous forme de chaîne du jeu de données

Type renvoyé
str

transform_shapes(*orig*, *func*, *param=None*)

Appliquer la fonction de transformation aux coordonnées de la forme / des annotations du résultat.

Paramètres
- **orig** – objet d'origine
- **func** – fonction de transformation
- **param** – paramètre de la fonction de transformation

update_metadata_view_settings() → None

Mettre à jour les paramètres d'affichage des métadonnées à partir de Conf.view

update_resultshapes_from(*other : BaseObj*) → None

Mettre à jour la forme géométrique à partir d'un autre objet (fusionner les métadonnées).

Paramètres
other – autre objet, à partir duquel mettre à jour cet objet

view(*parent : QWidget | None = None*, *wordwrap : bool = True*, *size : QSize | tuple[int, int] | None = None*) → None

Ouvrir une boîte de dialogue pour afficher le jeu de données

Paramètres
- **parent** – widget parent (par défaut, None signifie pas de parent)
- **wordwrap** – si True, le texte du commentaire est enveloppé
- **size** – taille de la boîte de dialogue (objet QSize ou tuple d'entiers (largeur, hauteur))

write_config(*conf : UserConfig*, *section : str*, *option : str*) → None

Écrire la configuration dans une instance UserConfig

Paramètres
- **conf** (*UserConfig*) – instance UserConfig
- **section** (*str*) – nom de la section
- **option** (*str*) – nom de l'option

cdl.obj.**read_image**(*filename : str*) → *ImageObj*

Lire une image à partir d'un fichier.

Paramètres
filename (*str*) – Nom de fichier.

Renvoie
Image.

Type renvoyé
Image

cdl.obj.**create_image**(*title : str*, *data : ndarray | None = None*, *metadata : dict | None = None*, *units : tuple | None = None*, *labels : tuple | None = None*) → *ImageObj*

Créer un nouvel objet Image

Paramètres
— **title** – titre de l'image
— **data** – données de l'image
— **metadata** – métadonnées de l'image
— **units** – unités X, Y, Z (tuple de chaînes)
— **labels** – étiquettes X, Y, Z (tuple de chaînes)

Renvoie
Objet image

cdl.obj.**create_image_from_param**(*newparam :* NewImageParam, *addparam : gds.DataSet | None = None*, *edit : bool = False*, *parent : QW.QWidget | None = None*) → *ImageObj* | None

Créer un nouvel objet Image à partir d'une boîte de dialogue.

Paramètres
— **newparam** – nouveaux paramètres de l'image
— **addparam** – paramètres supplémentaires
— **edit** – Ouvrir une boîte de dialogue pour modifier les paramètres (par défaut : False)
— **parent** – widget parent

Renvoie
Nouvel objet image ou None si l'utilisateur a annulé

cdl.obj.**new_image_param**(*title : str | None = None*, *itype :* ImageTypes *| None = None*, *height : int | None = None*, *width : int | None = None*, *dtype :* ImageDatatypes *| None = None*) → *NewImageParam*

Créer une nouvelle instance de jeu de données Image.

Paramètres
— **title** – titre du jeu de données (par défaut : None, utilise le titre par défaut)
— **itype** – type d'image (par défaut : None, utilise le type par défaut)
— **height** – hauteur de l'image (par défaut : None, utilise la hauteur par défaut)
— **width** – largeur de l'image (par défaut : None, utilise la largeur par défaut)
— **dtype** – type de données de l'image (par défaut : None, utilise le type de données par défaut)

Renvoie
Nouvelle instance de jeu de données image

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

Types d'image

class cdl.obj.**NewImageParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Nouveau jeu de données image

class cdl.obj.**Gauss2DParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres gaussiens 2D

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

Types de géométrie des données ROI

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

Types de données d'image

classmethod from_dtype(*dtype*)

Renvoie le membre à partir du dtype NumPy

classmethod check()

Vérifie si les types de données sont valides

3.4 Calcul (cdl.core.computation)

Ce paquet contient les fonctions de calcul utilisées par le projet DataLab. Ces fonctions opèrent directement sur les objets DataLab (c'est-à-dire *cdl.obj.SignalObj* et *cdl.obj.ImageObj*) et sont conçues pour être utilisées dans le pipeline DataLab, mais peuvent également être utilisées indépendamment.

Voir aussi :

Le module *cdl.core.computation* est le point d'entrée principal pour les fonctions de calcul DataLab lors de la manipulation d'objets DataLab. Voir le module *cdl.algorithms* pour les algorithmes qui opèrent directement sur les tableaux NumPy.

Chaque module de calcul définit un ensemble d'objets de calcul, c'est-à-dire des fonctions qui implémentent des fonctionnalités de traitement et des classes qui implémentent les paramètres correspondants (sous la forme de sous-classes `guidata.dataset.datatypes.Dataset`). Les fonctions de calcul prennent un objet DataLab (par exemple *cdl.obj.SignalObj*) et un objet de paramètre (par exemple `cdl.param.MovingAverageParam`) en entrée et renvoient un objet DataLab en sortie (le résultat du calcul). L'objet de paramètre est utilisé pour configurer la fonction de calcul (par exemple, la taille de la fenêtre de moyenne mobile).

Dans l'architecture globale de DataLab, le but de ce paquet est de fournir les fonctions de calcul qui sont utilisées par le module `cdl.core.gui.processor`, basé sur les algorithmes définis dans le module *cdl.algorithms* et sur le modèle de données défini dans le module *cdl.obj* (ou `cdl.core.model`).

Les modules de calcul sont organisés en sous-paquets en fonction de leur objectif. Les sous-paquets suivants sont disponibles :

- *cdl.core.computation.base* : fonctionnalités de traitement communes
- *cdl.core.computation.signal* : fonctionnalités de traitement du signal
- *cdl.core.computation.image* : fonctionnalités de traitement d'image

3.4.1 Fonctionnalités de traitement communes

class cdl.core.computation.base.**GaussianParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres du filtre gaussien

class cdl.core.computation.base.**MovingAverageParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres de la moyenne mobile

class cdl.core.computation.base.**MovingMedianParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres de la médiane mobile

class cdl.core.computation.base.**ThresholdParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres du seuillage

class cdl.core.computation.base.**ClipParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres du rognage des données

class cdl.core.computation.base.**HistogramParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres de l'histogramme

get_suffix(*data : ndarray*) → str

Renvoie le suffixe pour le calcul de l'histogramme

Paramètres
data – tableau de données

class cdl.core.computation.base.**ROIDataParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Données de l'éditeur de ROI

classmethod create(*roidata : ndarray | None = None*, *singleobj : bool | None = None*)

Créer une instance de ROIDataParam

property is_empty: bool

Renvoie True s'il n'y a pas de ROI

class cdl.core.computation.base.**FFTParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres FFT

cdl.core.computation.base.**new_signal_result**(*src : SignalObj | ImageObj*, *name : str*, *suffix : str | None = None*, *units : tuple[str, str] | None = None*, *labels : tuple[str, str] | None = None*) → *SignalObj*

Créer un nouvel objet de signal en tant que résultat d'une fonction compute_11

Contrairement aux fonctions *dst_11*, cette fonction crée un nouvel objet de signal sans copier les métadonnées de l'objet d'origine, sauf pour l'entrée « source ».

Paramètres
— **src** – objet de signal ou d'image d'entrée
— **name** – nom de la fonction de traitement
— **suffix** – suffixe à ajouter au titre

— **units** – unités du signal de sortie
— **labels** – étiquettes du signal de sortie

Renvoie
Objet de signal de sortie

3.4.2 Fonctionnalités de traitement du signal

cdl.core.computation.signal.**dst_11**(*src :* SignalObj, *name : str*, *suffix : str | None = None*) → *SignalObj*

Créer un objet de signal résultant pour la fonction compute_11

Paramètres
— **src** (SignalObj) – signal source
— **name** (str) – nom de la fonction

Renvoie
objet de signal résultant

Type renvoyé
SignalObj

class cdl.core.computation.signal.**Wrap11Func**(*func : Callable*)

Encapsule une fonction 1 tableau -> 1 tableau (le cas simple de y1 = f(y0)) pour produire une fonction 1 SignalObj -> 1 SignalObj, qui peut être utilisée à l'intérieur de l'infrastructure de DataLab pour effectuer des calculs avec SignalProcessor.

Ce mécanisme d'encapsulation à l'aide d'une classe est nécessaire pour que la fonction résultante puisse être sélectionnée par le module multiprocessing.

L'instance de cet encapsuleur est appelable et renvoie un objet SignalObj.

Exemple

```
>>> import numpy as np
>>> from cdl.core.computation.signal import Wrap11Func
>>> import cdl.obj
>>> def square(y):
...     return y**2
>>> compute_square = Wrap11Func(square)
>>> x = np.linspace(0, 10, 100)
>>> y = np.sin(x)
>>> sig0 = cdl.obj.create_signal("Example", x, y)
>>> sig1 = compute_square(sig0)
```

Paramètres
func – Fonction 1 tableau -> 1 tableau

cdl.core.computation.signal.**dst_n1n**(*src1 :* SignalObj, *src2 :* SignalObj, *name : str*, *suffix : str | None = None*)

Créer un objet de signal résultant pour la fonction compute_n1n

Paramètres
— **src1** (SignalObj) – signal source 1
— **src2** (SignalObj) – signal source 2
— **name** (str) – nom de la fonction

Renvoie
objet de signal résultant

Type renvoyé
SignalObj

cdl.core.computation.signal.**compute_add**(*dst :* SignalObj, *src :* SignalObj) → *SignalObj*

Ajoute un signal au signal résultant

Paramètres
— **dst** (SignalObj) – signal de destination
— **src** (SignalObj) – signal source

cdl.core.computation.signal.**compute_product**(*dst :* SignalObj, *src :* SignalObj) → *SignalObj*

Multiplie un signal au signal résultant

Paramètres
— **dst** (SignalObj) – signal de destination
— **src** (SignalObj) – signal source

cdl.core.computation.signal.**compute_difference**(*src1 :* SignalObj, *src2 :* SignalObj) → *SignalObj*

Calculer la différence entre deux signaux

Paramètres
— **src1** (SignalObj) – signal source 1
— **src2** (SignalObj) – signal source 2

Renvoie
objet de signal résultant

Type renvoyé
SignalObj

cdl.core.computation.signal.**compute_quadratic_difference**(*src1 :* SignalObj, *src2 :* SignalObj) → *SignalObj*

Calculer la différence quadratique entre deux signaux

Paramètres
— **src1** (SignalObj) – signal source 1
— **src2** (SignalObj) – signal source 2

Renvoie
objet de signal résultant

Type renvoyé
SignalObj

cdl.core.computation.signal.**compute_division**(*src1 :* SignalObj, *src2 :* SignalObj) → *SignalObj*

Calculer la division entre deux signaux

Paramètres
— **src1** (SignalObj) – signal source 1
— **src2** (SignalObj) – signal source 2

Renvoie
objet de signal résultant

Type renvoyé
SignalObj

cdl.core.computation.signal.**extract_multiple_roi**(*src :* SignalObj, *group :* *DataSetGroup*) → *SignalObj*

Extrait plusieurs régions d'intérêt des données

Paramètres
— **src** (SignalObj) – signal source
— **group** (*gds.DataSetGroup*) – groupe de paramètres

Renvoie
signal avec plusieurs régions d'intérêt

Type renvoyé
SignalObj

cdl.core.computation.signal.**extract_single_roi**(*src :* SignalObj, *p : DataSet*) → *SignalObj*

Extrait une seule région d'intérêt des données

Paramètres
— **src** (SignalObj) – signal source
— **p** (*gds.DataSet*) – paramètres

Renvoie
signal avec une seule région d'intérêt

Type renvoyé
SignalObj

cdl.core.computation.signal.**compute_swap_axes**(*src :* SignalObj) → *SignalObj*

Permute les axes

Paramètres
src (SignalObj) – signal source

Renvoie
objet de signal résultant

Type renvoyé
SignalObj

cdl.core.computation.signal.**compute_abs**(*src :* SignalObj) → *SignalObj*

Calcule la valeur absolue

Paramètres
src (SignalObj) – signal source

Renvoie
objet de signal résultant

Type renvoyé
SignalObj

cdl.core.computation.signal.**compute_re**(*src :* SignalObj) → *SignalObj*

Calcule la partie réelle

Paramètres
src (SignalObj) – signal source

Renvoie
objet de signal résultant

Type renvoyé
SignalObj

cdl.core.computation.signal.**compute_im**(*src :* SignalObj) → *SignalObj*

Calcule la partie imaginaire

Paramètres
src (SignalObj) – signal source

Renvoie
objet de signal résultant

Type renvoyé
SignalObj

class cdl.core.computation.signal.**DataTypeSParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Convertir les paramètres du type de données du signal

cdl.core.computation.signal.**compute_astype**(*src :* SignalObj, *p :* DataTypeSParam) → *SignalObj*

Convertit le type de données

Paramètres
— **src** – signal source
— **p** – paramètres

Renvoie
Objet de signal résultant

cdl.core.computation.signal.**compute_log10**(*src :* SignalObj) → *SignalObj*

Calcule Log10

Paramètres
src (SignalObj) – signal source

Renvoie
objet de signal résultant

Type renvoyé
SignalObj

class cdl.core.computation.signal.**PeakDetectionParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres de détection de pics

cdl.core.computation.signal.**compute_peak_detection**(*src :* SignalObj, *p :* PeakDetectionParam) → *SignalObj*

Détection de pics

Paramètres
— **src** (SignalObj) – signal source
— **p** (PeakDetectionParam) – paramètres

Renvoie
objet de signal résultant

Type renvoyé
SignalObj

class cdl.core.computation.signal.**NormalizeYParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres de normalisation

cdl.core.computation.signal.**compute_normalize**(*src :* SignalObj, *p :* NormalizeYParam) → *SignalObj*

Normalise les données

Paramètres
— **src** (SignalObj) – signal source
— **p** (NormalizeYParam) – paramètres

Renvoie
objet de signal résultant

Type renvoyé
SignalObj

cdl.core.computation.signal.**compute_derivative**(*src :* SignalObj) → *SignalObj*

Calcule la dérivée

Paramètres
src (SignalObj) – signal source

Renvoie
objet de signal résultant

Type renvoyé
SignalObj

cdl.core.computation.signal.**compute_integral**(*src :* SignalObj) → *SignalObj*

Calcule l'intégrale

Paramètres
src (SignalObj) – signal source

Renvoie
objet de signal résultant

Type renvoyé
SignalObj

class cdl.core.computation.signal.**XYCalibrateParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres d'étalonnage du signal

cdl.core.computation.signal.**compute_calibration**(*src :* SignalObj, *p :* XYCalibrateParam) → *SignalObj*

Calcule l'étalonnage linéaire

Paramètres
— **src** (SignalObj) – signal source
— **p** (XYCalibrateParam) – paramètres

Renvoie
objet de signal résultant

Type renvoyé
SignalObj

cdl.core.computation.signal.**compute_threshold**(*src :* SignalObj, *p :* ThresholdParam) → *SignalObj*

Calcule le seuillage des données

Paramètres
— **src** (SignalObj) – signal source
— **p** (ThresholdParam) – paramètres

Renvoie
objet de signal résultant

Type renvoyé
SignalObj

cdl.core.computation.signal.**compute_clip**(*src :* SignalObj, *p :* ClipParam) → *SignalObj*

Calcule l'écrêtage des données

Paramètres
— **src** (SignalObj) – signal source
— **p** (ClipParam) – paramètres

Renvoie
objet de signal résultant

Type renvoyé
SignalObj

cdl.core.computation.signal.**compute_gaussian_filter**(*src :* SignalObj, *p :* GaussianParam) → *SignalObj*

Calcule le filtre gaussien

Paramètres
— **src** (SignalObj) – signal source
— **p** (GaussianParam) – paramètres

Renvoie
objet de signal résultant

Type renvoyé
SignalObj

cdl.core.computation.signal.**compute_moving_average**(*src :* SignalObj, *p :* MovingAverageParam) → *SignalObj*

Calcule la moyenne mobile

Paramètres
— **src** (SignalObj) – signal source
— **p** (MovingAverageParam) – paramètres

Renvoie
objet de signal résultant

Type renvoyé
SignalObj

cdl.core.computation.signal.**compute_moving_median**(*src :* SignalObj, *p :* MovingMedianParam) → *SignalObj*

Calcule la médiane mobile

Paramètres
— **src** (SignalObj) – signal source
— **p** (MovingMedianParam) – paramètres

Renvoie
objet de signal résultant

Type renvoyé
SignalObj

cdl.core.computation.signal.**compute_wiener**(*src :* SignalObj) → *SignalObj*

Calcule le filtre de Wiener

Paramètres
src (SignalObj) – signal source

Renvoie
objet de signal résultant

Type renvoyé
SignalObj

cdl.core.computation.signal.**compute_fft**(*src :* SignalObj, *p :* FFTParam) → *SignalObj*

Calcule la FFT

Paramètres
— **src** (SignalObj) – signal source
— **p** (FFTParam) – paramètres

Renvoie
objet de signal résultant

Type renvoyé
SignalObj

cdl.core.computation.signal.**compute_ifft**(*src :* SignalObj, *p :* FFTParam) → *SignalObj*

Calcule la iFFT

Paramètres

— **src** (SignalObj) – signal source
— **p** (FFTParam) – paramètres

Renvoie
objet de signal résultant

Type renvoyé
SignalObj

class cdl.core.computation.signal.**PolynomialFitParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres de l'ajustement polynomial

class cdl.core.computation.signal.**FWHMParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres LMH

cdl.core.computation.signal.**compute_fwhm**(*signal :* SignalObj, *param :* FWHMParam)

Calcule la LMH

cdl.core.computation.signal.**compute_fw1e2**(*signal :* SignalObj)

Calcule la largeur à $1/e^2$

cdl.core.computation.signal.**compute_histogram**(*src :* SignalObj, *p :* HistogramParam) → *SignalObj*

Calcule l'histogramme

Paramètres
— **src** (SignalObj) – signal source
— **p** (HistogramParam) – paramètres

Renvoie
objet de signal résultant

Type renvoyé
SignalObj

class cdl.core.computation.signal.**InterpolationParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres d'interpolation

cdl.core.computation.signal.**compute_interpolation**(*src1 :* SignalObj, *src2 :* SignalObj, *p :* InterpolationParam) → *SignalObj*

Interpole les données

Paramètres
— **src1** (SignalObj) – signal source 1
— **src2** (SignalObj) – signal source 2
— **p** (InterpolationParam) – paramètres

Renvoie
objet de signal résultant

Type renvoyé
SignalObj

class cdl.core.computation.signal.**ResamplingParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres de rééchantillonnage

cdl.core.computation.signal.**compute_resampling**(*src :* SignalObj, *p :* ResamplingParam) → *SignalObj*

Rééchantillonne les données

Paramètres

— **src** (SignalObj) – signal source
— **p** (*ResampleParam*) – paramètres

Renvoie
objet de signal résultant

Type renvoyé
SignalObj

class cdl.core.computation.signal.**DetrendingParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres d'élimination de tendance

cdl.core.computation.signal.**compute_detrending**(*src :* SignalObj, *p :* DetrendingParam) → *SignalObj*

Élimine la tendance des données

Paramètres
— **src** (SignalObj) – signal source
— **p** (DetrendingParam) – paramètres

Renvoie
objet de signal résultant

Type renvoyé
SignalObj

cdl.core.computation.signal.**compute_convolution**(*src1 :* SignalObj, *src2 :* SignalObj) → *SignalObj*

Calcule la convolution de deux signaux

Paramètres
— **src1** (SignalObj) – signal source 1
— **src2** (SignalObj) – signal source 2

Renvoie
objet de signal résultant

Type renvoyé
SignalObj

3.4.3 Fonctionnalités de traitement d'image

Fonctionnalités de traitement d'image de base

cdl.core.computation.image.**dst_11**(*src :* ImageObj, *name : str*, *suffix : str | None = None*) → *ImageObj*

Créer un objet d'image résultant pour la fonction compute_11

Paramètres
— **src** – objet d'image d'entrée
— **name** – nom de la fonction de traitement

Renvoie
Objet d'image de sortie

class cdl.core.computation.image.**Wrap11Func**(*func : Callable*)

Encapsule une fonction 1 tableau -> 1 tableau pour produire une fonction 1 `ImageObj` -> 1 `ImageObj`, qui peut être utilisée à l'intérieur de l'infrastructure de DataLab pour effectuer des calculs avec `ImageProcessor`.

Ce mécanisme d'encapsulation à l'aide d'une classe est nécessaire pour que la fonction résultante puisse être sélectionnée par le module `multiprocessing`.

L'instance de cet encapsuleur est appelable et renvoie un objet `ImageObj`.

Exemple

```
>>> import numpy as np
>>> from cdl.core.computation.signal import Wrap11Func
>>> import cdl.obj
>>> def add_noise(data):
...     return data + np.random.random(data.shape)
>>> compute_add_noise = Wrap11Func(add_noise)
>>> data= np.ones((100, 100))
>>> ima0 = cdl.obj.create_image("Example", data)
>>> ima1 = compute_add_noise(ima0)
```

Paramètres
func – Fonction 1 tableau -> 1 tableau

cdl.core.computation.image.**dst_11_signal**(*src :* ImageObj, *name : str*, *suffix : str* | *None* = *None*) → *SignalObj*

Créer un objet de signal résultant pour la fonction compute_11

Paramètres
— **src** – objet d'image d'entrée
— **name** – nom de la fonction de traitement

Renvoie
Objet de signal de sortie

cdl.core.computation.image.**dst_n1n**(*src1 :* ImageObj, *src2 :* ImageObj, *name : str*, *suffix : str* | *None* = *None*) → *ImageObj*

Créer un objet d'image résultant pour la fonction compute_n1n

Paramètres
— **src1** – objet d'image d'entrée
— **src2** – objet d'image d'entrée
— **name** – nom de la fonction de traitement

Renvoie
Objet d'image de sortie

cdl.core.computation.image.**compute_add**(*dst :* ImageObj, *src :* ImageObj) → *ImageObj*

Calculer l'addition entre deux images

Paramètres
— **dst** – objet d'image résultant
— **src** – objet d'image d'entrée

cdl.core.computation.image.**compute_product**(*dst :* ImageObj, *src :* ImageObj) → *ImageObj*

Calculer le produit entre deux images

Paramètres
— **dst** – objet d'image résultant
— **src** – objet d'image d'entrée

cdl.core.computation.image.**compute_difference**(*src1 :* ImageObj, *src2 :* ImageObj) → *ImageObj*

Calcule la différence entre deux images

Paramètres
— **src1** – objet d'image d'entrée
— **src2** – objet d'image d'entrée

Renvoie
Objet d'image de sortie

cdl.core.computation.image.**compute_quadratic_difference**(*src1 :* ImageObj, *src2 :* ImageObj) → *ImageObj*

Calcule la différence quadratique entre deux images

Paramètres
— **src1** – objet d'image d'entrée
— **src2** – objet d'image d'entrée

Renvoie
Objet d'image de sortie

cdl.core.computation.image.**compute_division**(*src1 :* ImageObj, *src2 :* ImageObj) → *ImageObj*

Calcule la division entre deux images

Paramètres
— **src1** – objet d'image d'entrée
— **src2** – objet d'image d'entrée

Renvoie
Objet d'image de sortie

class cdl.core.computation.image.**FlatFieldParam**(*title=None*, *comment=None*, *icon=''*)

Paramètres de correction de champ plat

cdl.core.computation.image.**compute_flatfield**(*src1 :* ImageObj, *src2 :* ImageObj, *p :* FlatFieldParam) → *ImageObj*

Calcule la correction de champ plat

Paramètres
— **src1** – objet image brute
— **src2** – objet image de champ plat
— **p** – paramètres de champ plat

Renvoie
Objet d'image de sortie

class cdl.core.computation.image.**LogP1Param**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres Log10

cdl.core.computation.image.**compute_logp1**(*src :* ImageObj, *p :* LogP1Param) → *ImageObj*

Calcule log10(z+n)

Paramètres
— **src** – objet d'image d'entrée
— **p** – paramètres

Renvoie
Objet d'image de sortie

class cdl.core.computation.image.**RotateParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres de rotation

cdl.core.computation.image.**rotate_obj_coords**(*angle : float*, *obj :* ImageObj, *orig :* ImageObj, *coords : ndarray*) → None

Appliquer la rotation aux coordonnées associées à l'objet d'image

Paramètres
— **angle** – angle de rotation (en degrés)
— **obj** – objet image
— **orig** – objet image original

— **coords** – coordonnées à faire tourner

Renvoie

Données de sortie

cdl.core.computation.image.**rotate_obj_alpha**(*obj :* ImageObj, *orig :* ImageObj, *coords :* *ndarray*, *p :* RotateParam) → None

Appliquer la rotation aux coordonnées associées à l'objet d'image

cdl.core.computation.image.**compute_rotate**(*src :* ImageObj, *p :* RotateParam) → *ImageObj*

Rotation des données

Paramètres

— **src** – objet d'image d'entrée

— **p** – paramètres

Renvoie

Objet d'image de sortie

cdl.core.computation.image.**rotate_obj_90**(*dst :* ImageObj, *src :* ImageObj, *coords :* *ndarray*) → None

Appliquer la rotation aux coordonnées associées à l'objet d'image

cdl.core.computation.image.**compute_rotate90**(*src :* ImageObj) → *ImageObj*

Rotation des données de 90°

Paramètres

src – objet d'image d'entrée

Renvoie

Objet d'image de sortie

cdl.core.computation.image.**rotate_obj_270**(*dst :* ImageObj, *src :* ImageObj, *coords :* *ndarray*) → None

Appliquer la rotation aux coordonnées associées à l'objet d'image

cdl.core.computation.image.**compute_rotate270**(*src :* ImageObj) → *ImageObj*

Rotation des données de 270°

Paramètres

src – objet d'image d'entrée

Renvoie

Objet d'image de sortie

cdl.core.computation.image.**hflip_coords**(*dst :* ImageObj, *src :* ImageObj, *coords :* *ndarray*) → None

Appliquer HFlip aux coordonnées

cdl.core.computation.image.**compute_fliph**(*src :* ImageObj) → *ImageObj*

Inversion horizontale des données

Paramètres

src – objet d'image d'entrée

Renvoie

Objet d'image de sortie

cdl.core.computation.image.**vflip_coords**(*dst :* ImageObj, *src :* ImageObj, *coords :* *ndarray*) → None

Appliquer VFlip aux coordonnées

cdl.core.computation.image.**compute_flipv**(*src :* ImageObj) → *ImageObj*

Inversion verticale des données

Paramètres

src – objet d'image d'entrée

Renvoie

Objet d'image de sortie

class cdl.core.computation.image.**GridParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres de grille

class cdl.core.computation.image.**ResizeParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres de redimensionnement

cdl.core.computation.image.**compute_resize**(*src :* ImageObj, *p :* ResizeParam) → *ImageObj*

Fonction de zoom

Paramètres

— **src** – objet d'image d'entrée

— **p** – paramètres

Renvoie

Objet d'image de sortie

class cdl.core.computation.image.**BinningParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres de binning

cdl.core.computation.image.**compute_binning**(*src :* ImageObj, *param :* BinningParam) → *ImageObj*

Fonction de binning sur les données

Paramètres

— **src** – objet d'image d'entrée

— **param** – paramètres

Renvoie

Objet d'image de sortie

cdl.core.computation.image.**extract_multiple_roi**(*src :* ImageObj, *group : DataSetGroup*) → *ImageObj*

Extrait plusieurs régions d'intérêt des données

Paramètres

— **src** – objet d'image d'entrée

— **group** – paramètres définissant les régions d'intérêt

Renvoie

Objet d'image de sortie

cdl.core.computation.image.**extract_single_roi**(*src :* ImageObj, *p : DataSet*) → *ImageObj*

Extrait une seule ROI

Paramètres

— **src** – objet d'image d'entrée

— **p** – Paramètres de ROI

Renvoie

Objet d'image de sortie

class cdl.core.computation.image.**ProfileParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres de profil horizontal ou vertical

cdl.core.computation.image.**compute_profile**(*src :* ImageObj, *p :* ProfileParam) → *ImageObj*

Calcule le profil horizontal ou vertical

Paramètres

— **src** – objet d'image d'entrée

— **p** – paramètres

Renvoie
Objet d'image de sortie

class cdl.core.computation.image.**AverageProfileParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres de profil horizontal ou vertical moyen

cdl.core.computation.image.**compute_average_profile**(*src :* ImageObj, *p :* AverageProfileParam) → *ImageObj*

Calcule le profil horizontal ou vertical moyen

Paramètres
— **src** – objet d'image d'entrée
— **p** – paramètres

Renvoie
Objet d'image de sortie

class cdl.core.computation.image.**RadialProfileParam**(**args*, ***kwargs*)

Paramètres de profil radial

update_from_image(*obj :* ImageObj) → None

Mettre à jour les paramètres à partir de l'image

choice_callback(*item*, *value*)

Callback pour l'item de choix

cdl.core.computation.image.**compute_radial_profile**(*src :* ImageObj, *p :* RadialProfileParam) → *SignalObj*

Calculer le profil radial autour du centre de gravité

Paramètres
— **src** – objet d'image d'entrée
— **p** – paramètres

Renvoie
Objet d'image de sortie

cdl.core.computation.image.**compute_histogram**(*src :* ImageObj, *p :* HistogramParam) → *SignalObj*

Calcule l'histogramme des données de l'image

Paramètres
— **src** – objet d'image d'entrée
— **p** – paramètres

Renvoie
Objet de signal de sortie

cdl.core.computation.image.**compute_swap_axes**(*src :* ImageObj) → *ImageObj*

Permute les axes de l'image

Paramètres
src – objet d'image d'entrée

Renvoie
Objet d'image de sortie

cdl.core.computation.image.**compute_abs**(*src :* ImageObj) → *ImageObj*

Calcule la valeur absolue

Paramètres
src – objet d'image d'entrée

Renvoie
Objet d'image de sortie

cdl.core.computation.image.**compute_re**(*src :* ImageObj) → *ImageObj*

Calcule la partie réelle

Paramètres
src – objet d'image d'entrée

Renvoie
Objet d'image de sortie

cdl.core.computation.image.**compute_im**(*src :* ImageObj) → *ImageObj*

Calcule la partie imaginaire

Paramètres
src – objet d'image d'entrée

Renvoie
Objet d'image de sortie

class cdl.core.computation.image.**DataTypeIParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Convertir les paramètres du type de données de l'image

cdl.core.computation.image.**compute_astype**(*src :* ImageObj, *p :* DataTypeIParam) → *ImageObj*

Convertit le type de données de l'image

Paramètres
— **src** – objet d'image d'entrée
— **p** – paramètres

Renvoie
Objet d'image de sortie

cdl.core.computation.image.**compute_log10**(*src :* ImageObj) → *ImageObj*

Calcule Log10

Paramètres
src – objet d'image d'entrée

Renvoie
Objet d'image de sortie

class cdl.core.computation.image.**ZCalibrateParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres d'étalonnage linéaire de l'image

cdl.core.computation.image.**compute_calibration**(*src :* ImageObj, *p :* ZCalibrateParam) → *ImageObj*

Calcule l'étalonnage linéaire

Paramètres
— **src** – objet d'image d'entrée
— **p** – paramètres d'étalonnage

Renvoie
Objet d'image de sortie

cdl.core.computation.image.**compute_threshold**(*src :* ImageObj, *p :* ThresholdParam) → *ImageObj*

Applique le seuillage

Paramètres
— **src** – objet d'image d'entrée
— **p** – paramètres

Renvoie
Objet d'image de sortie

cdl.core.computation.image.**compute_clip**(*src :* ImageObj, *p :* ClipParam) → *ImageObj*

Applique l'écrêtage

Paramètres
— **src** – objet d'image d'entrée
— **p** – paramètres

Renvoie
Objet d'image de sortie

cdl.core.computation.image.**compute_gaussian_filter**(*src :* ImageObj, *p :* GaussianParam) → *ImageObj*

Calcule le filtre gaussien

Paramètres
— **src** – objet d'image d'entrée
— **p** – paramètres

Renvoie
Objet d'image de sortie

cdl.core.computation.image.**compute_moving_average**(*src :* ImageObj, *p :* MovingAverageParam) → *ImageObj*

Calcule la moyenne mobile

Paramètres
— **src** – objet d'image d'entrée
— **p** – paramètres

Renvoie
Objet d'image de sortie

cdl.core.computation.image.**compute_moving_median**(*src :* ImageObj, *p :* MovingMedianParam) → *ImageObj*

Calcule la médiane mobile

Paramètres
— **src** – objet d'image d'entrée
— **p** – paramètres

Renvoie
Objet d'image de sortie

cdl.core.computation.image.**compute_wiener**(*src :* ImageObj) → *ImageObj*

Calcule le filtre de Wiener

Paramètres
src – objet d'image d'entrée

Renvoie
Objet d'image de sortie

cdl.core.computation.image.**compute_fft**(*src :* ImageObj, *p :* FFTParam) → *ImageObj*

Calcule la FFT

Paramètres
— **src** – objet d'image d'entrée
— **p** – paramètres

Renvoie
Objet d'image de sortie

cdl.core.computation.image.**compute_ifft**(*src :* ImageObj, *p :* FFTParam) → *ImageObj*

Calcule la FFT inverse

Paramètres

— **src** – objet d'image d'entrée
— **p** – paramètres

Renvoie
Objet d'image de sortie

class cdl.core.computation.image.**ButterworthParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres du filtre de Butterworth

cdl.core.computation.image.**compute_butterworth**(*src :* ImageObj, *p :* ButterworthParam) → *ImageObj*

Calcule le filtre de Butterworth

Paramètres
— **src** – objet d'image d'entrée
— **p** – paramètres

Renvoie
Objet d'image de sortie

cdl.core.computation.image.**calc_with_osr**(*image :* ImageObj, *func : Callable*, **args : Any*) → ndarray

Exécuter le calcul en tenant compte de l'image x0, y0, dx, dy et ROIs

Paramètres
— **image** – objet d'image d'entrée
— **func** – fonction de calcul
— ***args** – arguments de la fonction de calcul

Renvoie
Résultat du calcul

> **Avertissement :** La fonction de calcul doit prendre un seul argument (les données) ou de multiples arguments (les données suivies des paramètres de calcul).
>
> De plus, la fonction de calcul doit renvoyer une seule valeur ou un tableau NumPy contenant le résultat du calcul. Ce tableau contient les coordonnées des points, des polygones, des cercles ou des ellipses sous la forme [[x, y], …], ou [[x0, y0, x1, y1, …], …], ou [[x0, y0, r], …], ou [[x0, y0, a, b, theta], …].

cdl.core.computation.image.**get_centroid_coords**(*data : ndarray*) → ndarray

Retourne les coordonnées du centre de gravité

Paramètres
data – données d'entrée

Renvoie
Coordonnées du barycentre

cdl.core.computation.image.**compute_centroid**(*image :* ImageObj) → ndarray

Calcule le barycentre

Paramètres
image – image d'entrée

Renvoie
Coordonnées du barycentre

cdl.core.computation.image.**get_enclosing_circle_coords**(*data : ndarray*) → ndarray

Retourne les coordonnées du diamètre pour le contour de cercle contenant les valeurs d'image au-dessus du seuil (LMH)

Paramètres
data – données d'entrée

Renvoie
Coordonnées du diamètre

cdl.core.computation.image.**compute_enclosing_circle**(*image :* ImageObj) → ndarray

Calcule le cercle minimum englobant

Paramètres
image – image d'entrée

Renvoie
Coordonnées du diamètre

class cdl.core.computation.image.**HoughCircleParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres de transformation de Hough circulaire

cdl.core.computation.image.**compute_hough_circle_peaks**(*image :* ImageObj, *p :* HoughCircleParam) → ndarray

Calcule les cercles de Hough

Paramètres
— **image** – image d'entrée
— **p** – paramètres

Renvoie
Coordonnées du cercle

Fonctionnalités de correction d'exposition

Module de calcul d'exposition

class cdl.core.computation.image.exposure.**AdjustGammaParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres d'ajustement gamma

cdl.core.computation.image.exposure.**compute_adjust_gamma**(*src :* ImageObj, *p :* AdjustGammaParam) → *ImageObj*

Correction gamma

Paramètres
— **src** – objet d'image d'entrée
— **p** (AdjustGammaParam) – paramètres

Renvoie
Objet d'image de sortie

class cdl.core.computation.image.exposure.**AdjustLogParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres d'ajustement logarithmique

cdl.core.computation.image.exposure.**compute_adjust_log**(*src :* ImageObj, *p :* AdjustLogParam) → *ImageObj*

Calcule la correction logarithmique

Paramètres
— **src** – objet d'image d'entrée
— **p** – paramètres

Renvoie
Objet d'image de sortie

class cdl.core.computation.image.exposure.**AdjustSigmoidParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres d'ajustement sigmoïde

cdl.core.computation.image.exposure.**compute_adjust_sigmoid**(*src :* ImageObj, *p :* AdjustSigmoidParam) → *ImageObj*

Calcule la correction sigmoïde

Paramètres
— **src** – objet d'image d'entrée
— **p** – paramètres

Renvoie
Objet d'image de sortie

class cdl.core.computation.image.exposure.**RescaleIntensityParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres de rééchelonnement d'intensité

cdl.core.computation.image.exposure.**compute_rescale_intensity**(*src :* ImageObj, *p :* RescaleIntensityParam) → *ImageObj*

Rééchelonne les niveaux d'intensité de l'image

Paramètres
— **src** – objet d'image d'entrée
— **p** – paramètres

Renvoie
Objet d'image de sortie

class cdl.core.computation.image.exposure.**EqualizeHistParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres d'égalisation d'histogramme

cdl.core.computation.image.exposure.**compute_equalize_hist**(*src :* ImageObj, *p :* EqualizeHistParam) → *ImageObj*

Égalisation d'histogramme

Paramètres
— **src** – objet d'image d'entrée
— **p** – paramètres

Renvoie
Objet d'image de sortie

class cdl.core.computation.image.exposure.**EqualizeAdaptHistParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres d'égalisation d'histogramme adaptatif

cdl.core.computation.image.exposure.**compute_equalize_adapthist**(*src :* ImageObj, *p :* EqualizeAdaptHistParam) → *ImageObj*

Égalisation d'histogramme adaptative

Paramètres
— **src** – objet d'image d'entrée
— **p** – paramètres

Renvoie
Objet d'image de sortie

Fonctionnalités de restauration

Module de calcul de restauration

class cdl.core.computation.image.restoration.**DenoiseTVParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres de débruitage de variation totale

cdl.core.computation.image.restoration.**compute_denoise_tv**(*src :* ImageObj, *p :* DenoiseTVParam) → *ImageObj*

Calcule le débruitage de variation totale

Paramètres
— **src** – objet d'image d'entrée
— **p** – paramètres

Renvoie
Objet d'image de sortie

class cdl.core.computation.image.restoration.**DenoiseBilateralParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres de débruitage du filtre bilatéral

cdl.core.computation.image.restoration.**compute_denoise_bilateral**(*src :* ImageObj, *p :* DenoiseBilateralParam) → *ImageObj*

Calcule le débruitage du filtre bilatéral

Paramètres
— **src** – objet d'image d'entrée
— **p** – paramètres

Renvoie
Objet d'image de sortie

class cdl.core.computation.image.restoration.**DenoiseWaveletParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres de débruitage d'ondelettes

cdl.core.computation.image.restoration.**compute_denoise_wavelet**(*src :* ImageObj, *p :* DenoiseWaveletParam) → *ImageObj*

Calcule le débruitage d'ondelettes

Paramètres
— **src** – objet d'image d'entrée
— **p** – paramètres

Renvoie
Objet d'image de sortie

cdl.core.computation.image.restoration.**compute_denoise_tophat**(*src :* ImageObj, *p :* MorphologyParam) → *ImageObj*

Débruitage en utilisant White Top-Hat

Paramètres
— **src** – objet d'image d'entrée
— **p** – paramètres

Renvoie
Objet d'image de sortie

Fonctionnalités morphologiques

Module de calcul de morphologie

class cdl.core.computation.image.morphology.**MorphologyParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres de White Top-Hat

cdl.core.computation.image.morphology.**compute_white_tophat**(*src :* ImageObj, *p :* MorphologyParam) → *ImageObj*

Calcule le filtre White Top-Hat

Paramètres
— **src** – objet d'image d'entrée
— **p** – paramètres

Renvoie
Objet d'image de sortie

cdl.core.computation.image.morphology.**compute_black_tophat**(*src :* ImageObj, *p :* MorphologyParam) → *ImageObj*

Calcule le filtre Black Top-Hat

Paramètres
— **src** – objet d'image d'entrée
— **p** – paramètres

Renvoie
Objet d'image de sortie

cdl.core.computation.image.morphology.**compute_erosion**(*src :* ImageObj, *p :* MorphologyParam) → *ImageObj*

Calcule l'érosion

Paramètres
— **src** – objet d'image d'entrée

— **p** – paramètres

Renvoie
Objet d'image de sortie

cdl.core.computation.image.morphology.**compute_dilation**(*src :* ImageObj, *p :* MorphologyParam) → *ImageObj*

Calcule la dilatation

Paramètres
— **src** – objet d'image d'entrée
— **p** – paramètres

Renvoie
Objet d'image de sortie

cdl.core.computation.image.morphology.**compute_opening**(*src :* ImageObj, *p :* MorphologyParam) → *ImageObj*

Calcule l'ouverture morphologique

Paramètres
— **src** – objet d'image d'entrée
— **p** – paramètres

Renvoie
Objet d'image de sortie

cdl.core.computation.image.morphology.**compute_closing**(*src :* ImageObj, *p :* MorphologyParam) → *ImageObj*

Calcule la fermeture morphologique

Paramètres
— **src** – objet d'image d'entrée
— **p** – paramètres

Renvoie
Objet d'image de sortie

Fonctionnalités de détection de contours

Module de calcul de contours

class cdl.core.computation.image.edges.**CannyParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres du filtre de Canny

cdl.core.computation.image.edges.**compute_canny**(*src :* ImageObj, *p :* CannyParam) → *ImageObj*

Calcule le filtre de Canny

Paramètres
— **src** – objet d'image d'entrée
— **p** – paramètres

Renvoie
Objet d'image de sortie

cdl.core.computation.image.edges.**compute_roberts**(*src :* ImageObj) → *ImageObj*

Calcule le filtre de Roberts

Paramètres
src – objet d'image d'entrée

Renvoie
Objet d'image de sortie

cdl.core.computation.image.edges.**compute_prewitt**(*src :* ImageObj) → *ImageObj*

Calcule le filtre de Prewitt

Paramètres
src – objet d'image d'entrée

Renvoie
Objet d'image de sortie

cdl.core.computation.image.edges.**compute_prewitt_h**(*src :* ImageObj) → *ImageObj*

Calcule le filtre de Prewitt horizontal

Paramètres
src – objet d'image d'entrée

Renvoie
Objet d'image de sortie

cdl.core.computation.image.edges.**compute_prewitt_v**(*src :* ImageObj) → *ImageObj*

Calcule le filtre de Prewitt vertical

Paramètres
src – objet d'image d'entrée

Renvoie
Objet d'image de sortie

cdl.core.computation.image.edges.**compute_sobel**(*src :* ImageObj) → *ImageObj*

Calcule le filtre de Sobel

Paramètres
src – objet d'image d'entrée

Renvoie
Objet d'image de sortie

cdl.core.computation.image.edges.**compute_sobel_h**(*src :* ImageObj) → *ImageObj*

Calcule le filtre de Sobel horizontal

Paramètres
src – objet d'image d'entrée

Renvoie
Objet d'image de sortie

cdl.core.computation.image.edges.**compute_sobel_v**(*src :* ImageObj) → *ImageObj*

Calcule le filtre de Sobel vertical

Paramètres
src – objet d'image d'entrée

Renvoie
Objet d'image de sortie

cdl.core.computation.image.edges.**compute_scharr**(*src :* ImageObj) → *ImageObj*

Calcule le filtre de Scharr

Paramètres
src – objet d'image d'entrée

Renvoie
Objet d'image de sortie

cdl.core.computation.image.edges.**compute_scharr_h**(*src :* ImageObj) → *ImageObj*

Calcule le filtre de Scharr horizontal

Paramètres
src – objet d'image d'entrée

Renvoie
Objet d'image de sortie

cdl.core.computation.image.edges.**compute_scharr_v**(*src :* ImageObj) → *ImageObj*

Calcule le filtre de Scharr vertical

Paramètres
src – objet d'image d'entrée

Renvoie
Objet d'image de sortie

cdl.core.computation.image.edges.**compute_farid**(*src :* ImageObj) → *ImageObj*

Calcule le filtre de Farid

Paramètres
src – objet d'image d'entrée

Renvoie
Objet d'image de sortie

cdl.core.computation.image.edges.**compute_farid_h**(*src :* ImageObj) → *ImageObj*

Calcule le filtre de Farid horizontal

Paramètres
src – objet d'image d'entrée

Renvoie
Objet d'image de sortie

cdl.core.computation.image.edges.**compute_farid_v**(*src :* ImageObj) → *ImageObj*

Calcule le filtre de Farid vertical

Paramètres
src – objet d'image d'entrée

Renvoie
Objet d'image de sortie

cdl.core.computation.image.edges.**compute_laplace**(*src :* ImageObj) → *ImageObj*

Calcule le filtre de Laplace

Paramètres
src – objet d'image d'entrée

Renvoie
Objet d'image de sortie

Fonctionnalités de détection

Module de calcul de détection de taches

class cdl.core.computation.image.detection.**GenericDetectionParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres de détection génériques

class cdl.core.computation.image.detection.**Peak2DDetectionParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres de détection de pics

cdl.core.computation.image.detection.**compute_peak_detection**(*image :* ImageObj, *p :* Peak2DDetectionParam) → ndarray

Calcule la détection de pics 2D

Paramètres
— **imageOutput** – image d'entrée
— **p** – paramètres

Renvoie
Coordonnées des pics

class cdl.core.computation.image.detection.**ContourShapeParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Paramètres de forme de contour

cdl.core.computation.image.detection.**compute_contour_shape**(*image :* ImageObj, *p :* ContourShapeParam) → ndarray

Calculer l'ajustement de la forme du contour

class cdl.core.computation.image.detection.**BaseBlobParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Classe de base pour les paramètres de détection de taches

class cdl.core.computation.image.detection.**BlobDOGParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Détection de taches à l'aide de la méthode de différence de Gauss

cdl.core.computation.image.detection.**compute_blob_dog**(*image :* ImageObj, *p :* BlobDOGParam) → ndarray

Détection d taches à l'aide de la méthode de différence de Gauss

Paramètres
— **imageOutput** – image d'entrée
— **p** – paramètres

Renvoie
Coordonnées des taches

class cdl.core.computation.image.detection.**BlobDOHParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Détection de taches à l'aide de la méthode du déterminant de Hessian

cdl.core.computation.image.detection.**compute_blob_doh**(*image :* ImageObj, *p :* BlobDOHParam) → ndarray

Détection de taches à l'aide de la méthode du déterminant de Hessian

Paramètres
— **imageOutput** – image d'entrée
— **p** – paramètres

Renvoie
Coordonnées des taches

class cdl.core.computation.image.detection.**BlobLOGParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Détection de taches à l'aide de la méthode du Laplacien de Gauss

cdl.core.computation.image.detection.**compute_blob_log**(*image :* ImageObj, *p :* BlobLOGParam) → ndarray

Détection de taches à l'aide de la méthode du Laplacien de Gauss

Paramètres
— **imageOutput** – image d'entrée
— **p** – paramètres

Renvoie
Coordonnées des taches

class cdl.core.computation.image.detection.**BlobOpenCVParam**(*title : str | None = None*, *comment : str | None = None*, *icon : str = ''*, *readonly : bool = False*)

Détection de taches à l'aide d'OpenCV

cdl.core.computation.image.detection.**compute_blob_opencv**(*image :* ImageObj, *p :* BlobOpenCVParam) → ndarray

Détection de taches à l'aide d'OpenCV

Paramètres
— **imageOutput** – image d'entrée
— **p** – paramètres

Renvoie
Coordonnées des taches

3.5 Objets proxy (cdl.proxy)

Le module *cdl.proxy* fournit un moyen d'accéder aux fonctionnalités de DataLab à partir d'une classe proxy.

3.5.1 Proxy distant

Le proxy distant est utilisé lorsque DataLab est démarré à partir d'un processus différent du proxy. Dans ce cas, le proxy se connecte au serveur XML-RPC de DataLab.

class cdl.proxy.**RemoteProxy**(*autoconnect : bool = True*)

Classe proxy distant DataLab.

Cette classe fournit un accès aux fonctionnalités de DataLab à partir d'une classe proxy. Il s'agit de la version distante du proxy, qui est utilisée lorsque DataLab est démarré à partir d'un processus différent du proxy.

Paramètres
autoconnect (*bool*) – Connexion automatique au serveur XML-RPC de DataLab.

Lève
— **ConnectionRefusedError** – Impossible de se connecter à DataLab
— **ValueError** – Délai d'attente invalide (doit être >= 0.0)
— **ValueError** – Nombre de tentatives invalide (doit être >= 1)

Exemples

Voici un exemple simple de l'utilisation de RemoteProxy dans un script Python ou dans un notebook Jupyter :

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

add_annotations_from_items(*items : list*, *refresh_plot : bool = True*, *panel : str | None = None*) → None

Ajouter des annotations d'objet (éléments de tracé d'annotation).

Paramètres

- **items** (*list*) – éléments de tracé d'annotation
- **refresh_plot** (*bool | None*) – rafraîchir le tracé. Par défaut, True.
- **panel** (*str | None*) – nom du panneau (valeurs valides : « signal », « image »). Si None, le panneau actuel est utilisé.

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

Ajouter des données d'image à DataLab.

Paramètres

- **title** (*str*) – Titre de l'image
- **data** (*numpy.ndarray*) – Données de l'image
- **xunit** (*str | None*) – Unité X. Par défaut, None.
- **yunit** (*str | None*) – Unité Y. Par défaut, None.
- **zunit** (*str | None*) – Unité Z. Par défaut, None.
- **xlabel** (*str | None*) – Libellé X. Par défaut, None.
- **ylabel** (*str | None*) – Libellé Y. Par défaut, None.
- **zlabel** (*str | None*) – Libellé Z. Par défaut, None.

Renvoie

Vrai si l'image a été ajoutée avec succès, Faux sinon

Type renvoyé

bool

Lève

ValueError – Type de données invalide

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

Ajouter une étiquette avec le titre de l'objet sur le tracé associé

Paramètres

- **title** (*str | None*) – Titre de l'étiquette. Par défaut, None. Si None, le titre est le titre de l'objet.

— **panel** (*str | None*) – nom du panneau (valeurs valides : « signal », « image »). Si None, le panneau actuel est utilisé.

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

Ajouter un objet à DataLab.

Paramètres
obj (*SignalObj | ImageObj*) – Objet signal ou image

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

Ajouter des données de signal à DataLab.

Paramètres
— **title** (*str*) – Titre du signal
— **xdata** (*numpy.ndarray*) – Données X
— **ydata** (*numpy.ndarray*) – Données Y
— **xunit** (*str | None*) – Unité X. Par défaut, None.
— **yunit** (*str | None*) – Unité Y. Par défaut, None.
— **xlabel** (*str | None*) – Libellé X. Par défaut, None.
— **ylabel** (*str | None*) – Libellé Y. Par défaut, None.

Renvoie
Vrai si le signal a été ajouté avec succès, Faux sinon

Type renvoyé
bool

Lève
— **ValueError** – Type de données xdata invalide
— **ValueError** – Type de données ydata invalide

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

Appeler la fonction de calcul `name` dans le processeur du panneau actuel.

Paramètres
— **name** (*str*) – Nom de la fonction de calcul
— **param** (*guidata.dataset.DataSet | None*) – Paramètre de la fonction de calcul. Par défaut, None.

Renvoie
Résultat de la fonction de calcul

Type renvoyé
guidata.dataset.DataSet

close_application() → None

Fermer l'application DataLab

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

Tentative de connexion au serveur XML-RPC de DataLab.

Paramètres
— **port** (*str | None*) – Port XML-RPC auquel se connecter. S'il n'est pas spécifié, le port est récupéré automatiquement à partir de la configuration de DataLab.
— **timeout** (*float | None*) – Délai d'attente en secondes. Par défaut, 5.0.
— **retries** (*int | None*) – Nombre de tentatives. Par défaut, 10.

Lève
— **ConnectionRefusedError** – Impossible de se connecter à DataLab
— **ValueError** – Délai d'attente invalide (doit être >= 0.0)
— **ValueError** – Nombre de tentatives invalide (doit être >= 1)

context_no_refresh() → Generator[None, None, None]

Renvoie un gestionnaire de contexte pour désactiver temporairement le rafraîchissement automatique.

Renvoie
Gestionnaire de contexte

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*, *keep_roi : bool = False*) → None

Supprimer les métadonnées des objets sélectionnés

Paramètres
— **refresh_plot** – Actualiser le tracé. Par défaut, True.
— **keep_roi** – Conserver la ROI. Par défaut, False.

disconnect() → None

Déconnexion du serveur XML-RPC de DataLab.

get_current_panel() → str

Renvoie le nom du panneau actuel.

Renvoie
Nom du panneau (valeurs valides : « signal », « image », « macro »)

Type renvoyé
str

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

Renvoie les titres des groupes et les listes des UUID et des titres des objets internes.

Renvoie
titres des groupes, listes des UUID et des titres des objets internes

Type renvoyé
Tuple

get_method_list() → list[str]

Renvoie la liste des méthodes disponibles.

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

Obtenir l'objet (signal/image) à partir de l'index.

Paramètres
— **nb_id_title** – Numéro de l'objet, ou ID de l'objet, ou titre de l'objet. Par défaut, None (objet actuel).
— **panel** – Nom du panneau. Par défaut, None (panneau actuel).

Renvoie
Objet

Lève
KeyError – si l'objet n'est pas trouvé

get_object_shapes(*nb_id_title : int | str | None = None*, *panel : str | None = None*) → list

Obtenir les formes des éléments de tracé associées à l'objet (signal/image).

Paramètres

— **nb_id_title** – Numéro de l'objet, ou ID de l'objet, ou titre de l'objet. Par défaut, None (objet actuel).

— **panel** – Nom du panneau. Par défaut, None (panneau actuel).

Renvoie

Liste des formes des éléments de tracé

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

Obtenir la liste des objets (signal/image) pour le panneau actuel. Les objets sont triés par numéro de groupe et index d'objet dans le groupe.

Paramètres

panel – nom du panneau (valeurs valides : « signal », « image », « macro »). Si None, le panneau de données actuel est utilisé (c'est-à-dire le panneau signal ou image).

Renvoie

Liste des titres des objets

Lève

ValueError – si le panneau n'est pas trouvé

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

Obtenir la liste des UUID des objets (signal/image) pour le panneau actuel. Les objets sont triés par numéro de groupe et index d'objet dans le groupe.

Paramètres

panel (*str | None*) – nom du panneau (valeurs valides : « signal », « image »). Si None, le panneau actuel est utilisé.

Renvoie

liste des UUID des objets

Type renvoyé

list[str]

Lève

ValueError – si le panneau n'est pas trouvé

classmethod get_public_methods() → list[str]

Renvoie toutes les méthodes publiques de la classe, à l'exception de celle-ci.

Renvoie

Liste des méthodes publiques

Type renvoyé

list[str]

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

Renvoie les UUID des objets sélectionnés.

Paramètres

include_groups – Si True, renvoie également les objets des groupes sélectionnés.

Renvoie

Liste des UUID des objets sélectionnés.

get_version() → str

Renvoie la version de DataLab.

Renvoie

Version de DataLab

Type renvoyé

str

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

Ouvrir le navigateur DataLab HDF5 pour importer un fichier HDF5.

Paramètres

— **filename** (*str*) – Nom du fichier HDF5

— **reset_all** (*bool | None*) – Réinitialiser toutes les données de l'application. Par défaut, None.

import_macro_from_file(*filename : str*) → None

Importer une macro à partir d'un fichier

Paramètres

filename – Nom de fichier.

is_connected() → bool

Renvoie True si connecté au serveur XML-RPC de DataLab.

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

Ouvrir un fichier HDF5 DataLab ou importer à partir de tout autre fichier HDF5.

Paramètres

— **h5files** (*list[str] | None*) – Liste des fichiers HDF5 à ouvrir. Par défaut, None.

— **import_all** (*bool | None*) – Importer tous les objets à partir des fichiers HDF5. Par défaut, None.

— **reset_all** (*bool | None*) – Réinitialiser toutes les données de l'application. Par défaut, None.

open_object(*filename : str*) → None

Ouvrir un objet à partir d'un fichier dans le panneau actuel (signal/image).

Paramètres

filename (*str*) – Nom du fichier

raise_window() → None

Afficher la fenêtre DataLab

reset_all() → None

Réinitialiser toutes les données de l'application

run_macro(*number_or_title : int | str | None = None*) → None

Exécute la macro.

Paramètres

number_or_title – Numéro de macro, ou titre de macro. Par défaut, None (macro actuelle).

Lève

ValueError – si la macro n'est pas trouvée

save_to_h5_file(*filename : str*) → None

Enregistrer dans un fichier HDF5 DataLab.

Paramètres

filename (*str*) – Nom du fichier HDF5

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

Sélectionner des groupes dans le panneau actuel.

Paramètres

— **selection** – Liste de numéros de groupe (1 à N), ou liste d'UUID de groupe, ou None pour sélectionner tous les groupes. Par défaut, None.

— **panel** (*str | None*) – nom du panneau (valeurs valides : « signal », « image »). Si None, le panneau actuel est utilisé. Par défaut, None.

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

Sélectionner des objets dans le panneau actuel.

Paramètres

— **selection** – Liste de numéros d'objet (1 à N) ou d'UUID à sélectionner
— **panel** – nom du panneau (valeurs valides : « signal », « image »). Si None, le panneau actuel est utilisé. Par défaut, None.

set_current_panel(*panel : str*) → None

Passer au panneau.

Paramètres

panel (*str*) – Nom du panneau (valeurs valides : « signal », « image », « macro »)

stop_macro(*number_or_title : int | str | None = None*) → None

Arrête la macro.

Paramètres

number_or_title – Numéro de macro, ou titre de macro. Par défaut, None (macro actuelle).

Lève

ValueError – si la macro n'est pas trouvée

toggle_auto_refresh(*state : bool*) → None

Basculer l'état de rafraîchissement automatique.

Paramètres

state (*bool*) – État de rafraîchissement automatique

toggle_show_titles(*state : bool*) → None

Basculer l'état d'affichage des titres.

Paramètres

state (*bool*) – État d'affichage des titres

3.5.2 Proxy local

Le proxy local est utilisé lorsque DataLab est démarré à partir du même processus que le proxy. Dans ce cas, le proxy est directement connecté à l'instance de la fenêtre principale de DataLab. Le cas d'utilisation typique est le script de haut niveau.

class cdl.proxy.**LocalProxy**(*cdl : CDLMainWindow | ServerProxy | None = None*)

Classe de proxy local DataLab.

Cette classe permet d'accéder aux fonctionnalités de DataLab à partir d'une classe proxy. Il s'agit de la version locale du proxy, utilisée lorsque DataLab est démarré à partir du même processus que le proxy.

Paramètres

cdl (*CDLMainWindow*) – Instance de CDLMainWindow.

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

Ajouter des données de signal à DataLab.

Paramètres

— **title** (*str*) – Titre du signal
— **xdata** (*numpy.ndarray*) – Données X
— **ydata** (*numpy.ndarray*) – Données Y
— **xunit** (*str | None*) – Unité X. Par défaut, None.
— **yunit** (*str | None*) – Unité Y. Par défaut, None.
— **xlabel** (*str | None*) – Libellé X. Par défaut, None.

— **ylabel** (*str | None*) – Libellé Y. Par défaut, None.

Renvoie
Vrai si le signal a été ajouté avec succès, Faux sinon

Type renvoyé
bool

Lève
— **ValueError** – Type de données xdata invalide
— **ValueError** – Type de données ydata invalide

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

Ajouter des données d'image à DataLab.

Paramètres
— **title** (*str*) – Titre de l'image
— **data** (*numpy.ndarray*) – Données de l'image
— **xunit** (*str | None*) – Unité X. Par défaut, None.
— **yunit** (*str | None*) – Unité Y. Par défaut, None.
— **zunit** (*str | None*) – Unité Z. Par défaut, None.
— **xlabel** (*str | None*) – Libellé X. Par défaut, None.
— **ylabel** (*str | None*) – Libellé Y. Par défaut, None.
— **zlabel** (*str | None*) – Libellé Z. Par défaut, None.

Renvoie
Vrai si l'image a été ajoutée avec succès, Faux sinon

Type renvoyé
bool

Lève
ValueError – Type de données invalide

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

Appeler la fonction de calcul **name** dans le processeur du panneau actuel.

Paramètres
— **name** (*str*) – Nom de la fonction de calcul
— **param** (*guidata.dataset.DataSet | None*) – Fonction de calcul
— **None.** (*parameter. Defaults to*) –

Renvoie
Résultat de la fonction de calcul

Type renvoyé
guidata.dataset.DataSet

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

Obtenir l'objet (signal/image) à partir de l'index.

Paramètres
— **nb_id_title** – Numéro de l'objet, ou ID de l'objet, ou titre de l'objet. Par défaut, None (objet actuel).
— **panel** – Nom du panneau. Par défaut, None (panneau actuel).

Renvoie
Objet

Lève
KeyError – si l'objet n'est pas trouvé

get_object_shapes(*nb_id_title : int | str | None = None*, *panel : str | None = None*) → list

Obtenir les formes des éléments de tracé associées à l'objet (signal/image).

Paramètres

— **nb_id_title** – Numéro de l'objet, ou ID de l'objet, ou titre de l'objet. Par défaut, None (objet actuel).
— **panel** – Nom du panneau. Par défaut, None (panneau actuel).

Renvoie
Liste des formes des éléments de tracé

add_annotations_from_items(*items : list*, *refresh_plot : bool = True*, *panel : str | None = None*) → None

Ajouter des annotations d'objet (éléments de tracé d'annotation).

Paramètres
— **items** (*list*) – éléments de tracé d'annotation
— **refresh_plot** (*bool | None*) – rafraîchir le tracé. Par défaut, True.
— **panel** (*str | None*) – nom du panneau (valeurs valides : « signal », « image »). Si None, le panneau actuel est utilisé.

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

Ajouter un objet à DataLab.

Paramètres
obj (SignalObj | ImageObj) – Objet signal ou image

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

Ajouter une étiquette avec le titre de l'objet sur le tracé associé

Paramètres
— **title** (*str | None*) – Titre de l'étiquette. Par défaut, None. Si None, le titre est le titre de l'objet.
— **panel** (*str | None*) – nom du panneau (valeurs valides : « signal », « image »). Si None, le panneau actuel est utilisé.

close_application() → None

Fermer l'application DataLab

context_no_refresh() → Generator[None, None, None]

Renvoie un gestionnaire de contexte pour désactiver temporairement le rafraîchissement automatique.

Renvoie
Gestionnaire de contexte

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*, *keep_roi : bool = False*) → None

Supprimer les métadonnées des objets sélectionnés

Paramètres
— **refresh_plot** – Actualiser le tracé. Par défaut, True.
— **keep_roi** – Conserver la ROI. Par défaut, False.

get_current_panel() → str

Renvoie le nom du panneau actuel.

Renvoie
Nom du panneau (valeurs valides : « signal », « image », « macro »)

Type renvoyé
str

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

Renvoie les titres des groupes et les listes des UUID et des titres des objets internes.

Renvoie
titres des groupes, listes des UUID et des titres des objets internes

Type renvoyé
Tuple

`get_object_titles`(*panel : str | None = None*) → list[str]

Obtenir la liste des objets (signal/image) pour le panneau actuel. Les objets sont triés par numéro de groupe et index d'objet dans le groupe.

Paramètres
panel – nom du panneau (valeurs valides : « signal », « image », « macro »). Si None, le panneau de données actuel est utilisé (c'est-à-dire le panneau signal ou image).

Renvoie
Liste des titres des objets

Lève
ValueError – si le panneau n'est pas trouvé

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

Obtenir la liste des UUID des objets (signal/image) pour le panneau actuel. Les objets sont triés par numéro de groupe et index d'objet dans le groupe.

Paramètres
panel (*str | None*) – nom du panneau (valeurs valides : « signal », « image »). Si None, le panneau actuel est utilisé.

Renvoie
liste des UUID des objets

Type renvoyé
list[str]

Lève
ValueError – si le panneau n'est pas trouvé

`classmethod get_public_methods`() → list[str]

Renvoie toutes les méthodes publiques de la classe, à l'exception de celle-ci.

Renvoie
Liste des méthodes publiques

Type renvoyé
list[str]

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

Renvoie les UUID des objets sélectionnés.

Paramètres
include_groups – Si True, renvoie également les objets des groupes sélectionnés.

Renvoie
Liste des UUID des objets sélectionnés.

`get_version`() → str

Renvoie la version de DataLab.

Renvoie
Version de DataLab

Type renvoyé
str

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

Ouvrir le navigateur DataLab HDF5 pour importer un fichier HDF5.

Paramètres
- **filename** (*str*) – Nom du fichier HDF5
- **reset_all** (*bool | None*) – Réinitialiser toutes les données de l'application. Par défaut, None.

import_macro_from_file(*filename : str*) → None

Importer une macro à partir d'un fichier

Paramètres
filename – Nom de fichier.

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

Ouvrir un fichier HDF5 DataLab ou importer à partir de tout autre fichier HDF5.

Paramètres
- **h5files** (*list[str] | None*) – Liste des fichiers HDF5 à ouvrir. Par défaut, None.
- **import_all** (*bool | None*) – Importer tous les objets à partir des fichiers HDF5. Par défaut, None.
- **reset_all** (*bool | None*) – Réinitialiser toutes les données de l'application. Par défaut, None.

open_object(*filename : str*) → None

Ouvrir un objet à partir d'un fichier dans le panneau actuel (signal/image).

Paramètres
filename (*str*) – Nom du fichier

raise_window() → None

Afficher la fenêtre DataLab

reset_all() → None

Réinitialiser toutes les données de l'application

run_macro(*number_or_title : int | str | None = None*) → None

Exécute la macro.

Paramètres
number_or_title – Numéro de macro, ou titre de macro. Par défaut, None (macro actuelle).

Lève
ValueError – si la macro n'est pas trouvée

save_to_h5_file(*filename : str*) → None

Enregistrer dans un fichier HDF5 DataLab.

Paramètres
filename (*str*) – Nom du fichier HDF5

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

Sélectionner des groupes dans le panneau actuel.

Paramètres
- **selection** – Liste de numéros de groupe (1 à N), ou liste d'UUID de groupe, ou None pour sélectionner tous les groupes. Par défaut, None.

— **panel** (*str | None*) – nom du panneau (valeurs valides : « signal », « image »). Si None, le panneau actuel est utilisé. Par défaut, None.

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

Sélectionner des objets dans le panneau actuel.

Paramètres

— **selection** – Liste de numéros d'objet (1 à N) ou d'UUID à sélectionner

— **panel** – nom du panneau (valeurs valides : « signal », « image »). Si None, le panneau actuel est utilisé. Par défaut, None.

set_current_panel(*panel : str*) → None

Passer au panneau.

Paramètres

panel (*str*) – Nom du panneau (valeurs valides : « signal », « image », « macro »)

stop_macro(*number_or_title : int | str | None = None*) → None

Arrête la macro.

Paramètres

number_or_title – Numéro de macro, ou titre de macro. Par défaut, None (macro actuelle).

Lève

ValueError – si la macro n'est pas trouvée

toggle_auto_refresh(*state : bool*) → None

Basculer l'état de rafraîchissement automatique.

Paramètres

state (*bool*) – État de rafraîchissement automatique

toggle_show_titles(*state : bool*) → None

Basculer l'état d'affichage des titres.

Paramètres

state (*bool*) – État d'affichage des titres

3.5.3 Gestionnaire de contexte proxy

Le gestionnaire de contexte proxy est un moyen pratique de gérer la création et la destruction du proxy. Il est utilisé comme suit :

```
with proxy_context("local") as proxy:
    proxy.add_signal(...)
```

Le type de proxy peut être « local » ou « remote ». Pour un proxy distant, le port peut être spécifié comme « remote :port ».

Note : Le gestionnaire de contexte proxy permet d'utiliser le proxy dans différents contextes (script Python, notebook Jupyter, etc.). Il permet également de passer sans heurts du proxy local au proxy distant, en conservant le même code à l'intérieur du contexte.

---

cdl.proxy.**proxy_context**(*what : str*) → Generator[*LocalProxy* | *RemoteProxy*, None, None]

Gestionnaire de contexte gérant la création et la destruction du proxy CDL.

Paramètres

what (*str*) – type de proxy (« local » ou « remote »). Pour un proxy distant, le port peut être spécifié comme « remote :port ».

Renvoie

Générateur[LocalProxy | RemoteProxy, None, None] –

proxy

LocalProxy si what == « local », RemoteProxy si what == « remote » ou « remote :port »

Exemple

avec proxy_context(« local ») en tant que proxy :
proxy.add_signal(…)

CHAPITRE 4

Contribuer

DataLab est **votre** plateforme. Si vous souhaitez qu'elle s'améliore, vous **pouvez** contribuer au projet, que vous soyez développeur ou non.

Il existe de nombreuses façons de contribuer au projet DataLab, en fonction du temps dont vous disposez, de votre expérience avec les projets open source et de vos compétences.

4.1 Partagez vos idées et vos expériences

Outre les rapports d'anomalies classiques et les demandes de fonctionnalités, vous pouvez partager vos idées et vos expériences pour améliorer DataLab. En particulier, nous sommes très intéressés par vos commentaires sur la documentation et les tutoriels. De plus, si vous avez un cas d'utilisation que vous souhaitez partager avec la communauté, faites-le nous savoir.

Sans coder une seule ligne, vous pouvez contribuer au projet DataLab en :

- Signaler une anomalie
- Suggérer une amélioration
- Suggérer un sujet de documentation
- Suggérer un sujet de tutoriel

4.2 Partagez vos connaissances scientifiques/techniques

Vos connaissances techniques ou scientifiques sont également très précieuses pour nous, car vous pouvez contribuer directement à la documentation ou aux tutoriels. Ou, mieux encore, si vous souhaitez rédiger un tutoriel, nous serons heureux de vous aider.

Encore une fois sans coder, vous pouvez contribuer au projet DataLab en :

- Ecrire de la documentation
- Ecrire un tutoriel

4.3 Contribuer aux nouvelles fonctionnalités

Même si vous n'êtes pas développeur, vous pouvez contribuer au projet en :
- Testant de nouvelles fonctionnalités
- Ecrivant et soumettant de nouveaux plugins
- Ecivant et soumettant des macro-commandes

4.4 Développer de nouvelles fonctionnalités

Si vous êtes développeur, vous pouvez contribuer au cœur du projet en corrigeant des anomalies ou en implémentant de nouvelles fonctionnalités.

Voir la section *Contribuer au code* pour plus d'informations.

4.4.1 Contribuer au code

Cette page explique comment vous pouvez contribuer au code du projet.

Voir aussi :

Les règles de codage sont présentées dans la page *Règles de codage*.

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 GitHub du projet.

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.

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.

4.4.2 Règles de codage

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.

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.

4.4.3 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).

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

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.

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).

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\\texmfs\\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:\C2OIQ-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).

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

4.4.4 Feuille de route

Jalons futurs

Fonctionnalités

- Ajouter une interface de noyau Jupyter à DataLab :
  - Cela permettrait d'utiliser DataLab à partir d'autres logiciels, tels que des notebooks Jupyter, Spyder ou Visual Studio Code
  - Cela permettrait également de partager des données entre DataLab et d'autres logiciels (par exemple, afficher les résultats numériques de DataLab dans des notebooks Jupyter ou vice versa, afficher les résultats de Jupyter dans DataLab, etc.)
- Créer un plugin Jupyter pour l'analyse de données interactive avec DataLab :
  - Il est déjà possible d'utiliser DataLab à partir d'un notebook Jupyter, grâce aux fonctionnalités de contrôle à distance (voir *Contrôle à distance*), mais il serait utile d'avoir un plugin dédié
  - Ce plugin permettrait d'utiliser DataLab comme noyau Jupyter, et d'afficher les résultats numériques de DataLab dans des notebooks Jupyter ou vice versa (par exemple, afficher les résultats de Jupyter dans DataLab)
  - Ce plugin permettrait également d'utiliser les fonctionnalités de traitement de DataLab à partir de notebooks Jupyter
  - Un cas d'utilisation typique pourrait également consister à utiliser DataLab pour manipuler des signaux ou des images de manière efficace, et à utiliser Jupyter pour l'analyse de données personnalisée basée sur des algorithmes spécifiques / faits maison
  - Ce plugin pourrait être implémenté en utilisant l'interface du noyau Jupyter (voir ci-dessus)
- Créer un plugin Spyder pour l'analyse de données interactive connectée à DataLab :
  - Il s'agit exactement du même cas d'utilisation que pour le plugin Jupyter, mais pour Spyder
  - Ce plugin pourrait également être implémenté en utilisant l'interface du noyau Jupyter (voir ci-dessus)

— Ajouter la prise en charge des séries temporelles (voir Issue #27)

Maintenance

— 2024 : passer à gRPC pour le contrôle à distance (au lieu de XML-RPC), s'il y a besoin d'un protocole de communication plus efficace (voir Issue #18)
— 2025 : abandonner le support de PyQt5 (fin de vie : mi-2025), et passer à PyQt6 ; cela devrait être simple, grâce à la couche de compatibilité *qtpy* et au fait que *PlotPyStack* est déjà compatible avec PyQt6)

Autres tâches

— Créer un modèle de plugin DataLab (voir Issue #26)
— Réaliser des vidéos de démonstration : système de plugins, fonctionnalités de contrôle à distance, etc. (voir Issue #25)

Jalons passés

DataLab 0.11

— Ajouter une fonctionnalité de glisser-déposer aux panneaux de signaux et d'images, pour permettre de réorganiser les signaux et les images (voir Issue #17)
— Ajouter des boutons « Monter » et « Descendre » aux panneaux de signaux et d'images, pour permettre de réorganiser les signaux et les images (voir Issue #22)
— Ajouter des fonctionnalités de convolution 1D, d'interpolation, de rééchantillonnage et d'élimination des tendances

DataLab 0.10

— Développer un plugin DataLab très simple pour démontrer le système de plugins
— Sérialiser les styles de courbes et d'images dans les fichiers HDF5
— Ajouter une option globale « Rafraîchissement automatique », pour pouvoir désactiver le rafraîchissement automatique de la fenêtre principale lors de l'exécution de plusieurs étapes de traitement, améliorant ainsi les performances
— Améliorer la lisibilité des courbes (par exemple, éviter les lignes en pointillés, utiliser des couleurs contrastées et utiliser l'anti-crénelage)

DataLab 0.9

— Python 3.11 est la nouvelle référence
— Exécuter les calculs dans un processus séparé :
  — Exécuter un « serveur de calcul » en arrière-plan, dans un autre processus
  — Pour chaque calcul, envoyer les données sérialisées et la fonction de calcul au serveur et attendre le résultat
  — Il est alors possible d'arrêter n'importe quel calcul à tout moment en tuant le processus du serveur et en le redémarrant (éventuellement après avoir incrémenté le numéro de port de communication)
— Optimiser les performances d'affichage des images
— Ajouter une boîte de dialogue de préférences
— Ajouter de nouvelles fonctionnalités de traitement d'images : débruitage, …
— Résultats du traitement d'images : ajout de la prise en charge des formes polygonales (par exemple, pour la détection des contours)

- Nouveau système de plugins : API pour les extensions tierces
  - Objectif n°1 : un plugin doit être gérable à l'aide d'un seul script Python, qui inclut une extension de *ImageProcessor*, *ActionHandler* et une nouvelle prise en charge du format de fichier
  - Objectif n°2 : les plugins doivent simplement être stockés dans un dossier qui est par défaut le répertoire de l'utilisateur (même dossier que le fichier de configuration « .DataLab.ini »)
- Ajouter un système de macro-commandes :
  - Nouvel éditeur Python intégré
  - Scripts utilisant la même API que les scénarios de test applicatifs de haut niveau
  - Prise en charge de l'enregistrement de macro
- Ajouter un serveur xmlrpc pour permettre le contrôle à distance de DataLab :
  - Contrôle des principales fonctionnalités de DataLab (ouvrir un signal ou une image, ouvrir un fichier HDF5, etc.) et des fonctionnalités de traitement (exécuter un calcul, etc.)
  - Prendre le contrôle de DataLab à partir d'un logiciel tiers
  - Exécuter des calculs interactifs à partir d'un IDE (par exemple Spyder ou Visual Studio Code)

4.4.5 Historique des modifications

Voir la page de la [feuille de route](https://datalab-platform.com/fr/contributing/roadmap.html) de DataLab pour les jalons futurs et passés.

DataLab Version 0.14.1

Nouveau nom de domaine : datalab-platform.com

Nouvelles fonctionnalités :

- Ajout de la prise en charge de l'inversion de la palette de couleurs dans la Vue Image :
  - Ajout de la nouvelle entrée « Inverser la palette de couleurs » dans le menu contextuel du graphique, les paramètres de l'image et dans les paramètres par défaut de la vue image
  - Cette fonctionnalité nécessite *PlotPy* v2.3 ou ultérieur
- Explorateur HDF5 :
  - Ajout du bouton « Afficher le tableau » dans le coin des onglets « Groupe » et « Attributs », pour afficher le tableau dans une fenêtre séparée (utile pour copier/coller des données dans d'autres applications, par exemple)
  - Attributs : ajout de la prise en charge de plus de types de données scalaires
- Testabilité et maintenabilité :
  - Les tests unitaires de DataLab utilisent désormais [pytest](https://pytest.org). Cela a nécessité beaucoup de travail pour la transition, en particulier pour réadapter les tests afin qu'ils puissent être exécutés dans le même processus. Par exemple, une attention particulière a été portée à l'isolement des tests, afin qu'ils n'interfèrent pas les uns avec les autres.
  - Ajout de l'intégration continue (CI) avec GitHub Actions
  - Pour cette version, la couverture des tests est de 87%
- Assistant d'importation de fichiers texte :
  - Amélioration drastique des performances de l'aperçu du tableau lors de l'importation de fichiers texte volumineux (plus de barre de progression, et l'aperçu est désormais affiché presque instantanément)

Corrections de bugs :

- Le serveur XML-RPC n'était pas arrêté correctement lors de la fermeture de DataLab
- Correction de problèmes liés aux tests : certains cas limites étaient masqués par l'ancienne suite de tests, et ont été révélés par la transition vers *pytest*. Cela a conduit à des corrections de bugs et à des améliorations du code.
- Sur Linux, lors de l'exécution d'un calcul sur un signal ou une image, et à de rares occasions, le calcul restait bloqué comme s'il s'exécutait indéfiniment. Même si l'interface graphique était toujours réactive, le calcul n'avançait pas et l'utilisateur devait annuler l'opération et la redémarrer. Cela était dû à la méthode de démarrage du processus séparé utilisé pour le calcul (la méthode par défaut était « fork » sur Linux). Ceci est maintenant

corrigé en utilisant la méthode « spawn » à la place, qui est la méthode recommandée pour les dernières versions de Python sur Linux lorsque le multithreading est employé.
- Correction de l'"[Issue #60](https://github.com/DataLab-Platform/DataLab/issues/60) - *OSError : Invalid HDF5 file […]* lors de la tentative d'ouverture d'un fichier HDF5 avec une extension autre que « .h5 »
- Région d'intérêt (ROI) d'image : lors de la modification des limites de l'image dans la boîte de dialogue de confirmation, la ROI n'était pas mise à jour en conséquence jusqu'à ce que l'opération soit relancée
- Problèmes d'obsolescence :
  - Correction de l'avertissement d'obsolescence *scipy.ndimage.filters*
  - Correction de l'avertissement d'obsolescence *numpy.fromstring*

DataLab Version 0.14.0

Nouvelles fonctionnalités :
- Nouvelle fonctionnalité « Histogramme » dans le menu « Calculs » :
  - Ajout de la fonctionnalité de calcul d'histogramme pour les signaux et les images
  - L'histogramme est calculé sur les régions d'intérêt (ROI) le cas échéant, ou sur l'ensemble du signal/image si aucune ROI n'est définie
  - Paramètres modifiables : nombre de classes, limites inférieure et supérieure
- Explorateur HDF5 :
  - Amélioration de la disposition de la vue arborescente (plus compacte et lisible)
  - Plusieurs fichiers peuvent désormais être ouverts en même temps, en utilisant la boîte de dialogue de sélection de fichiers
  - Ajout d'onglets avec des informations sous l'aperçu graphique :
    - Informations sur le groupe : chemin, aperçu textuel, etc.
    - Informations sur les attributs : nom, valeur
  - Ajout de la case à cocher « Afficher uniquement les données prises en charge » : lorsqu'elle est cochée, seules les données prises en charge (signaux et images) sont affichées dans la vue arborescente
  - Ajout de la case à cocher « Afficher les valeurs », pour afficher/masquer les valeurs dans la vue arborescente
- Panneau de macro-commandes :
  - Les macro-commandes sont désormais numérotées, à partir de 1, comme les signaux et les images
- API de contrôle à distance (*RemoteProxy* et *LocalProxy*) :
  - *get_object_titles* accepte désormais « macro » comme nom de panneau et retourne la liste des titres de macro
  - Nouvelles méthodes *run_macro*, *stop_macro* et *import_macro_from_file*

Corrections de bugs :
- Version autonome - Intégration dans le menu de démarrage de Windows :
  - Correction du raccourci « Désinstaller » (non cliquable en raison d'un nom générique)
  - Traduction des raccourcis « Parcourir le répertoire d'installation » et « Désinstaller »
- Correction de l'"[Issue #55](https://github.com/DataLab-Platform/DataLab/issues/55) - Changer les limites de l'image dans la vue d'image n'a aucun effet sur les propriétés de l'objet image associé
- Correction de l'"[Issue #56](https://github.com/DataLab-Platform/DataLab/issues/56) - Plugin « Données de test » : *AttributeError : "NoneType" object has no attribute "data"* lors de l'annulation de « Créer une image avec des pics »
- Ceci corrige l'"[Issue #57](https://github.com/DataLab-Platform/DataLab/issues/57) - Les formes résultat cercle et ellipse ne sont pas transformées correctement
- Cycle de couleur et de style de courbe :
  - Avant cette version, ce cycle était géré par le même mécanisme pour le Panneau Signal ou l'Explorateur HDF5, ce qui n'était pas le comportement attendu
  - A présent, le cycle est géré séparément : l'Explorateur HDF5 ou l'Assistant d'importation de texte utilisent toujours la même couleur et le même style pour les courbes, et ils n'interfèrent pas avec le cycle du Panneau Signal

DataLab Version 0.12.0

Clarification de certains libellés des interfaces graphiques :

- Les onglets utilisés pour basculer entre les panneaux de données (signaux et images) et les composants de visualisation (« Panneau de courbes » et « Panneau d'images ») ont été renommés en « Panneau Signal » et « Panneau Image » (au lieu de « Signaux » et « Images »)
- Les composants de visualisation ont été renommés en « Vue Signal » et « Vue Image » (au lieu de « Panneau de courbes » et « Panneau d'images »)
- Les barres d'outils des panneaux de données ont été renommées en « Barre d'outils Signal » et « Barre d'outils Image » (au lieu de « Barre d'outils Traitement du Signal » et « Barre d'outils Traitement d'Image »)
- Améliorations ergonomiques : le « Panneau Signal » et le « Panneau Image » sont désormais affichés sur le côté gauche de la fenêtre principale, et la « Vue Signal » et la « Vue Image » sont affichées sur le côté droit de la fenêtre principale. Cela réduit la distance entre la liste des objets (signaux et images) et les actions associées (barres d'outils et menus), et rend l'interface plus intuitive et plus facile à utiliser

Nouvelle fonctionnalité de visite guidée et de démonstration :

- Lors du premier démarrage de DataLab, une visite guidée est désormais affichée à l'utilisateur pour présenter les principales fonctionnalités de l'application
- La visite guidée peut être relancée à tout moment depuis le menu « ? »
- Ajout d'une nouvelle fonctionnalité « Démonstration » dans le menu « ? »

Nouvel environnement Binder pour tester DataLab en ligne sans rien installer

Documentation :

- De nouveaux tutoriels textuels sont disponibles :
  - Mesure de la taille d'un faisceau laser
  - DataLab et Spyder : un mariage parfait
- Section « Premiers pas » : ajout de plus d'explications et de liens vers les tutoriels
- Nouvelle section « Contribuer » expliquant comment contribuer à DataLab, que vous soyez développeur ou non
- Nouvelle section « Macros » expliquant comment utiliser la fonctionnalité de macro-commandes
- Ajout du bouton « Copier » aux blocs de code dans la documentation

Nouvelles fonctionnalités :

- Nouvelle fonctionnalité d'assistant d'importation de fichiers texte :
  - Cette fonctionnalité permet d'importer des fichiers texte en tant que signaux ou images
  - L'utilisateur peut définir la source (presse-papiers ou fichier texte)
  - Ensuite, il est possible de définir le délimiteur, le nombre de lignes à sauter, le type de données de destination, etc.
- Ajout d'un menu dans le coin des onglets « Panneau Signal » et « Panneau Image » pour accéder rapidement aux fonctionnalités les plus utilisées (par exemple « Ajouter », « Supprimer », « Dupliquer », etc.)
- Fonctionnalité d'extraction de profil d'intensité :
  - Ajout d'une interface graphique pour extraire des profils d'intensité des images, pour les profils linéaires et moyennés
  - Les paramètres sont toujours directement modifiables par l'utilisateur (bouton « Modifier les paramètres du profil »)
  - Les paramètres sont désormais stockés d'une extraction de profil à l'autre
- Fonctionnalité de statistiques :
  - Ajout de *<y>/(y)* au tableau de résultats « Statistiques » du signal (en plus de la moyenne, de la médiane, de l'écart-type, etc.)
  - Ajout de *peak-to-peak* au tableau de résultats « Statistiques » du signal et de l'image
- Fonctionnalité d'ajustement de courbe : les résultats de l'ajustement sont désormais stockés dans un dictionnaire dans les métadonnées du signal (au lieu d'être stockés individuellement dans les métadonnées du signal)
- État de la fenêtre :
  - L'état des barres d'outils et des widgets de dock (visibilité, position, etc.) est désormais stocké dans le fichier de configuration et restauré au démarrage (la taille et la position étaient déjà stockées et restaurées)
  - Ceci implémente une partie de [Issue #30](https://github.com/DataLab-Platform/DataLab/issues/30) - Sauvegarder/restaurer la disposition de la fenêtre principale

Corrections de bugs :

- Correction de l"[Issue #41](https://github.com/DataLab-Platform/DataLab/issues/41) - Extraction du profil radial : impossible d'entrer les coordonnées du centre définies par l'utilisateur
- Correction de l"[Issue #49](https://github.com/DataLab-Platform/DataLab/issues/49) - Erreur lors de l'ouverture d'un fichier texte (UTF-8 BOM) en tant qu'image
- Correction de l"[Issue #51](https://github.com/DataLab-Platform/DataLab/issues/51) - Dimensions inattendues lors de l'ajout d'une nouvelle ROI sur une image avec des unités X/Y arbitraires (pas des pixels)
- Amélioration de la gestion de la sérialisation du style des items graphiques :
  - Avant cette version, le style des items graphiques était stocké dans les métadonnées du signal/image uniquement lors de la sauvegarde de l'espace de travail dans un fichier HDF5. Ainsi, lors de la modification du style d'un signal/image à partir du bouton « Paramètres » (barre d'outils de la vue), le style n'était pas conservé dans certains cas (par exemple lors de la duplication du signal/image).
  - A présent, le style des items graphiques est stocké dans les métadonnées du signal/image chaque fois que le style est modifié, et est restauré lors du rechargement de l'espace de travail
- Traitement de l'avertissement de conversion *ComplexWarning* lors de l'ajout de régions d'intérêt (ROI) à un signal avec des données complexes

DataLab Version 0.11.0

Nouvelles fonctionnalités :

- Les signaux et les images peuvent maintenant être réorganisés dans la vue arborescente :
  - En utilisant les nouvelles actions « Monter » et « Descendre » dans le menu « Édition » (ou en utilisant les boutons de la barre d'outils correspondants) :
  - Ceci corrige l"[Issue #22](https://github.com/DataLab-Platform/DataLab/issues/22) - Ajout des actions « monter/descendre » dans le menu « Édition », pour les signaux/images et les groupes
- Les signaux et les images peuvent également être réorganisés par glisser-déposer :
  - Les signaux et les images peuvent être déplacés et déposés dans leur propre panneau pour changer leur ordre
  - Les groupes peuvent également être déplacés et déposés dans leur panneau
  - La fonctionnalité prend également en charge la sélection multiple (en utilisant les modificateurs standard Ctrl et Shift), de sorte que plusieurs signaux/images/groupes peuvent être déplacés à la fois, pas nécessairement avec des positions contiguës
  - Ceci corrige l"[Issue #17](https://github.com/DataLab-Platform/DataLab/issues/17) - Ajout de la fonctionnalité de glisser-déposer aux vues arborescentes des signaux/images
- Nouvelles fonctionnalités d'interpolation 1D :
  - Ajout de la fonctionnalité « Interpolation » au menu « Traitement » du panneau de signal
  - Méthodes disponibles : linéaire, spline, quadratique, cubique, barycentrique et PCHIP
  - Merci à [@marcel-goldschen-ohm](https://github.com/marcel-goldschen-ohm) pour sa contribution à l'interpolation spline
  - Ceci corrige l"[Issue #20](https://github.com/DataLab-Platform/DataLab/issues/20) - Ajout des fonctionnalités d'interpolation 1D
- Nouvelle fonctionnalité de rééchantillonnage 1D :
  - Ajout de la fonctionnalité « Rééchantillonnage » au menu « Traitement » du panneau de signal
  - Mêmes méthodes d'interpolation que pour la fonctionnalité « Interpolation »
  - Possibilité de spécifier le pas de rééchantillonnage ou le nombre de points
  - Ceci corrige l"[Issue #21](https://github.com/DataLab-Platform/DataLab/issues/21) - Ajout de la fonctionnalité de rééchantillonnage 1D
- Nouvelle fonctionnalité de convolution 1D :
  - Ajout de la fonctionnalité « Convolution » au menu « Opération » du panneau de signal
  - Ceci corrige l"[Issue #23](https://github.com/DataLab-Platform/DataLab/issues/23) - Ajout de la fonctionnalité de convolution 1D
- Nouvelle fonctionnalité de d'élimination de tendance 1D :
  - Ajout de la fonctionnalité « Élimination de tendance » au menu « Traitement » du panneau de signal
  - Méthodes disponibles : linéaire ou constante

    - Ceci corrige l"[Issue #24](https://github.com/DataLab-Platform/DataLab/issues/24) - Ajout de la fonctionnalité d'élimination de tendance 1D
- Résultats de calcul 2D :
    - Avant cette version, les résultats de calcul 2D tels que les contours, les blobs, etc. étaient stockés dans le dictionnaire de métadonnées de l'image sous forme de coordonnées (x0, y0, x1, y1, …) même pour les cercles et les ellipses (c'est-à-dire les coordonnées des rectangles englobants).
    - Par souci de commodité, les coordonnées du cercle et de l'ellipse sont désormais stockées dans le dictionnaire de métadonnées de l'image sous la forme (x0, y0, rayon) et (x0, y0, a, b, theta) respectivement.
    - Ces résultats sont également affichés comme tels dans la boîte de dialogue « Résultats » (à la fin du processus de calcul ou en cliquant sur le bouton « Afficher les résultats »).
    - Cette correction corrige l"[Issue #32](https://github.com/DataLab-Platform/DataLab/issues/32) - Détection des contours : afficher le cercle *(x, y, r)* et l'ellipse *(x, y, a, b, theta)* au lieu de *(x0, y0, x1, x1, …)*
- Résultats de calcul 1D et 2D :
    - En complément de l'amélioration précédente, plus de résultats de calcul sont désormais affichés dans la boîte de dialogue « Résultats »
    - Cela concerne à la fois les résultats de calcul 1D (FHWM, …) et 2D (contours, blobs, …) :
        - Les résultats de type segment affichent désormais également la longueur (L) et les coordonnées du centre (Xc, Yc)
        - Les résultats de type cercle et ellipse affichent désormais également l'aire (A)
- Ajout de l'entrée « Tracer les résultats » dans le menu « Calculs » :
    - Cette fonctionnalité permet de tracer les résultats de calcul (1D ou 2D)
    - Cela crée un nouveau signal avec des axes X et Y correspondant à des paramètres définis par l'utilisateur (par exemple X = index et Y = rayon pour les résultats de cercle)
- Augmentation de la largeur par défaut de la boîte de dialogue de sélection d'objet :
    - La boîte de dialogue de sélection d'objet est désormais plus large par défaut, de sorte que les titres complets des signaux/images/groupes soient plus facilement lisibles
- Fonctionnalité de suppression de métadonnées :
    - Avant cette version, la fonctionnalité supprimait toutes les métadonnées, y compris les métadonnées des régions d'intérêt (ROI), le cas échéant.
    - A présent, une boîte de dialogue de confirmation est affichée à l'utilisateur avant de supprimer toutes les métadonnées si le signal/l'image a des métadonnées ROI : cela permet de conserver les métadonnées ROI si nécessaire.
- Extraction de profil d'image : ajout de la prise en charge des images masquées (lors de la définition des régions d'intérêt, les zones en dehors des ROI sont masquées, et le profil est extrait uniquement sur les zones non masquées, ou moyenné sur les zones non masquées dans le cas de l'extraction du profil moyen)
- Style de courbe : ajout de « Réinitialiser les styles de courbe » dans le menu « Affichage ». Cette fonctionnalité permet de réinitialiser le cycle de style de courbe à son état initial.
- Classe de base des plugins *PluginBase* :
    - Ajout de la méthode *edit_new_signal_parameters* pour afficher une boîte de dialogue pour éditer les paramètres d'un nouveau signal
    - Ajout de la méthode *edit_new_image_parameters* pour afficher une boîte de dialogue pour éditer les paramètres d'une nouvelle image (mise à jour du plugin *cdl_testdata.py* en conséquence)
- API de calculs sur les signaux et les images (*cdl.core.computations*) :
    - Ajout de wrappers pour les calculs 1 -> 1 sur les signaux et les images
    - Ces wrappers visent à simplifier la création d'une fonction de calcul de base opérant sur les objets natifs de DataLab (*SignalObj* et *ImageObj*) à partir d'une fonction opérant sur des tableaux NumPy
    - Cela simplifie les fonctionnalités internes de DataLab et facilite la création de nouvelles fonctionnalités de calcul dans les plugins
    - Voir le plugin d'exemple *cdl_custom_func.py* pour un cas d'utilisation pratique
- Ajout de la fonctionnalité « Extraction du profil radial » au menu « Opération » du panneau d'image :
    - Cette fonctionnalité permet d'extraire un profil moyenné radialement à partir d'une image
    - Le profil est extrait autour d'un centre défini par l'utilisateur (x0, y0)
    - Le centre peut également être calculé (centre de gravité ou centre de l'image)
- Suite de tests automatisés :

    - Depuis la version 0.10, l'objet proxy de DataLab dispose d'une méthode *toggle_auto_refresh* pour activer/désactiver la fonctionnalité « Auto-refresh ». Cette fonctionnalité peut être utile pour améliorer les performances lors de l'exécution de scripts de test
    - Les scénarios de test sur les signaux et les images utilisent désormais cette fonctionnalité pour améliorer les performances
  - Métadonnées des signaux et des images :
    - Ajout de l'entrée « source » dans le dictionnaire de métadonnées, pour stocker le chemin du fichier source lors de l'importation d'un signal ou d'une image à partir d'un fichier
    - Ce champ est conservé lors du traitement du signal/image, afin de garder une trace du chemin du fichier source

Documentation :

  - Nouvelle [section Tutoriels](https://datalab-platform.com/fr/intro/tutorials/index.html) dans la documentation :
    - Cette section fournit un ensemble de tutoriels pour apprendre à utiliser DataLab
    - Les tutoriels vidéo suivants sont disponibles :
      - Démo rapide
      - Ajout de vos propres fonctionnalités
    - Les tutoriels textuels suivants sont disponibles :
      - Traitement d'un spectre
      - Détection de taches sur une image
      - Mesure de franges Fabry-Perot
      - Prototypage d'un pipeline de traitement personnalisé
  - Nouvelle [section API](https://datalab-platform.com/fr/api/index.html) dans la documentation :
    - Cette section explique comment utiliser DataLab comme une bibliothèque Python, en abordant les sujets suivants :
      - Comment utiliser les algorithmes DataLab sur les tableaux NumPy
      - Comment utiliser les fonctionnalités de calcul de DataLab sur les objets DataLab (signaux et images)
      - Comment utiliser les fonctionnalités d'entrée/sortie de DataLab
      - Comment utiliser les objets proxy pour contrôler DataLab à distance
    - Cette section fournit également une référence complète de l'API pour les objets et les fonctionnalités de DataLab
    - Ceci corrige l"[Issue #19](https://github.com/DataLab-Platform/DataLab/issues/19) - Ajout de la documentation de l'API (modèle de données, fonctions sur les tableaux ou les objets de signaux/images, …)

Corrections de bugs :

  - Ceci corrige l"[Issue #29](https://github.com/DataLab-Platform/DataLab/issues/29) - Erreur d'ajustement polynomial : *QDialog […] argument 1 has an unexpected type "SignalProcessor"*
  - Fonctionnalité d'extraction de ROI d'image :
    - Avant cette version, lors de l'extraction d'une seule ROI circulaire d'une image avec l'option « Extraire toutes les régions d'intérêt dans un seul objet d'image » activée, le résultat était une seule image sans le masque de ROI (le masque de ROI n'était disponible que lors de l'extraction de ROI avec l'option désactivée)
    - Cela conduisait à un comportement inattendu, car on pouvait interpréter le résultat (une image carrée sans le masque de ROI) comme le résultat d'une seule ROI rectangulaire
    - A présent, lors de l'extraction d'une seule ROI circulaire d'une image avec l'option « Extraire toutes les régions d'intérêt dans un seul objet d'image » activée, le résultat est une seule image avec le masque de ROI (comme si l'option était désactivée)
    - Ceci corrige l"[Issue #31](https://github.com/DataLab-Platform/DataLab/issues/31) - Extraction d'une seule ROI circulaire : bascule automatique vers la fonction *extract_single_roi*
  - Calcul sur ROI circulaire :
    - Avant cette version, lors de l'exécution de calculs sur une ROI circulaire, les résultats étaient inattendus en termes de coordonnées (les résultats semblaient être calculés dans une région située au-dessus de la ROI réelle).
    - Cela était dû à une régression introduite dans une version antérieure.
    - A présent, lors de la définition d'une ROI circulaire et de l'exécution de calculs sur celle-ci, les résultats sont calculés sur la ROI réelle
    - Ceci corrige l"[Issue #33](https://github.com/DataLab-Platform/DataLab/issues/33) - Calcul sur ROI cir-

culaire : résultats inattendus
- Détection des contours sur ROI :
  - Avant cette version, lors de l'exécution de la détection des contours sur une ROI, certains contours étaient détectés en dehors de la ROI (cela peut être dû à une limitation de la fonction *find_contours* de scikit-image).
  - A présent, grâce à une solution de contournement, les contours erronés sont filtrés.
  - Un nouveau module de test *cdl.tests.features.images.contour_fabryperot_app* a été ajouté pour tester la fonctionnalité de détection des contours sur une image Fabry-Perot (merci à [@emarin2642](https://github.com/emarin2642) pour sa contribution)
  - Ceci corrige l"[Issue #34](https://github.com/DataLab-Platform/DataLab/issues/34) - Détection des contours : résultats inattendus en dehors de la ROI
- Fusion des résultats de calcul :
  - Avant cette version, lors d'un calcul *1->N* (somme, moyenne, produit) sur un groupe de signaux/images, les résultats de calcul associés à chaque signal/image étaient fusionnés en un seul résultat, mais seul le type de résultat présent dans le premier signal/image était conservé.
  - A présent, les résultats de calcul associés à chaque signal/image sont fusionnés en un seul résultat, quel que soit le type de résultat.
- Ceci corrige l"[Issue #36](https://github.com/DataLab-Platform/DataLab/issues/36) - L'état d'activation de l'action « Tout supprimer » n'est parfois pas rafraîchi
- Inversion X/Y de l'image : lors de l'inversion des axes X et Y, les régions d'intérêt (ROI) n'étaient ni supprimées ni inversées (les ROI sont désormais supprimées, jusqu'à ce que nous implémentions la fonction d'inversion, si demandé)
- Groupe « Propriétés » : le bouton « Appliquer » était activé par défaut, même lorsqu'aucune propriété n'avait été modifiée, ce qui était déroutant pour l'utilisateur (le bouton « Appliquer » est désormais désactivé par défaut, et n'est activé que lorsqu'une propriété est modifiée)
- Correction de la méthode *get_object* du proxy lorsqu'il n'y a pas d'objet à retourner (*None* est retourné au lieu d'une exception)
- Correction de *IndexError : list index out of range* lors de certaines opérations ou calculs sur des groupes de signaux/images (par exemple « Extraction de ROI », « Détection de pics », « Redimensionnement », etc.)
- Glisser-déposer depuis un gestionnaire de fichiers : les noms de fichiers sont désormais triés par ordre alphabétique

DataLab Version 0.10.1

Note : la version 0.10.0 a été presque immédiatement remplacée par la version 0.10.1 en raison d'une correction de bogue de dernière minute

Nouvelles fonctionnalités :
- Fonctionnalités communes aux signaux et aux images :
  - Ajout des fonctionnalités « Partie réelle » et « Partie imaginaire » au menu « Opération »
  - Ajout de la fonctionnalité « Convertir le type de données » au menu « Opération »
- Fonctionnalités ajoutées suite aux demandes des utilisateurs (rencontre du 18/12/2023 au CEA) :
  - Les styles de courbe et d'image sont désormais enregistrés dans le fichier HDF5 :
    - Le style de courbe couvre les propriétés suivantes : couleur, style de ligne, largeur de ligne, style de marqueur, taille de marqueur, couleur de bordure de marqueur, couleur de remplissage de marqueur, etc.
    - Le style d'image couvre les propriétés suivantes : colormap, interpolation, etc.
    - Ces propriétés étaient déjà persistantes pendant la session de travail, mais étaient perdues lors de l'enregistrement et du rechargement du fichier HDF5
    - Désormais, ces propriétés sont enregistrées dans le fichier HDF5 et sont restaurées lors du rechargement du fichier HDF5
  - Nouvelles fonctionnalités d'extraction de profil pour les images :
    - Ajout de la fonctionnalité « Profil rectiligne » au menu « Opérations », pour extraire un profil d'une image le long d'une ligne ou d'une colonne

  - Ajout de la fonctionnalité « Profil moyen » au menu « Opérations », pour extraire le profil moyen sur une zone rectangulaire d'une image, le long d'une ligne ou d'une colonne
- La plage LUT de l'image (paramètres de contraste/luminosité) est désormais enregistrée dans le fichier HDF5 :
  - Comme pour les styles de courbe et d'image, la plage LUT était déjà persistante pendant la session de travail, mais était perdue lors de l'enregistrement et du rechargement du fichier HDF5
  - Désormais, la plage LUT est enregistrée dans le fichier HDF5 et est restaurée lors du rechargement
- Ajout des actions « Rafraîchissement automatique » et « Rafraîchissement manuel » dans le menu « Affichage » (et la barre d'outils principale) :
  - Lorsque « Rafraîchissement automatique » est activé (par défaut), la vue du graphique est automatiquement rafraîchie lorsqu'un signal/image est modifié, ajouté ou supprimé. Même si le rafraîchissement est optimisé, cela peut entraîner des problèmes de performances lors de l'utilisation de grands ensembles de données.
  - Lorsqu'il est désactivé, la vue du graphique n'est pas automatiquement rafraîchie. L'utilisateur doit rafraîchir manuellement la vue du graphique en cliquant sur le bouton « Rafraîchir manuellement » de la barre d'outils principale ou en appuyant sur la touche de rafraîchissement standard (par exemple « F5 »).
- Ajout de la méthode *toggle_auto_refresh* à l'objet proxy de DataLab :
  - Cette méthode permet d'activer/désactiver la fonctionnalité « Rafraîchissement automatique » à partir d'une macro-commande, d'un plugin ou d'un client de contrôle à distance.
  - Un gestionnaire de contexte *context_no_refresh* est également disponible pour désactiver temporairement la fonctionnalité « Rafraîchissement automatique » à partir d'une macro-commande, d'un plugin ou d'un client de contrôle à distance. Utilisation typique :

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

- Amélioration de la lisibilité des courbes :
  - Jusqu'à cette version, le style de la courbe était automatiquement défini en faisant défiler les styles prédéfinis de **PlotPy**
  - Cependant, certains styles ne sont pas adaptés à la lisibilité des courbes (par exemple, les couleurs « cyan » et « jaune » ne sont pas lisibles sur un fond blanc, en particulier lorsqu'elles sont combinées avec un style de ligne « pointillée »)
  - Cette version introduit une nouvelle gestion du style de courbe avec des couleurs qui sont distinguables et accessibles, même pour les personnes atteintes de déficience de la vision des couleurs
- Ajout de la fonctionnalité « Anti-aliasing de la courbe » au menu « Affichage » (et à la barre d'outils) :
  - Cette fonctionnalité permet d'activer/désactiver l'anti-aliasing de la courbe (par défaut : activé)
  - Lorsqu'il est activé, le rendu de la courbe est plus lisse mais peut entraîner des problèmes de performances lors de l'utilisation de grands ensembles de données (c'est pourquoi il peut être désactivé)
- Ajout de la méthode *toggle_show_titles* à l'objet proxy de DataLab. Cette méthode permet d'activer/désactiver la fonctionnalité « Afficher les titres des objets graphiques » à partir d'une macro-commande, d'un plugin ou d'un client de contrôle à distance.
- Le client distant vérifie désormais la version du serveur et affiche un message d'avertissement si la version du serveur n'est peut-être pas entièrement compatible avec la version du client.

Corrections de bugs :

- Fonctionnalité de détection des contours de l'image (menu « Calculer ») :
  - La fonctionnalité de détection des contours ne prenait pas en compte le paramètre « shape » (cercle, ellipse, polygone) lors du calcul des contours. Le paramètre était stocké mais vraiment utilisé uniquement lors de l'appel de la fonctionnalité une deuxième fois.
  - Ce comportement involontaire a conduit à une *AssertionError* lors du choix de « polygone » comme forme de contour et de la tentative de calcul des contours pour la première fois.
  - Ceci est maintenant corrigé (voir [Issue #9](https://github.com/DataLab-Platform/DataLab/issues/9) - Détection des contours de l'image : *AssertionError* lors du choix de « polygone » comme forme de contour)
- Raccourcis clavier :

  - Les raccourcis clavier pour les actions « Nouveau », « Ouvrir », « Enregistrer », « Dupliquer », « Supprimer », « Tout supprimer » et « Rafraîchir manuellement » ne fonctionnaient pas correctement.
  - Ces raccourcis étaient spécifiques à chaque panneau de signal/image et ne fonctionnaient que lorsque le panneau sur lequel le raccourci a été pressé pour la première fois était actif (lorsqu'il était activé à partir d'un autre panneau, le raccourci ne fonctionnait pas et un message d'avertissement était affiché dans la console, par exemple *QAction : :event : Ambiguous shortcut overload : Ctrl+C*)
  - De plus, les raccourcis ne fonctionnaient pas au démarrage (lorsqu'aucun panneau n'avait le focus).
  - Ceci est maintenant corrigé : les raccourcis fonctionnent maintenant quel que soit le panneau actif, et même au démarrage (voir [Issue #10](https://github.com/DataLab-Platform/DataLab/issues/10) - Raccourcis clavier ne fonctionnant pas correctement : *QAction : :event : Ambiguous shortcut overload : Ctrl+C*)
- Les actions « Afficher les titres des objets graphiques » et « Rafraîchissement automatique » ne fonctionnaient pas correctement :
  - Les actions « Afficher les titres des objets graphiques » et « Rafraîchissement automatique » ne fonctionnaient que sur le panneau de signal/image actif, et non sur tous les panneaux.
  - Ceci est maintenant corrigé (voir [Issue #11](https://github.com/DataLab-Platform/DataLab/issues/11) - Les actions « Afficher les titres des objets graphiques » et « Rafraîchissement automatique » ne fonctionnaient que sur le panneau de signal/image actuel)
- Correction de l"[Issue #14](https://github.com/DataLab-Platform/DataLab/issues/14) - Enregistrer/Réouvrir le projet HDF5 sans nettoyage entraîne une *ValueError*
- Correction de l"[Issue #15](https://github.com/DataLab-Platform/DataLab/issues/15) - MacOS : 1. Erreur *pip install cdl* - 2. Menus manquants :
  - Partie 1 : l'erreur *pip install cdl* sur MacOS était en fait un problème de **PlotPy** (voir [ce problème]
  - Partie 2 : les menus manquants sur MacOS étaient dus à un bogue PyQt/MacOS concernant les menus dynamiques
- Format de fichier HDF5 : lors de l'importation d'un ensemble de données HDF5 en tant que signal ou image, les attributs de l'ensemble de données étaient systématiquement copiés dans les métadonnées du signal/image : nous ne copions désormais que les attributs qui correspondent aux types de données standard (entiers, flottants, chaînes de caractères) pour éviter les erreurs lors de la sérialisation/désérialisation de l'objet signal/image
- Visionneuse d'installation/configuration : amélioration de la lisibilité (suppression de la coloration syntaxique)
- Fichier de spécification de PyInstaller : ajout manuel des fichiers de données *skimage* manquants afin de continuer à prendre en charge Python 3.8 (voir [Issue #12](https://github.com/DataLab-Platform/DataLab/issues/12) - Version autonome sur Windows 7 : *api-ms-win-core-path-l1-1-0.dll* manquant)
- Correction de l"[Issue #13](https://github.com/DataLab-Platform/DataLab/issues/13) - ArchLinux : *qt.qpa.plugin : Could not load the Qt platform plugin « xcb » in « » even though it was found*

DataLab Version 0.9.2

Corrections de bugs :

- Fonctionnalité d'extraction de la région d'intérêt (ROI) pour les images :
  - L'extraction de la ROI ne fonctionnait pas correctement lorsque l'option « Extraire toutes les régions d'intérêt dans un seul objet image » était activée s'il n'y avait qu'une seule ROI définie. Le résultat était une image positionnée à l'origine (0, 0) au lieu de la position attendue (x0, y0) et le rectangle de la ROI lui-même n'était pas supprimé comme prévu. Ceci est maintenant corrigé (voir [Issue #6](https://github.com/DataLab-Platform/DataLab/issues/6) - Fonctionnalité "Extraire plusieurs ROI" : résultat inattendu pour une seule ROI)
  - Les rectangles ROI avec des coordonnées négatives n'étaient pas correctement gérés : l'extraction de la ROI levait une exception *ValueError*, et le masque de l'image n'était pas correctement affiché. Ceci est maintenant corrigé (voir [Issue #7](https://github.com/DataLab-Platform/DataLab/issues/7) - Extraction de la ROI de l'image : *ValueError : zero-size array to reduction operation minimum which has no identity*)
  - L'extraction de la ROI ne prenait pas en compte la taille des pixels (dx, dy) et l'origine (x0, y0) de l'image. Ceci est maintenant corrigé (voir [Issue #8](https://github.com/DataLab-Platform/DataLab/issues/8) - Extraction de la ROI de l'image : prendre en compte la taille des pixels)
- La console de macro-commande est désormais en lecture seule :

- La console Python du panneau de macro-commande ne prend actuellement pas en charge le flux d'entrée standard (*stdin*) et c'est voulu (du moins pour l'instant)
- Définir la console Python en lecture seule pour éviter toute confusion

DataLab Version 0.9.1

Corrections de bugs :

- La traduction française n'est pas disponible sur Windows/Version autonome :
  - La locale n'était pas correctement détectée sur Windows pour la version autonome (gelée avec *pyinstaller*) en raison d'un problème avec *locale.getlocale()* (fonction renvoyant *None* au lieu de la locale attendue sur les applications gelées)
  - C'est finalement un problème de *pyinstaller*, mais une solution de contournement a été implémentée dans *guidata* V3.2.2 (voir [Issue guidata #68](https://github.com/PlotPyStack/guidata/issues/68) - Windows : la traduction gettext ne fonctionne pas sur les applications gelées)
  - [Issue #2](https://github.com/DataLab-Platform/DataLab/issues/2) - La traduction française n'est pas disponible sur la version autonome de Windows
- L'enregistrement d'une image au format JPEG2000 échoue pour les données non entières :
  - L'encodeur JPEG2000 ne prend pas en charge les données non entières ou les données entières signées
  - Avant, DataLab affichait un message d'erreur lors de la tentative de sauvegarde de données incompatibles au format JPEG2000 : cela n'était pas cohérent avec le comportement des autres formats d'image standard (par exemple PNG, JPG, etc.) pour lesquels DataLab convertissait automatiquement les données au format approprié (entier non signé sur 8 bits)
  - Le comportement actuel est maintenant cohérent avec les autres formats d'image standard : lors de la sauvegarde au format JPEG2000, DataLab convertit automatiquement les données en entier non signé sur 8 bits ou en entier non signé sur 16 bits (en fonction du type de données original)
  - [Issue #3](https://github.com/DataLab-Platform/DataLab/issues/3) - Sauvegarde d'image au format JPEG2000 : "OSError : erreur d'encodeur -2 lors de l'écriture du fichier image"
- Les raccourcis de la version autonome de Windows ne s'affichent pas dans le menu de démarrage de l'utilisateur actuel :
  - Lors de l'installation de DataLab sur Windows à partir d'un compte non administrateur, les raccourcis ne s'affichaient pas dans le menu de démarrage de l'utilisateur actuel, mais dans le menu de démarrage de l'administrateur (en raison des privilèges élevés de l'installeur et du fait que l'installeur ne prend pas en charge l'installation de raccourcis pour tous les utilisateurs)
  - Désormais, l'installeur ne demande plus de privilèges élevés, et les raccourcis sont installés dans le menu de démarrage de l'utilisateur actuel (cela signifie également que l'utilisateur actuel doit disposer d'un accès en écriture au répertoire d'installation)
  - Dans les futures versions, l'installeur prendra en charge l'installation de raccourcis pour tous les utilisateurs s'il y a une demande à cet effet (voir [Issue #5](https://github.com/DataLab-Platform/DataLab/issues/5))
  - [Issue #4](https://github.com/DataLab-Platform/DataLab/issues/4) - Windows : les raccourcis de la version autonome ne s'affichent pas dans le menu de démarrage de l'utilisateur actuel
- Fenêtre d'installation et de configuration pour la version autonome :
  - Ne plus afficher le message d'erreur ambigu "Dépendances invalides"
  - Les dépendances sont censées être vérifiées lors de la construction de la version autonome
- Ajout de la documentation PDF à la version autonome :
  - La documentation PDF était manquante dans la version précédente
  - Désormais, la documentation PDF (en anglais et en français) est incluse dans la version autonome

DataLab Version 0.9.0

Nouvelles dépendances :
- DataLab est désormais alimenté par [PlotPyStack](https://github.com/PlotPyStack) :
  - PythonQwt
  - guidata
  - PlotPy
- [opencv-python](https://pypi.org/project/opencv-python/) (algorithmes de traitement d'image)

Nouvelle plateforme de référence :
- DataLab est validé sur Windows 11 avec Python 3.11 et PyQt 5.15
- DataLab est également compatible avec d'autres systèmes d'exploitation (Linux, MacOS) et d'autres liaisons Python-Qt et versions (Python 3.8-3.12, PyQt6, PySide6)

Nouvelles fonctionnalités :
- DataLab est une plateforme :
  - Ajout de la prise en charge des plugins
    - Fonctionnalités de traitement personnalisées disponibles dans le menu « Plugins »
    - Fonctionnalités d'E/S personnalisées : de nouveaux formats de fichier peuvent être ajoutés aux fonctionnalités d'E/S standard pour les signaux et les images
    - Fonctionnalités HDF5 personnalisées : de nouveaux formats de fichier HDF5 peuvent être ajoutés à la fonctionnalité d'importation HDF5 standard
    - D'autres fonctionnalités à venir…
  - Ajout de la fonctionnalité de contrôle à distance : DataLab peut être contrôlé à distance via une connexion TCP/IP (voir [Contrôle à distance](https://datalab-platform.com/fr/remote_control.html))
  - Ajout de commandes de macro : DataLab peut être contrôlé via un fichier macro (voir [Commandes de macro](https://datalab-platform.com/fr/macro_commands.html))
- Fonctionnalités générales :
  - Ajout de la boîte de dialogue des paramètres (voir l'entrée « Paramètres » dans le menu « Fichier ») :
    - Paramètres généraux
    - Paramètres de visualisation
    - Paramètres de traitement
    - Etc.
  - Nouvelle disposition par défaut : les panneaux de signaux/images sont sur le côté droit de la fenêtre principale, les panneaux de visualisation sont sur le côté gauche avec une barre d'outils verticale
- Fonctionnalités de signal/image :
  - Ajout de l'isolation des processus : chaque signal/image est traité dans un processus séparé, de sorte que DataLab ne se fige plus lors du traitement de signaux/images volumineux
  - Ajout de la prise en charge des groupes : les signaux et les images peuvent être regroupés, et des opérations peuvent être appliquées à tous les objets d'un groupe ou entre les groupes
  - Ajout de boîtes de dialogue d'avertissement et d'erreur avec des liens de poursuite détaillés vers le code source (les avertissements peuvent être ignorés en option)
  - Amélioration significative des performances lors de la sélection d'objets
  - Optimisation des performances lors de l'affichage d'images volumineuses
  - Ajout de la prise en charge du dépôt de fichiers sur le panneau de signal/image
  - Ajout de la boîte de groupe « Paramètres de calcul » pour afficher les paramètres d'entrée du dernier résultat
  - Ajout de la fonctionnalité « Copier les titres dans le presse-papiers » dans le menu « Édition »
  - Pour chaque fonctionnalité de traitement individuelle (menus opération, traitement et calcul), les paramètres saisis (boîtes de dialogue) sont stockés en cache pour être utilisés par défaut la prochaine fois que la fonctionnalité est utilisée
- Traitement de signal :
  - Ajout de la prise en charge du décalage FFT facultatif (voir la boîte de dialogue des paramètres)
- Traitement d'image :
  - Ajout de l'opération de binning de pixels (facteurs de binning X/Y, opération : somme, moyenne, …)
  - Ajout des options « Distribuer sur une grille » et « Réinitialiser les positions des images » dans le menu des opérations

- Ajout du filtre de Butterworth
- Ajout des fonctionnalités de traitement de l'exposition :
  - Correction gamma
  - Correction logarithmique
  - Correction sigmoïde
- Ajout des fonctionnalités de traitement de la restauration :
  - Filtre de débruitage par variation totale (TV Chambolle)
  - Filtre bilatéral (débruitage)
  - Filtre de débruitage par ondelettes
  - Filtre de débruitage White Top-Hat
- Ajout des transformations morphologiques (empreinte de disque) :
  - White Top-Hat
  - Black Top-Hat
  - Érosion
  - Dilatation
  - Ouverture
  - Fermeture
- Ajout des fonctionnalités de détection des contours :
  - Filtre de Roberts
  - Filtre de Prewitt (vertical, horizontal, les deux)
  - Filtre de Sobel (vertical, horizontal, les deux)
  - Filtre de Scharr (vertical, horizontal, les deux)
  - Filtre de Farid (vertical, horizontal, les deux)
  - Filtre de Laplace
  - Filtre de Canny
- Détection des contours : ajout de la prise en charge des contours polygonaux (en plus des contours de cercle et d'ellipse)
- Ajout de la transformation de Hough pour les cercles (détection de cercles)
- Ajout du rééchelonnement des niveaux d'intensité de l'image
- Ajout de l'égalisation d'histogramme
- Ajout de l'égalisation d'histogramme adaptative
- Ajout des méthodes de détection de blobs :
  - Différence de Gaussienne
  - Méthode du déterminant de Hessian
  - Laplacien de Gaussienne
  - Détection de blobs à l'aide d'OpenCV
- Les formes et les annotations des résultats sont maintenant transformées (au lieu d'être supprimées) lors de l'exécution de l'une des opérations suivantes :
  - Rotation (angle arbitraire, +90°, -90°)
  - Symétrie (verticale/horizontale)
- Ajout de la prise en charge du décalage FFT facultatif (voir la boîte de dialogue des paramètres)
- Console : ajout d'un éditeur externe configurable (par défaut : VSCode) pour suivre les liens de poursuite vers le code source

---

Note : DataLab a été créé par Codra/Pierre Raybaut en 2023. Il est développé et maintenu par l'équipe de développement de la plateforme DataLab.

Index des modules Python

C

cdl.algorithms, 185
cdl.algorithms.coordinates, 193
cdl.algorithms.datatypes, 192
cdl.algorithms.fit, 194
cdl.algorithms.image, 188
cdl.algorithms.signal, 186
cdl.core.computation, 219
cdl.core.computation.base, 220
cdl.core.computation.image, 228
cdl.core.computation.image.detection, 243
cdl.core.computation.image.edges, 241
cdl.core.computation.image.exposure, 237
cdl.core.computation.image.morphology, 240
cdl.core.computation.image.restoration, 239
cdl.core.computation.signal, 221
cdl.core.gui.processor.image, 101
cdl.core.gui.processor.signal, 99
cdl.obj, 200
cdl.param, 195
cdl.plugins, 114
cdl.proxy, 245