DataLab

Version 1.0.1

nov. 18, 2025

Table des matières

1 Premiers pas **3**
1.1 Installation . . . 4
1.2 Cas d'usage, principales fonctionnalités et points forts . . . 10
1.3 Fonctionnalités principales . . . 12
1.4 Tutoriels . . . 15

2 Fonctionnalités **91**
2.1 Vue d'ensemble et fonctionnalités communes . . . 91
2.2 Traitement du signal . . . 105
2.3 Traitement d'image . . . 144
2.4 Validation . . . 189
2.5 Fonctionnalités avancées . . . 198

3 Contribuer **305**
3.1 Partagez vos idées et vos expériences . . . 305
3.2 Partagez vos connaissances scientifiques/techniques . . . 305
3.3 Contribuer aux nouvelles fonctionnalités . . . 306
3.4 Développer de nouvelles fonctionnalités . . . 306

4 Versions **321**
4.1 DataLab Version 1.0.1 . . . 321
4.2 DataLab Version 0.20.0 . . . 328
4.3 DataLab Version 0.19.2 . . . 329
4.4 DataLab Version 0.19.1 . . . 330
4.5 DataLab Version 0.19.0 . . . 330
4.6 DataLab Version 0.18.2 . . . 332
4.7 DataLab Version 0.18.1 . . . 333
4.8 DataLab Version 0.18.0 . . . 334
4.9 DataLab Version 0.17.1 . . . 334
4.10 DataLab Version 0.17.0 . . . 335
4.11 DataLab Version 0.16.4 . . . 337
4.12 DataLab Version 0.16.3 . . . 337
4.13 DataLab Version 0.16.2 . . . 337
4.14 DataLab Version 0.16.1 . . . 338
4.15 DataLab Version 0.16.0 . . . 339
4.16 DataLab Version 0.15.1 . . . 341
4.17 DataLab Version 0.15.0 . . . 342

4.18 DataLab Version 0.14.2 343
4.19 DataLab Version 0.14.1 343
4.20 DataLab Version 0.14.0 344
4.21 DataLab Version 0.12.0 345
4.22 DataLab Version 0.11.0 346
4.23 DataLab Version 0.10.1 349
4.24 DataLab Version 0.9.2 351
4.25 DataLab Version 0.9.1 352
4.26 DataLab Version 0.9.0 352

Index des modules Python **355**

DataLab est une **plateforme open-source de traitement et de visualisation de signaux et d'images** pour la recherche, l'éducation et l'industrie. S'appuyant sur la richesse de l'écosystème scientifique Python [1], DataLab est le complément idéal de vos flux de travail d'analyse de données car extensible avec votre code Python via *Plugins* ou directement depuis *votre IDE* ou *vos notebooks Jupyter*. Rendez-vous sur *Installation* pour démarrer !

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.

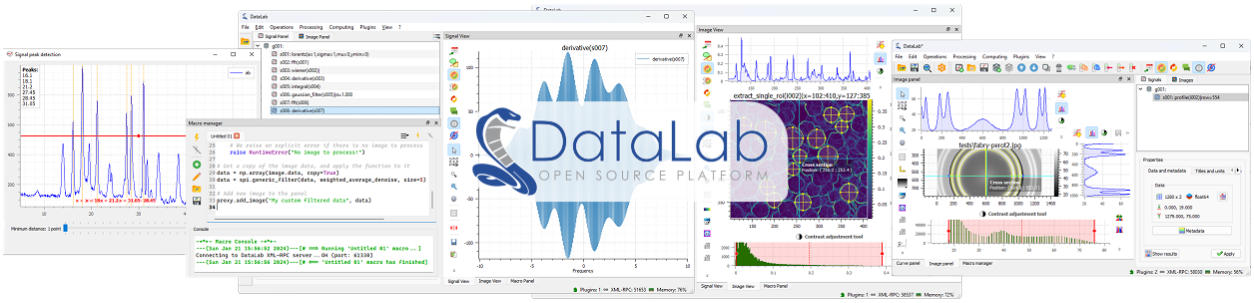


Fig. 1 – Visualisation de signaux et d'images dans DataLab

DataLab a été financé, chronologiquement, par les parties prenantes suivantes :

CEA, le Commissariat à l'énergie atomique et aux énergies alternatives, financeur principal de DataLab, contribue significativement au projet.
CODRA, une société d'ingénierie et d'édition de logiciels, a soutenu le parcours open-source de DataLab depuis ses débuts (voir ici).
NLnet Foundation, dans le cadre du NGI0 Commons Fund, soutenu par la Commission européenne, a financé la refonte de l'architecture centrale de DataLab.

Powered by PlotPyStack

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

Sigima

Fig. 3 – Les fonctionnalités de traitement de DataLab sont basées sur Sigima, la bibliothèque open-source de traitement du signal et de l'image (faisant partie de la plateforme DataLab).

1. DataLab processing primitives are mainly based on Sigima, NumPy, SciPy, scikit-image, OpenCV and PyWavelets libraries. DataLab visualization capabilities are based on PlotPyStack toolkit, a set of Python libraries for building scientific applications with Qt graphical user interfaces. DataLab advanced and specialized processing features are based on Sigima, the open-source signal and image processing library (part of the DataLab Platform).

CHAPITRE 1

Premiers pas

DataLab est une plateforme ouverte de traitement de signaux et d'images, conçue pour être utilisée par les scientifiques, ingénieurs et chercheurs du monde académique et industriel, tout en offrant la fiabilité d'un logiciel industriel. C'est un logiciel polyvalent qui peut être utilisé pour un large éventail d'applications, de l'analyse de données simple aux tâches complexes de traitement de signaux et d'analyse d'images.

DataLab s'intègre parfaitement dans votre flux de travail grâce à trois modes de fonctionnement principaux :

	Application autonome, avec une interface graphique qui vous permet d'interagir avec vos données et de visualiser les résultats de votre analyse en temps réel.
	Bibliothèque Python, vous permettant d'intégrer les fonctions de DataLab (ou les interfaces graphiques) dans vos propres scripts et programmes Python ou des notebooks Jupyter.
	Piloté à distance depuis votre propre logiciel, ou depuis un IDE (par exemple Spyder) ou un notebook Jupyter, en utilisant l'API de DataLab.

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

- Sigima pour le traitement du signal et de l'image (faisant partie de la plateforme DataLab)
- NumPy pour le calcul numérique (tableaux, algèbre linéaire, etc.)
- SciPy pour le calcul scientifique (interpolation, fonctions spéciales, etc.)
- scikit-image et OpenCV pour le traitement d'images
- PyWavelets pour la transformée en ondelettes
- PlotPyStack pour la visualisation interactive de données basée sur Qt

1.1 Installation

Avertissement : Note importante pour les utilisateurs effectuant une mise à niveau depuis DataLab v0.20 ou une version antérieure :

DataLab v1.0 introduit des **changements majeurs** qui ne sont **pas rétrocompatibles** avec v0.20.

- **Les plugins** développés pour v0.20 **doivent être mis à jour** pour fonctionner avec v1.0.
- **Les changements d'API** affectent les intégrations de code personnalisées.

Pour des informations détaillées sur la migration, voir le *guide de migration*.

Cette section fournit des informations sur l'installation de DataLab sur votre système. Une fois installé, vous pouvez démarrer DataLab en exécutant la commande `datalab` 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 *Fonctionnalités de la ligne de commande*.

1.1.1 Modes d'installation

DataLab est disponible sous plusieurs formes :

- En tant que *Paquet Conda*.
- 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.

Paquet Conda

GNU/Linux Windows macOS

Pour installer le paquet `datalab` depuis le canal `conda-forge` (https://anaconda.org/conda-forge/datalab), exécutez la commande suivante :

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

Gestionnaire de paquets `pip`

GNU/Linux Windows macOS

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

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 datalab-platform[qt]
```

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

```
$ pip install datalab-platform
```

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 datalab-platform[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 !

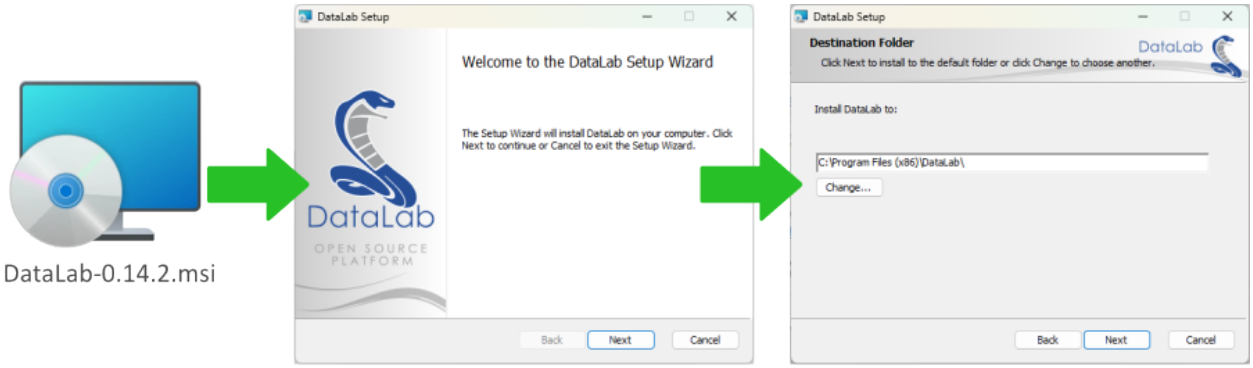


Fig. 1 – Installeur tout-en-un de DataLab pour Windows

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 7 SP1, 8, 10 et 11.

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.

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 datalab-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) :

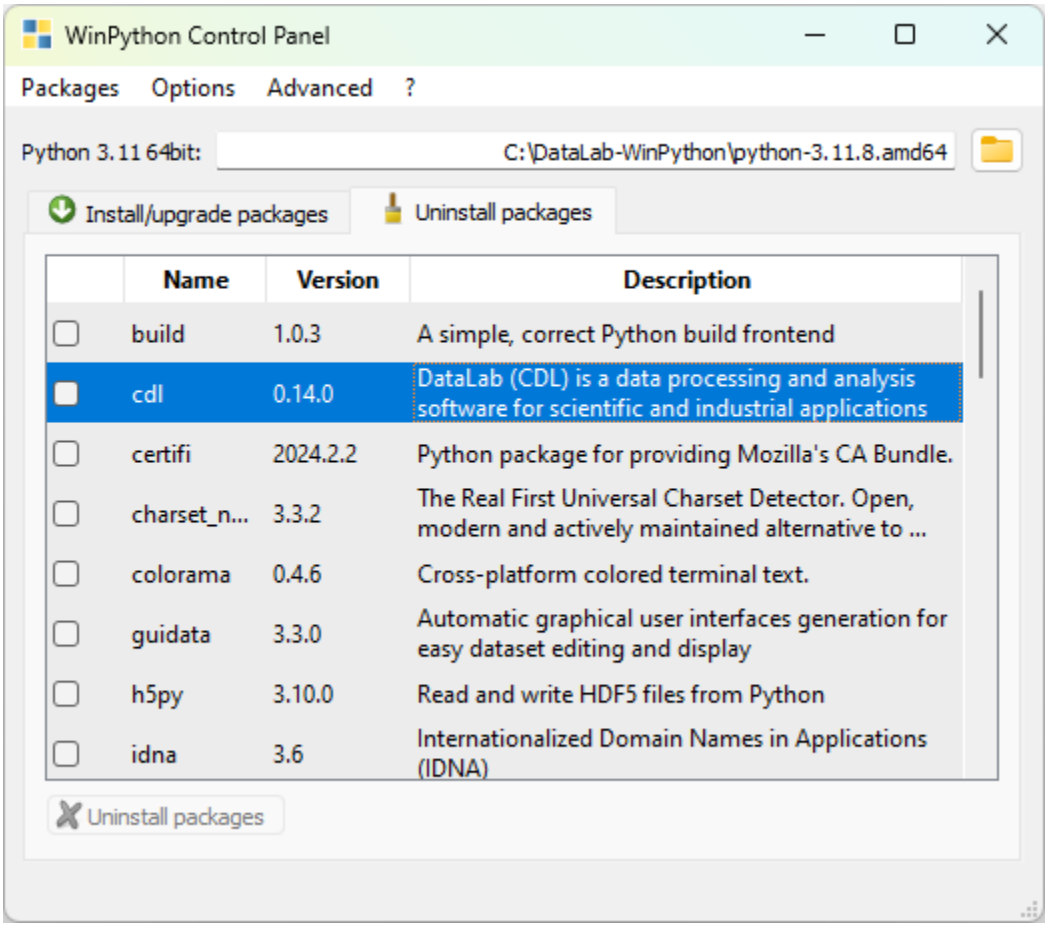


FIG. 3 – Panneau de contrôle DataLab-WinPython

```
$ pip install build wheel  # Install build and wheel packages (if needed)
$ python -m build  # Build the wheel package
$ pip install --upgrade dist/datalab-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 *datalab-platform* nécessite les modules Python suivants :

Nom	Version	Description
Python	>=3.9, <4	Langage de programmation Python
guidata	>= 3.13.3	Génération automatique d'interfaces graphiques pour l'édition et l'affichage de jeux de données
PlotPy	>= 2.8.2	Outils de tracé de courbes et d'images pour les applications Python/Qt
Sigima	>= 1.0.2	Scientific computing engine for 1D signals and 2D images, part of the DataLab open-source platform.
NumPy	>= 1.22	Paquet fondamental pour le calcul sur tableaux en Python
SciPy	>= 1.10.1	Algorithmes fondamentaux pour le calcul scientifique en Python
scikit-image	>= 0.19.2	Traitement d'images en Python
pandas	>= 1.4	Analyse de données, séries temporelles et statistiques
PyWavelets	>= 1.2	PyWavelets, module de transformation en ondelettes
psutil	>= 5.8	Cross-platform lib for process and system monitoring.
packaging	>= 21.3	Core utilities for Python packages

Modules optionnels pour le support de l'interface graphique (Qt) :

Nom	Version	Description
PyQt5	>= 5.15.6	Bibliothèque d'interfaces graphiques Qt pour Python

Modules optionnels pour le développement :

Nom	Version	Description
build		A simple, correct Python build frontend
babel		Internationalization utilities
Coverage		Mesure de la couverture de code pour Python
pylint		Analyseur statique de code Python
ruff		Analyseur de code et formateur Python extrêmement rapide, écrit en Rust.
pre-commit		A framework for managing and maintaining multi-language pre-commit hooks.

Modules optionnels pour la génération de la documentation :

Nom	Version	Description
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

Modules optionnels pour l'exécution de la suite de tests :

Nom	Version	Description
pytest		pytest : tests simples et puissants avec Python
pytest-xvfb		Plugin pytest pour exécuter Xvfb (ou Xephyr/Xvnc) pour les 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 duo gagnant*.
- 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++, etc.). 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 duo gagnant* 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. De plus, la suite de tests comprend des tests de validation basés soit sur des données de référence, soit sur des solutions analytiques

Voir aussi :

Voir la section *Vue d'ensemble et fonctionnalités communes* pour plus d'informations sur la stratégie de validation de DataLab.

1.3 Fonctionnalités principales

Cette page présente brièvement les principales fonctionnalités de DataLab.

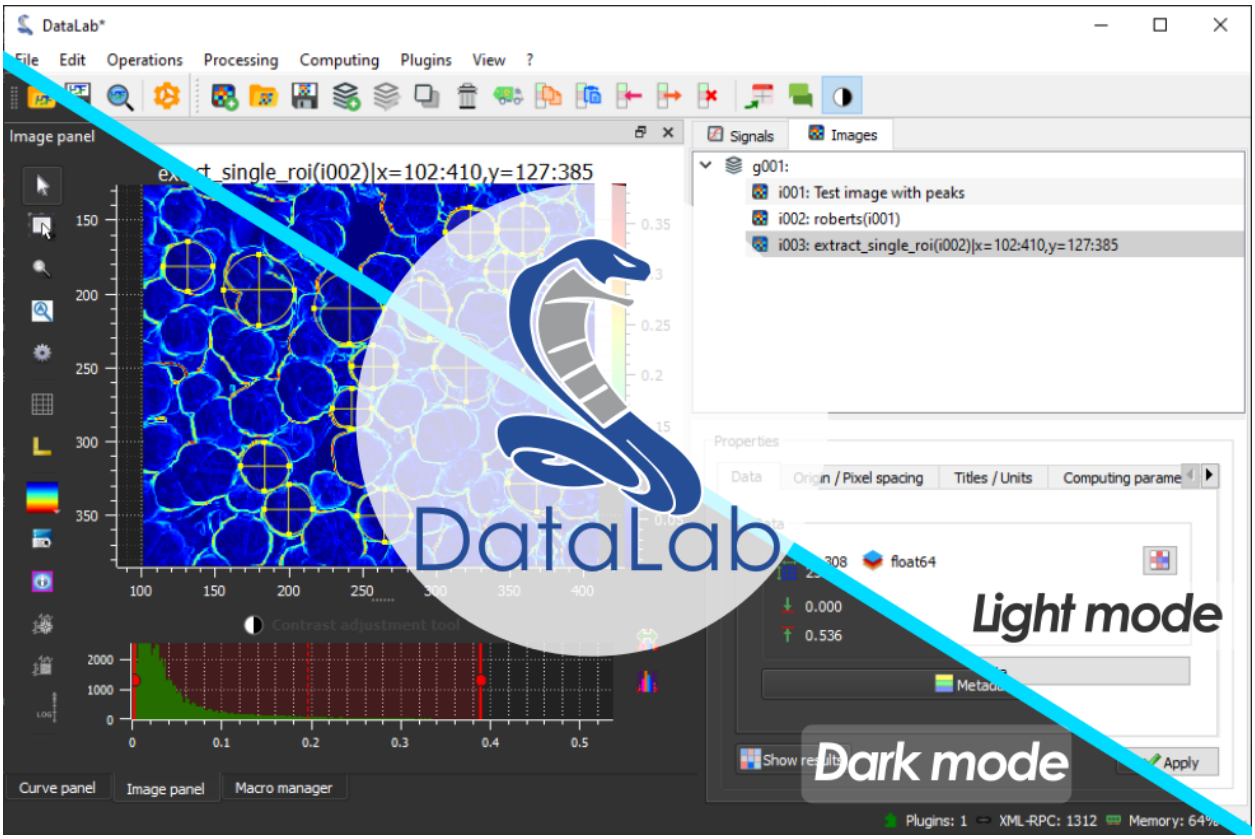


FIG. 4 – DataLab prend en charge le mode sombre et le mode clair en fonction des paramètres de votre plateforme (cela est géré par le paquet guidata et peut être remplacé en définissant la variable d'environnement *QT_COLOR_MODE* sur *dark* ou *light*).

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 (personnalisables)
	✓	Profils d'intensité (ligne, moyen, radial)
✓	✓	Annotations
✓	✓	Persistance des paramètres dans l'espace de travail
	✓	Distribuer les images sur une grille
✓	✓	Vues simples ou superposées

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…
✓	✓	Opérations avec une constante
✓	✓	Racine carrée, puissance, logarithme, exponentielle…
✓	✓	Moyenne, écart-type
✓		Dérivée, intégrale
✓	✓	Extraction de ROI, permutation des axes X/Y
✓		Détection de pics semi-automatique
✓	✓	Convolution, déconvolution
	✓	Correction de champ plat
	✓	Symétrie, rotation, redimensionnement…
	✓	Profils d'intensité (ligne, moyen, radial)
	✓	Binning de pixels
✓	✓	Calibration linéaire
✓	✓	Normalisation, rognage, correction de décalage
✓		Inversion de l'axe X
	✓	Seuillage (manuel, Otsu…)
✓	✓	Filtre gaussien, filtre de Wiener
✓	✓	Moyenne mobile, médiane mobile
✓	✓	FFT, FFT inverse, spectre de puissance/phase/magnitude, densité spectrale de puissance
✓		Interpolation, rééchantillonnage
✓		Elimination de tendance
✓		Mode XY
✓		Ajustement interactif : Gauss, Lorentz, Voigt, polynôme, CDF…
✓		Ajustement interactif multi-gaussien
✓		Filtres fréquentiels (passe-bas, passe-haut, passe-bande, coupe-bande)
✓		Fenêtrage (Hann, Hamming…)
	✓	Filtre de Butterworth

suite sur la page suivante

Tableau 1 – suite de la page précédente

Signal	Image	Fonctionnalité
	✓	Correction d'exposition (gamma, log…)
	✓	Restauration (variation totale, bilatérale…)
	✓	Morphologie (érosion, dilatation…)
	✓	Détection de contours (Roberts, Sobel…)
✓	✓	Analyse sur ROI personnalisée
✓		FWHM, FW @ $1/e^2$
✓		Paramètres dynamiques (ENOB, SNR…), Période/Taux d'échantillonnage
	✓	Barycentre (méthode robuste par rapport au bruit)
	✓	Centre du cercle d'encadrement minimal
	✓	Détection de pics 2D
	✓	Détection de contours
	✓	Transformée de Hough circulaire
	✓	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

0.jpg

Avertissement : Cette vidéo montre une version plus ancienne de DataLab. Certaines fonctionnalités ont peut-être changé depuis. Veuillez vous référer à la documentation pour les informations les plus à jour.

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.

Note : 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

01.jpg

Avertissement : Cette vidéo montre une version plus ancienne de DataLab. Certaines fonctionnalités ont peut-être changé depuis. Veuillez vous référer à la documentation pour les informations les plus à jour.

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

Les menus de DataLab changent en fonction du contexte. Comme nous allons travailler avec un spectre, un signal 1D, nous devons sélectionner le panneau Signal avant de continuer.

Dans un flux de travail typique, nous ouvririons DataLab et lirions le spectre à partir d'un fichier en utilisant « Fichier > Ouvrir… », le bouton dans la barre d'outils, ou en faisant glisser-déposer le fichier dans le panneau de droite. Cependant, pour ce tutoriel, nous utiliserons la fonction « Plugins > Données de test > Charger le spectre de paracétamol » pour générer un spectre de test, ce qui est pratique à des fins de démonstration.

Une fois ouvert, le spectre est affiché dans la fenêtre principale. C'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.

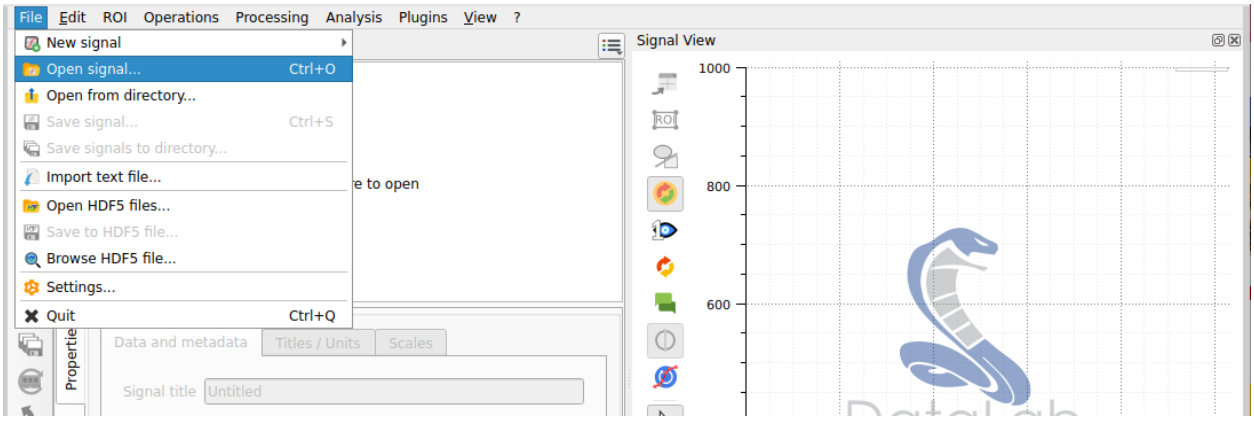


FIG. 5 – Le menu « Fichier > Ouvrir… » pour ouvrir un fichier de spectre.

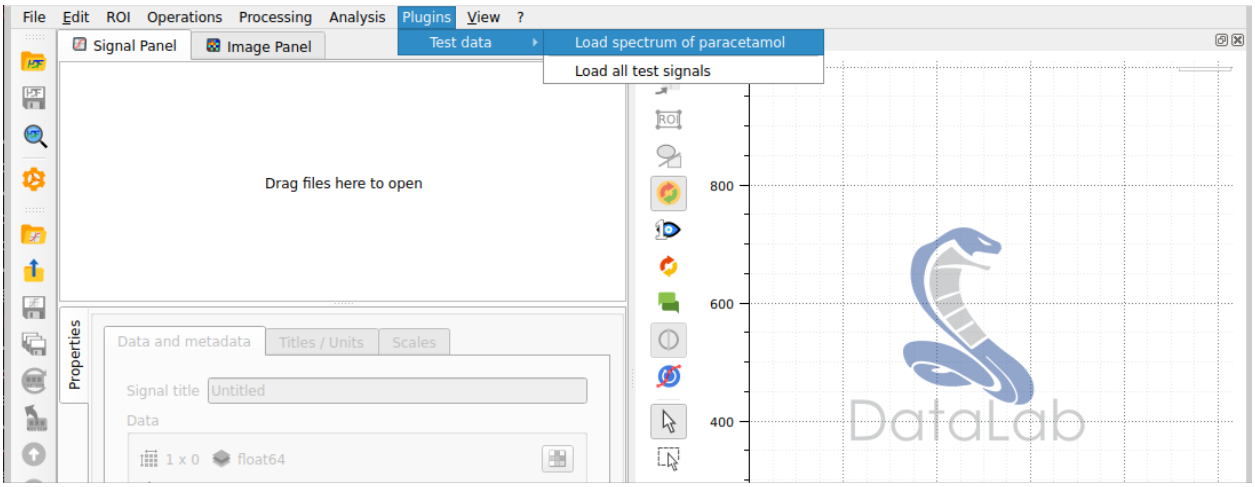


FIG. 6 – Le plugin « Plugins > Données de test > Charger le spectre de paracétamol » pour générer le spectre de test pour ce tutoriel.

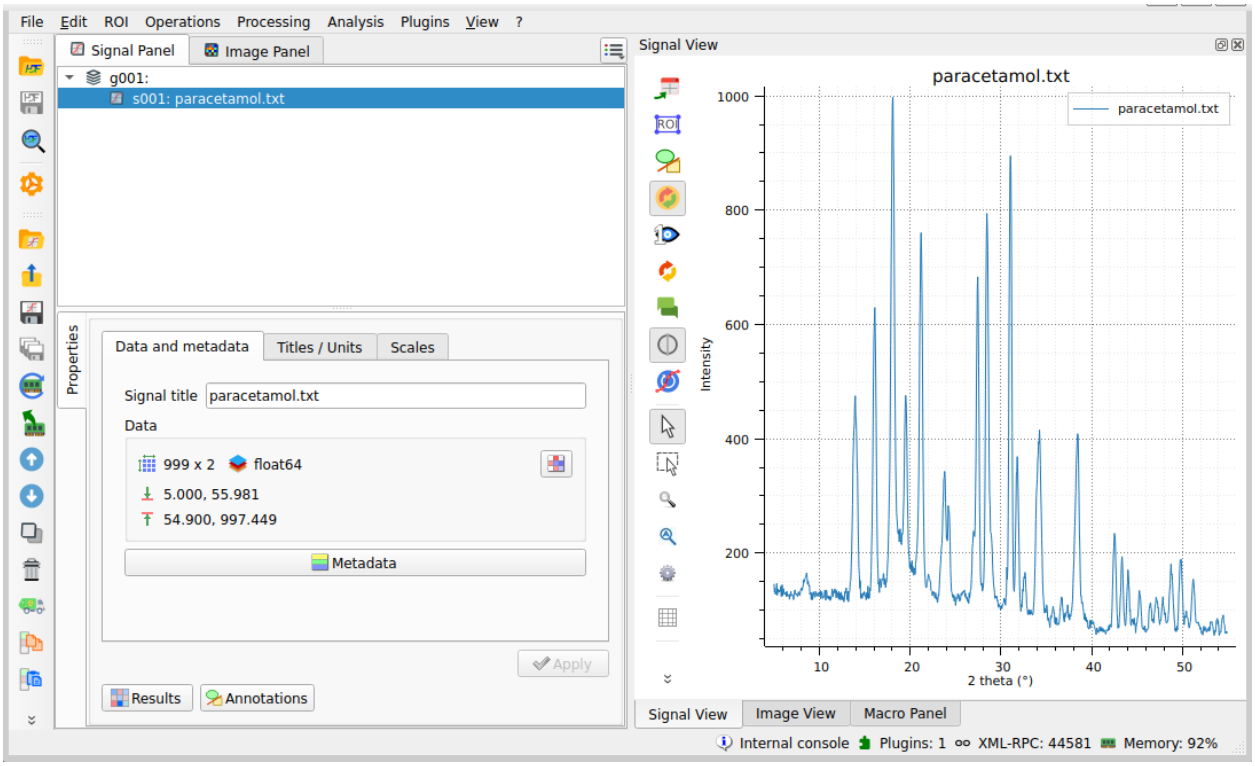


FIG. 7 – Le spectre affiché dans le panneau « Vue Signal ».

Le signal est assez propre. Cependant, pour démontrer les capacités de filtrage de DataLab, nous allons appliquer un filtre de Wiener pour réduire tout bruit résiduel tout en préservant les caractéristiques spectrales. Ceci est disponible sous « Traitement > Réduction du bruit > Filtre de Wiener ».

Concentrons notre analyse sur l'un des pics d'intérêt. Pour ce faire, nous définissons une région d'intérêt (ROI) autour de la caractéristique que nous voulons analyser et utilisons le menu « ROI > Extraire… » pour l'extraire. La boîte de dialogue « Régions d'intérêt » sera affichée. Sélectionnez une zone et cliquez sur « OK ». Une fenêtre de confirmation apparaîtra—cliquez sur « Oui » pour extraire la région d'intérêt. Un nouveau signal contenant la ROI sera créé et affiché dans la fenêtre principale.

Élimination de tendance linéaire

Après avoir ajusté le pic principal, nous pouvons vouloir éliminer toute dérive de ligne de base présente dans l'ensemble du spectre. La fonction d'élimination de tendance dans DataLab effectue un ajustement linéaire sur l'ensemble du signal, y compris les pics. Dans notre signal, les pics occupent une grande partie des données, ce qui est acceptable pour les signaux où les pics sont distribués symétriquement autour du centre avec des amplitudes similaires. Cependant, ce n'est pas le cas ici, nous ne pouvons donc pas nous attendre à ce que cette fonction fonctionne bien. Néanmoins, cet exemple illustre comment les fonctions DataLab peuvent être combinées pour effectuer des analyses plus avancées.

Pour visualiser la limitation mentionnée ci-dessus, nous appliquerons la fonction d'élimination de tendance directement au signal filtré. Il est important de se rappeler que nous avons précédemment défini une ROI sur le signal pour nous concentrer sur le pic principal. Nous devons supprimer cette contrainte de ROI pour effectuer l'élimination de tendance sur le signal complet. Pour exécuter l'élimination de tendance, nous utilisons « Traitement > Élimination de tendance », choisissons la méthode d'élimination de tendance linéaire et cliquons sur « OK ». Le résultat sera affiché dans la fenêtre principale.

La comparaison des signaux filtrés et sans tendance montre, comme prévu, que la fonction d'élimination de tendance ne

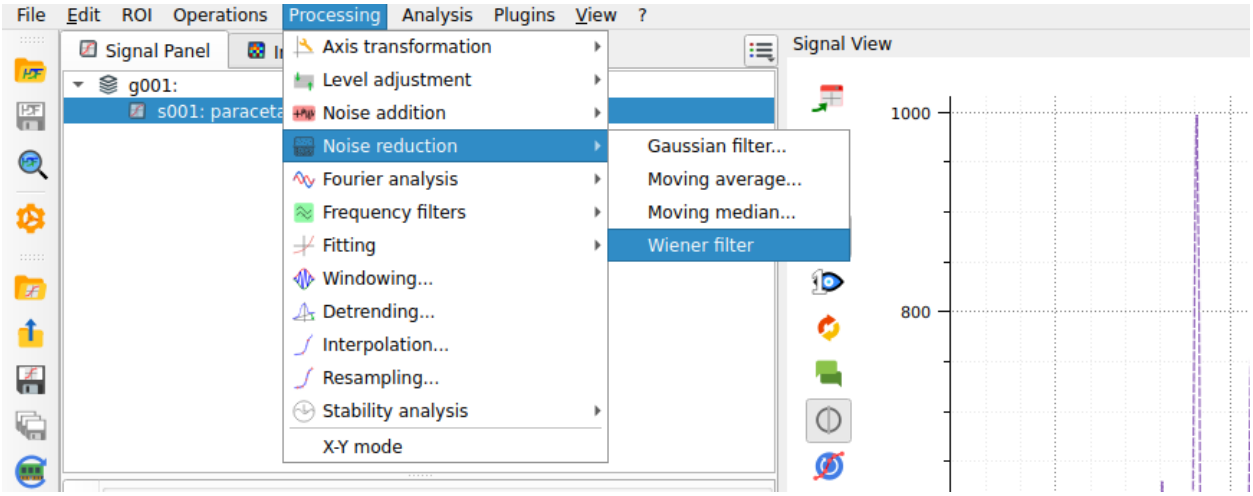


FIG. 8 – L'option de menu « Traitement > Réduction du bruit > Filtre de Wiener ».

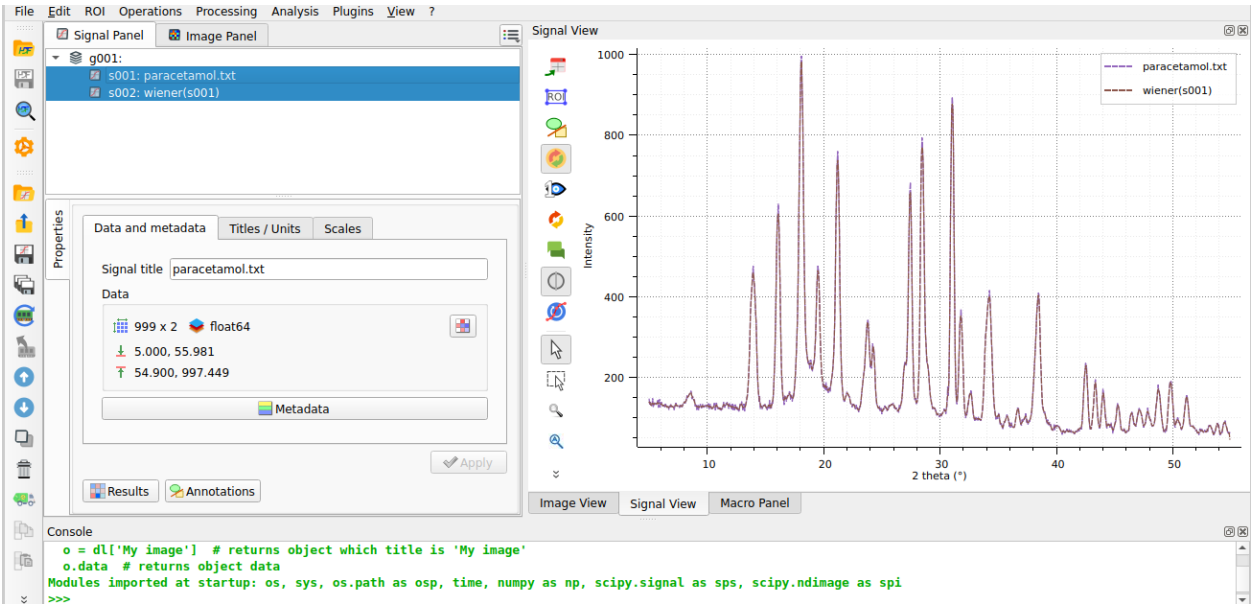


FIG. 9 – Le résultat affiché dans la fenêtre principale.

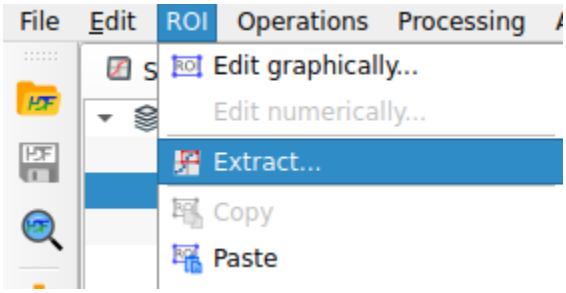


FIG. 10 – Le menu « Opérations > Extraction de ROI ».

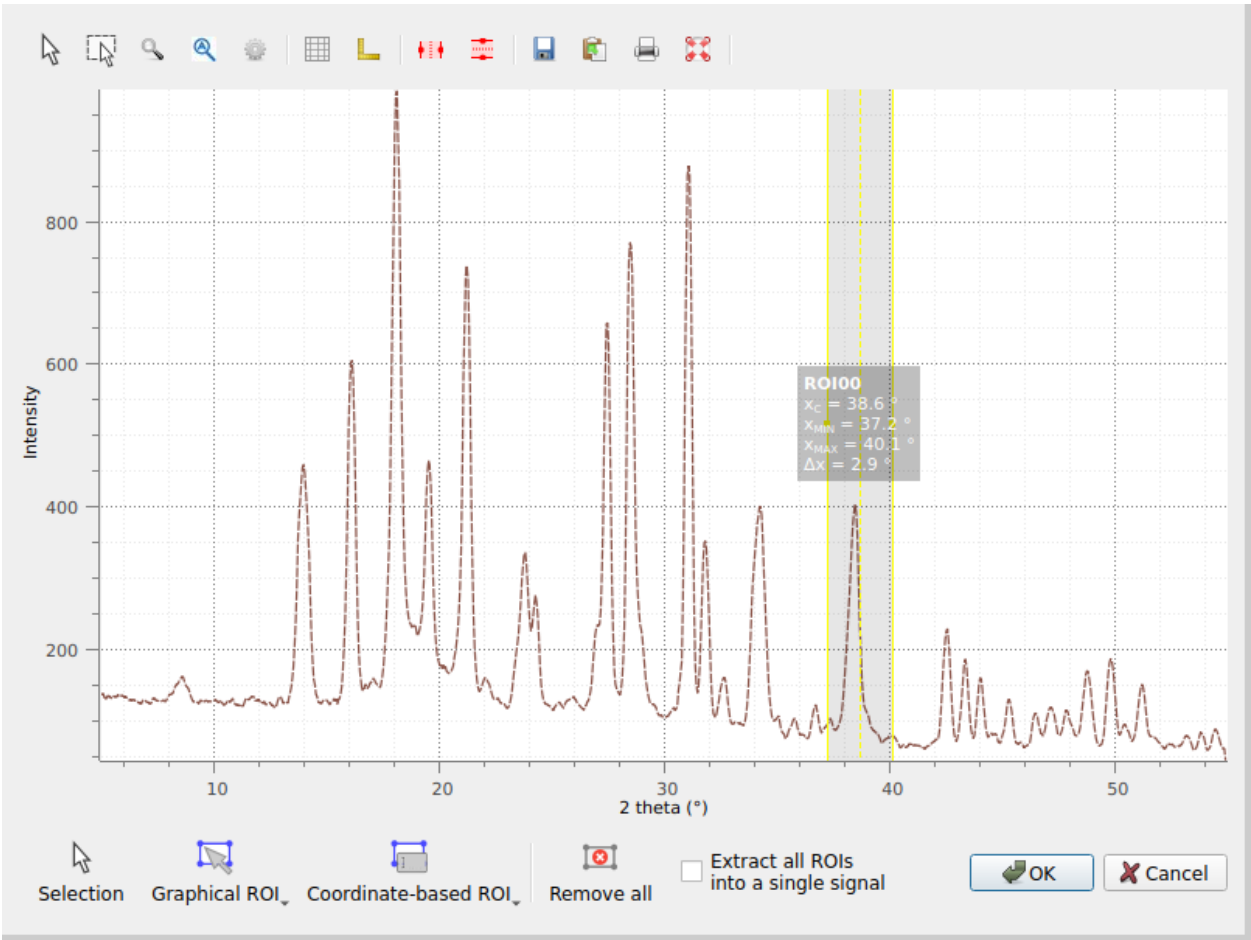


FIG. 11 – La boîte de dialogue « Régions d'intérêt » affichée.

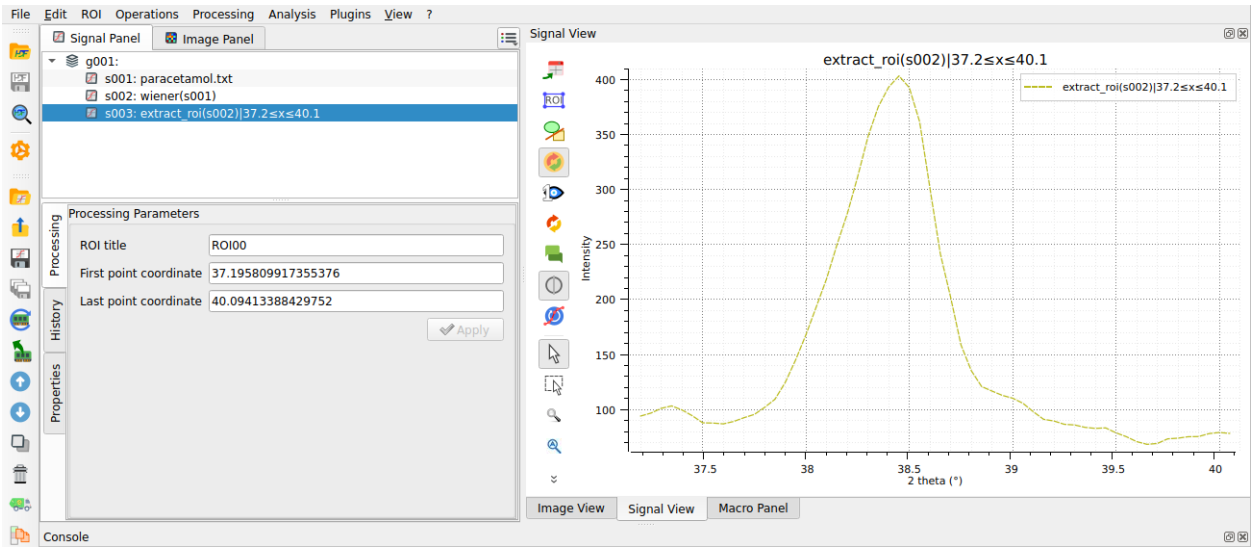


FIG. 12 – La région d'intérêt affichée dans la fenêtre principale.

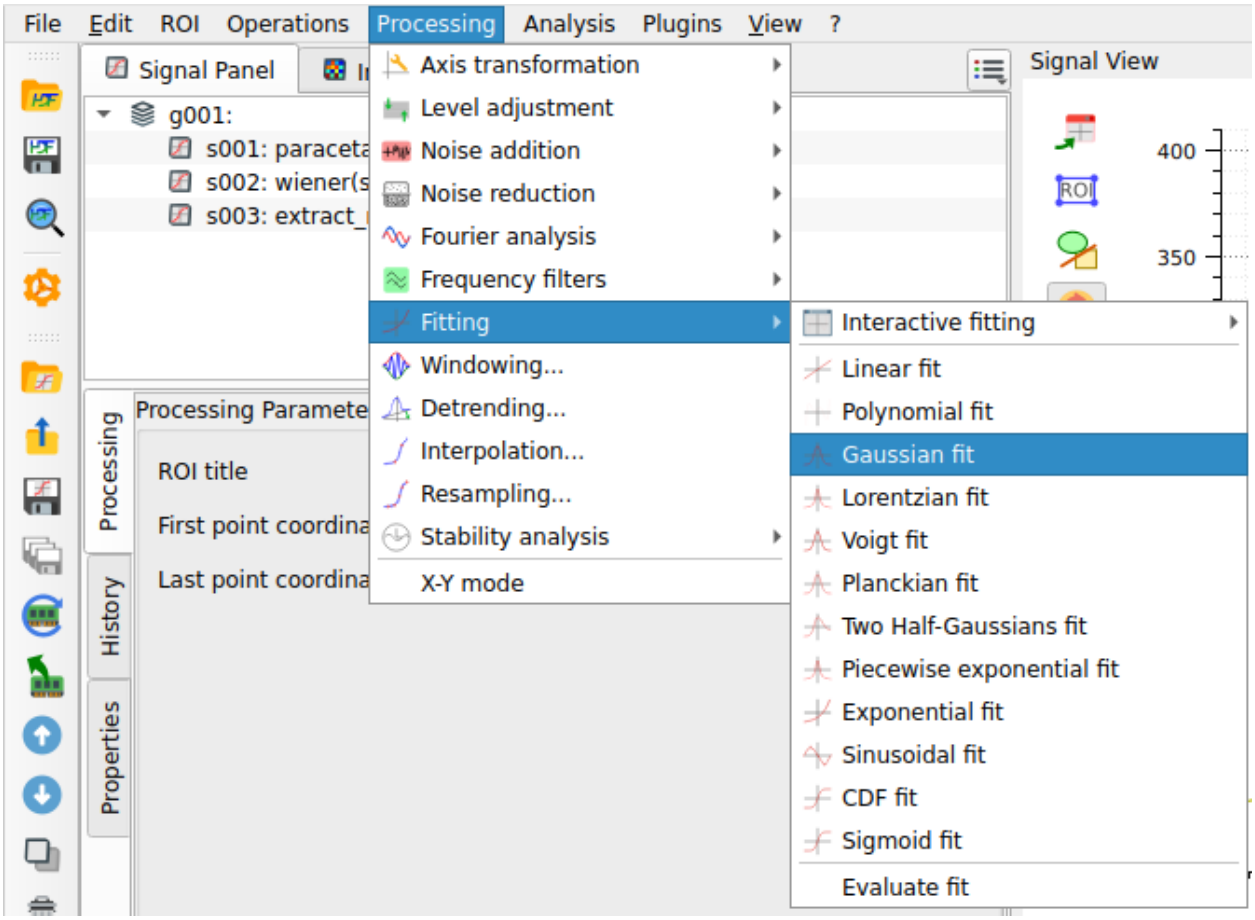


FIG. 13 – Ouvrir la fenêtre d'ajustement du modèle avec « Traitement > Ajustement > Ajustement gaussien ».

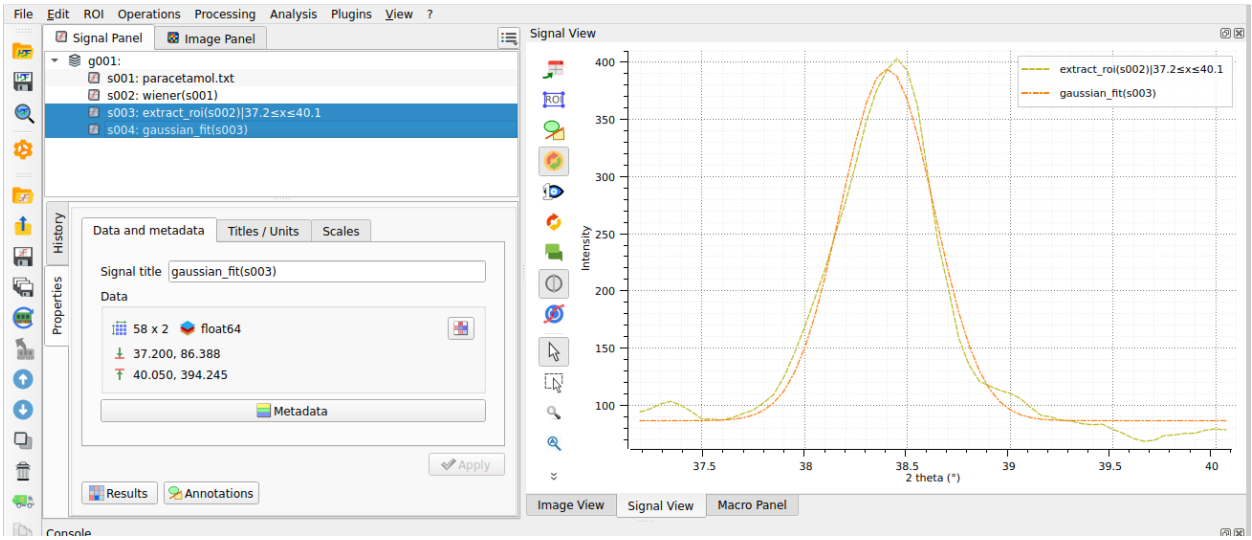


FIG. 14 – Le résultat de l'ajustement affiché dans la fenêtre principale.

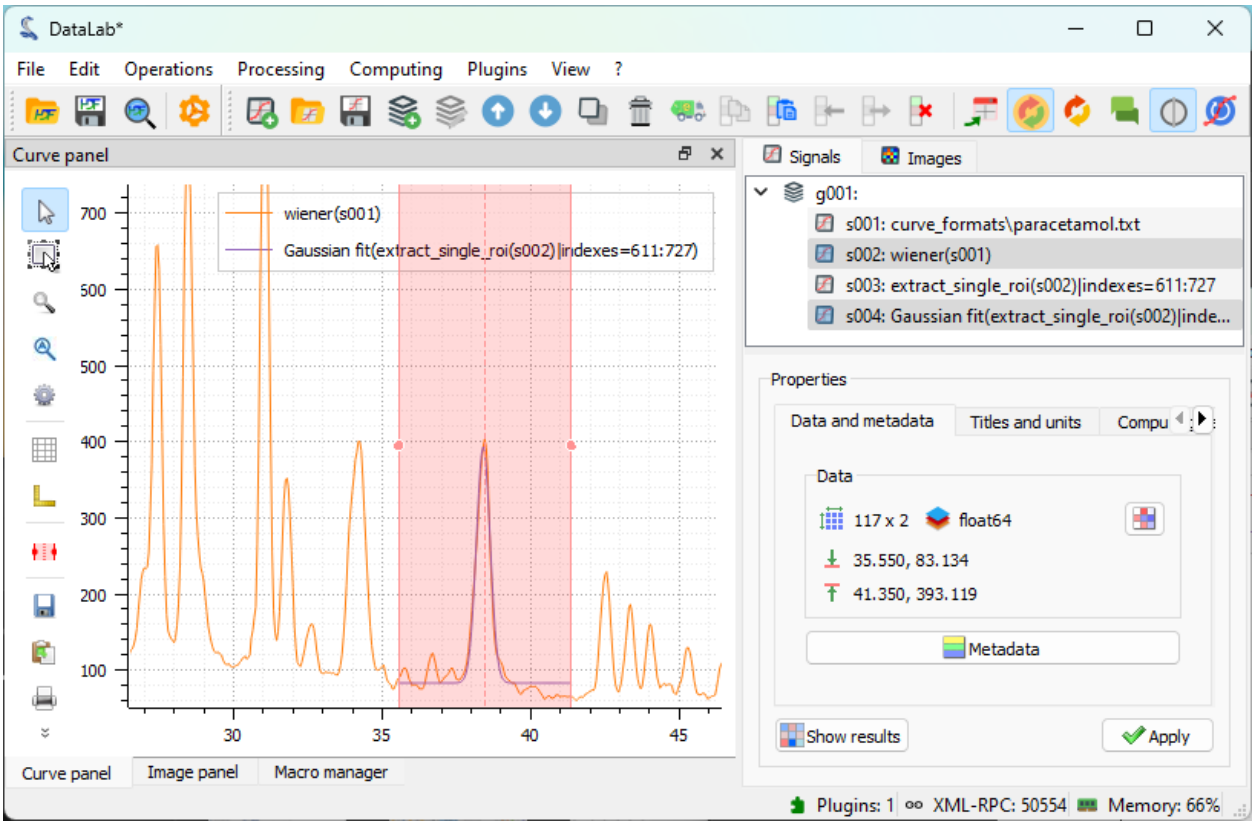


Fig. 15 – Le spectre complet et l'ajustement sont tous deux sélectionnés dans le panneau « Signaux », de sorte que les deux sont affichés dans le panneau de visualisation à gauche. Cela permet une comparaison visuelle facile si nécessaire pour l'analyse.

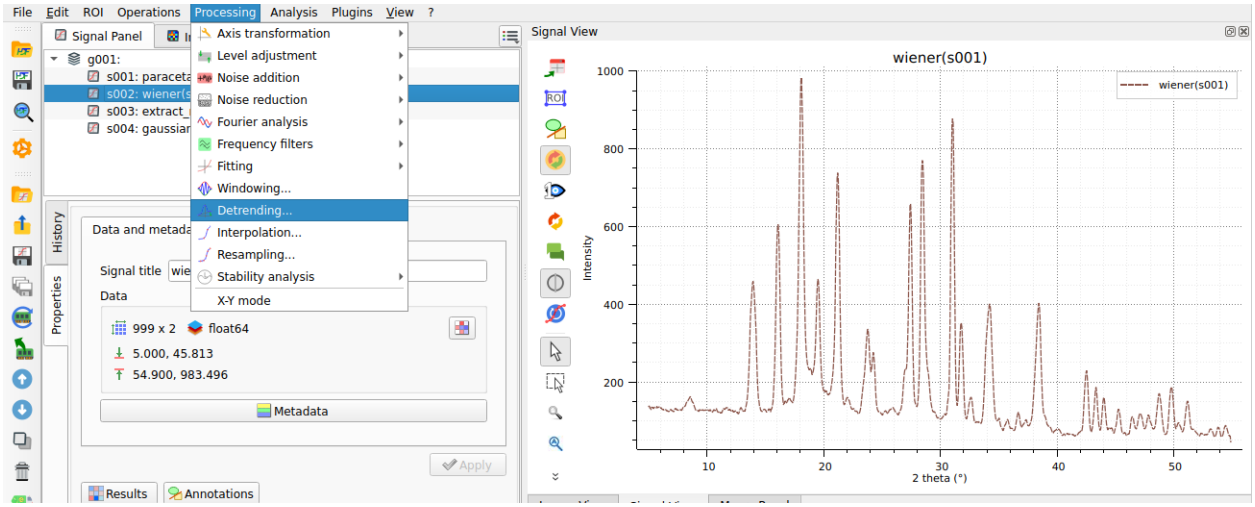


Fig. 16 – La fonction « Traitement > Élimination de tendance ».

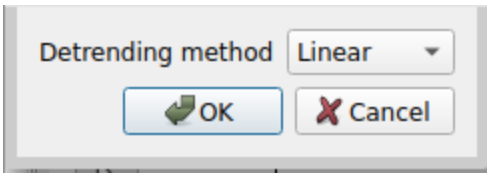


FIG. 17 – Nous choisissons une méthode d'élimination de tendance linéaire, et nous cliquons sur « OK ».

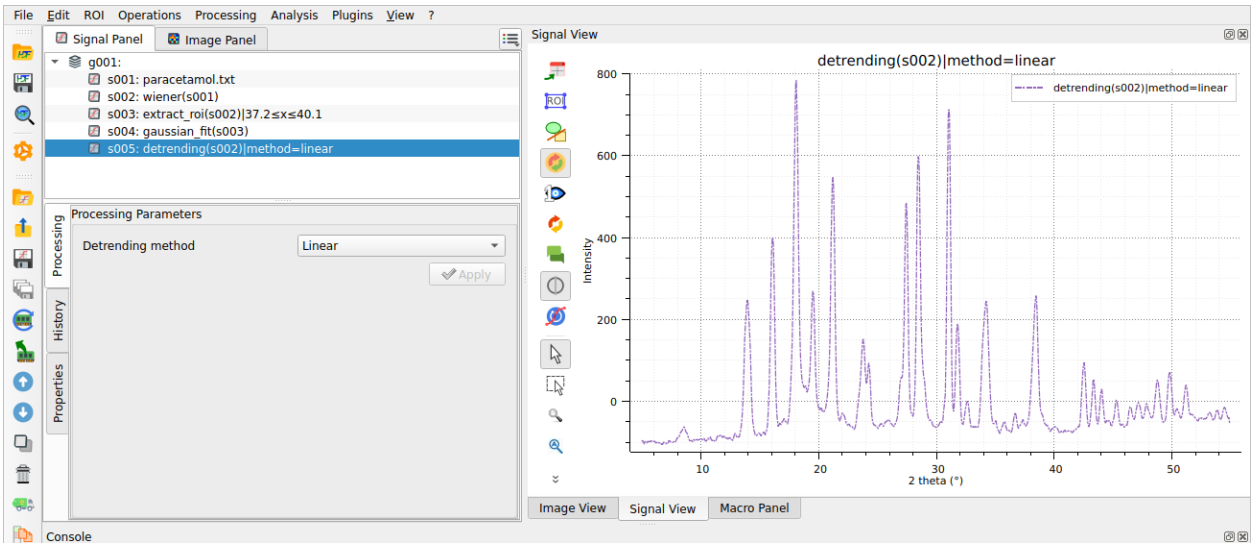


FIG. 18 – Le résultat de l'élimination de tendance affiché dans la fenêtre principale.

fonctionne pas bien sur ce signal. Comme expliqué précédemment, c'est parce que l'algorithme effectue un ajustement linéaire sur l'ensemble du signal, y compris les pics. L'effet est clairement visible dans le tracé : les pics les plus élevés à gauche commencent à une valeur d'intensité inférieure à ceux de droite après l'élimination de tendance, et tous les pics ont une ligne de base en dessous de zéro.

Élimination de tendance améliorée avec exclusion des pics

Pour surmonter la limitation de la fonction d'élimination de tendance, nous pouvons nous inspirer du comportement du signal sans tendance. Nous avons déjà identifié le problème : l'ajustement linéaire inclut à la fois la ligne de base et les pics.

Pour une meilleure élimination de tendance, nous pouvons d'abord exclure les pics puis effectuer un ajustement linéaire uniquement sur les régions sans pics. Nous pouvons raisonnablement nous attendre à ce que cette approche fournisse une estimation de ligne de base plus précise et un signal sans tendance de meilleure qualité.

Ceci est illustré dans les étapes suivantes :

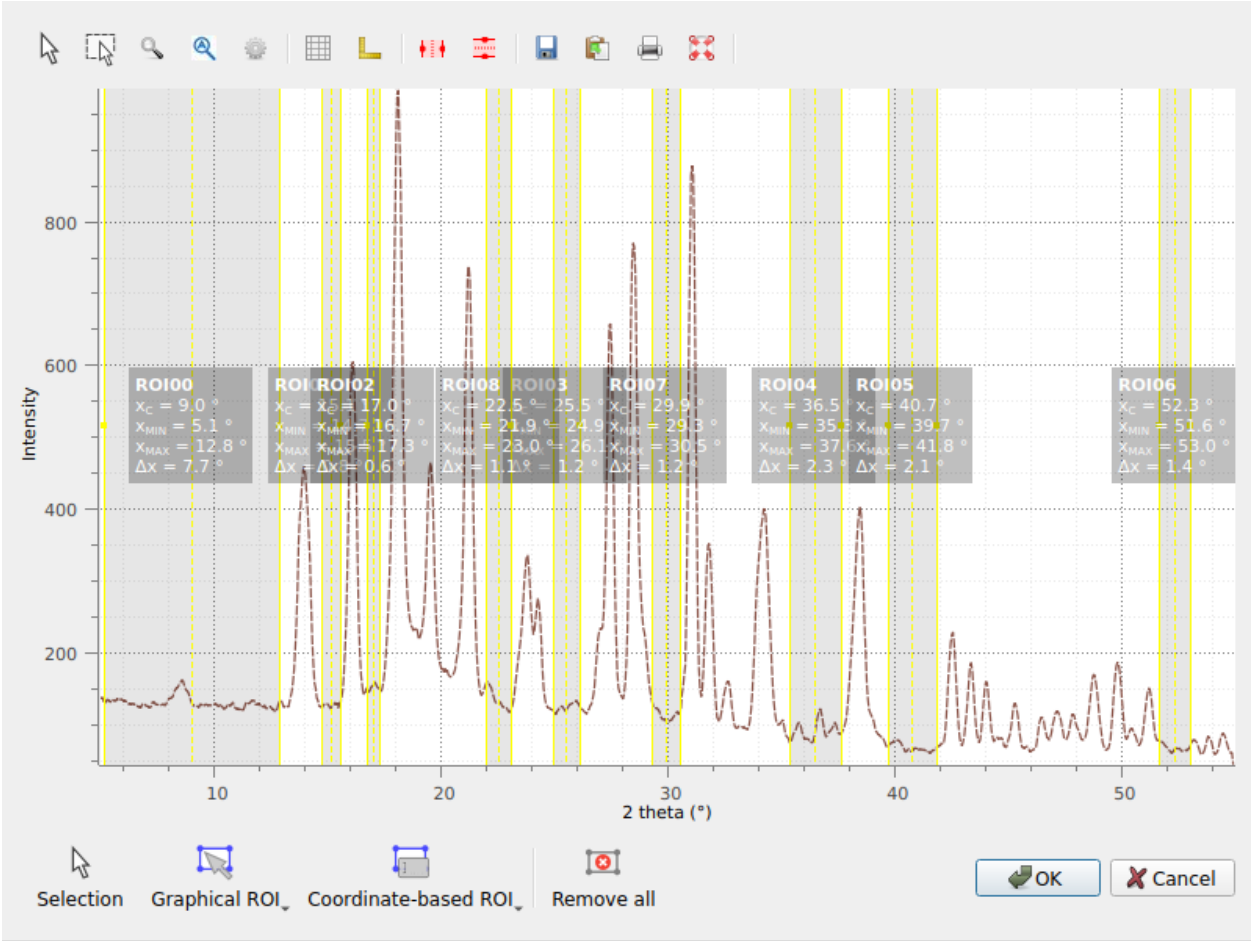


FIG. 19 – Nous sélectionnons les régions correspondant aux régions sans pics.

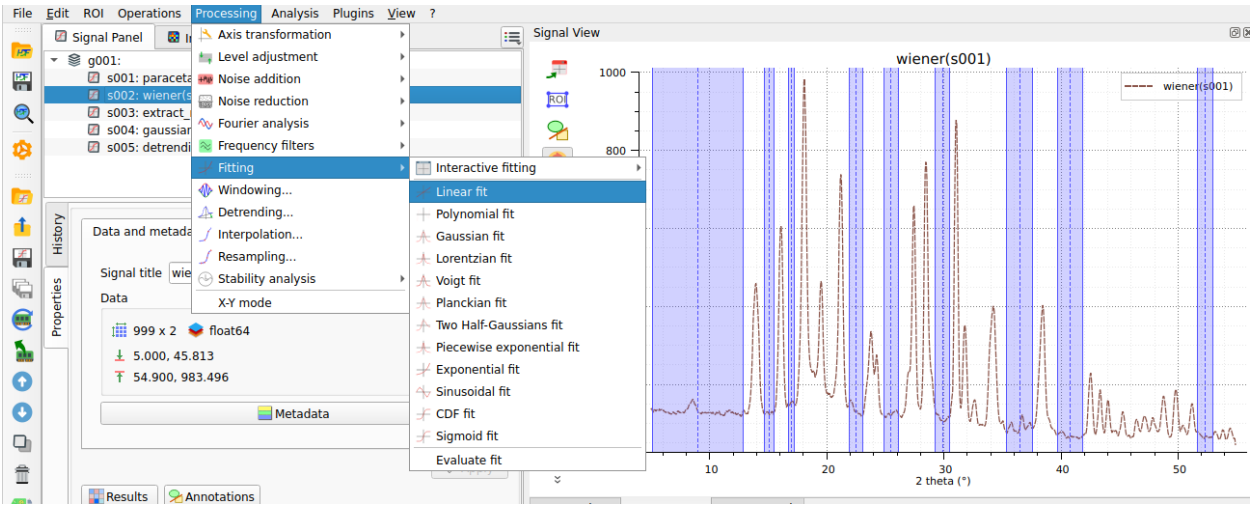


FIG. 20 – Un ajustement linéaire est effectué uniquement sur les régions sélectionnées.

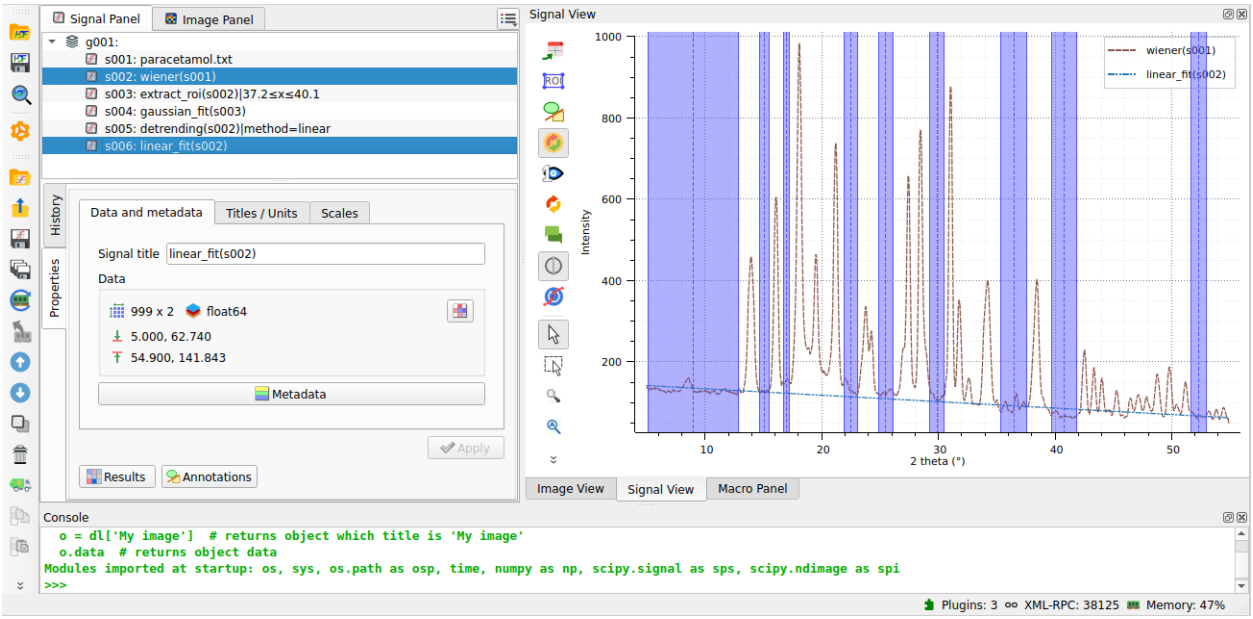


FIG. 21 – La ligne de base linéaire obtenue à partir de l'ajustement est affichée.

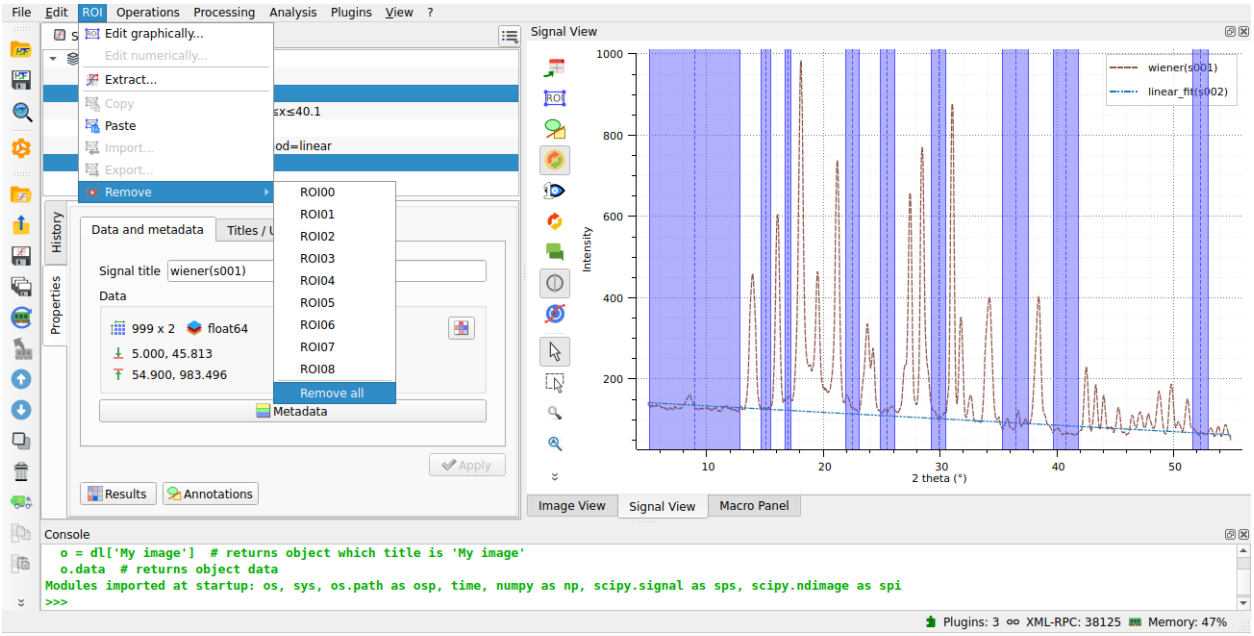


FIG. 22 – Nous supprimons les ROIs pour appliquer l'élimination de tendance sur l'ensemble du signal.

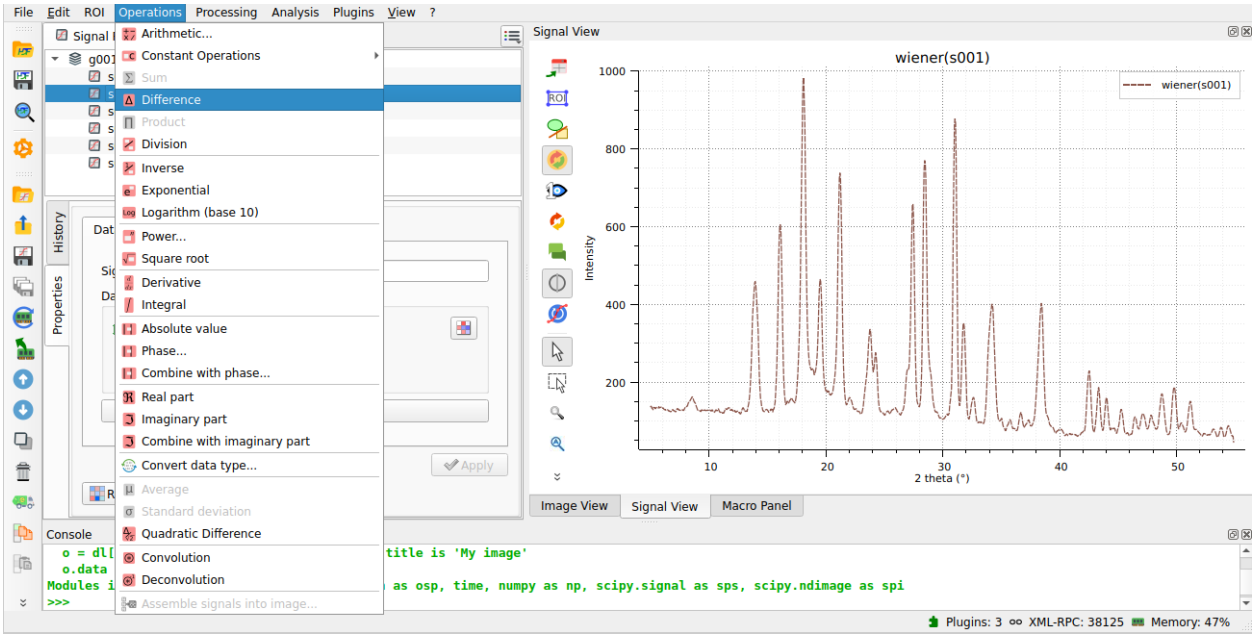


Fig. 23 – Nous utilisons l'opération « différence » pour soustraire la ligne de base du signal d'origine.

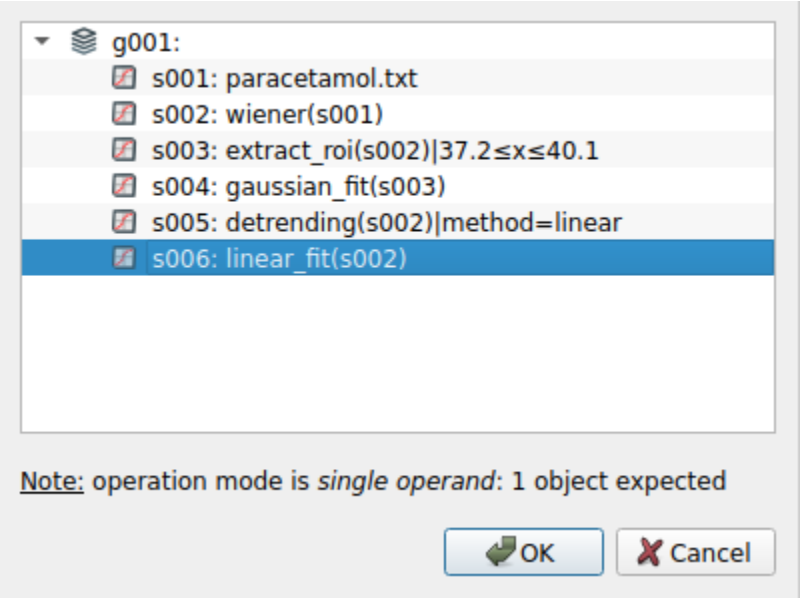


Fig. 24 – Nous sélectionnons la ligne de base linéaire pour effectuer la soustraction.

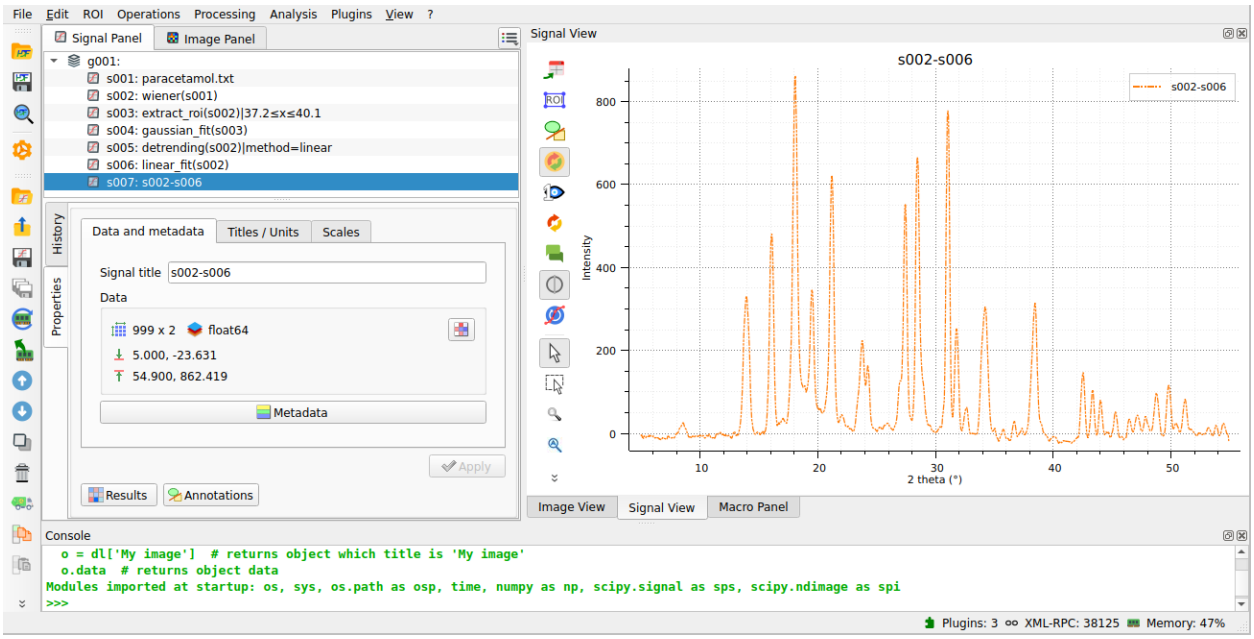


FIG. 25 – Le signal sans tendance résultant est affiché, maintenant avec une ligne de base correcte.

Détection automatique des pics

Nous pouvons utiliser la fonction « Ajustement multi-gaussien » de DataLab pour identifier et ajuster automatiquement plusieurs pics dans le spectre en sélectionnant « Traitement > Ajustement > Ajustement multi-gaussien » dans le menu.

Alternativement, nous pourrions utiliser la fonction « Détection de pics » du menu « Analyse » pour détecter les pics dans le spectre. Il s'agit de la première étape de la fonction « Ajustement multi-gaussien » et peut être utilisée indépendamment pour détecter les pics sans effectuer d'ajustement, créant un signal avec une fonction delta à chaque position de pic détectée.

Sauvegarde de l'espace de travail

Enfin, nous pouvons sauvegarder l'espace de travail dans un fichier. L'espace de travail contient tous les signaux chargés ou créés dans DataLab, ainsi que les résultats de traitement et les paramètres de visualisation (tels que les couleurs de courbe).

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.

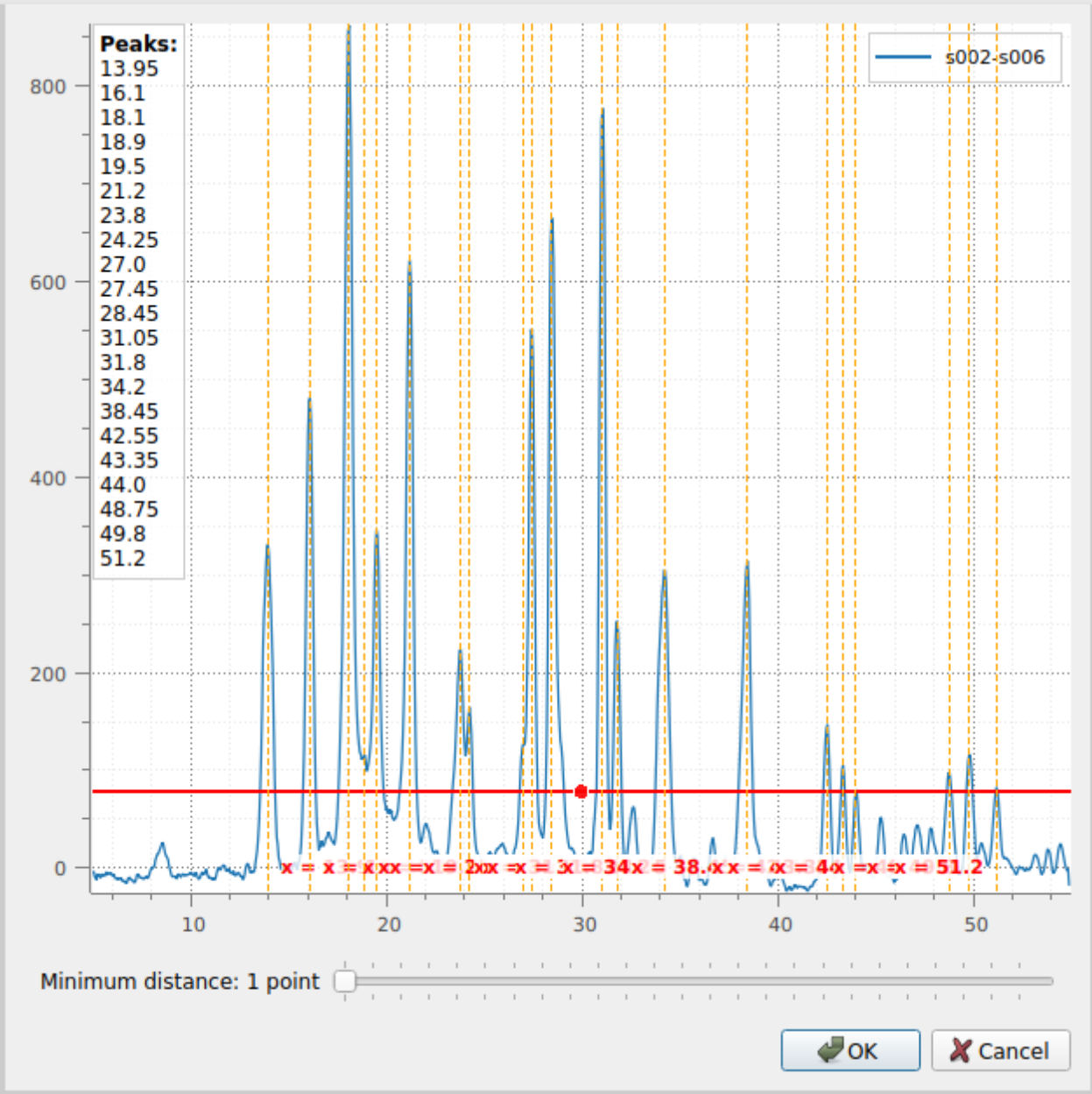


Fig. 26 – Tout d'abord, la boîte de dialogue « Détection de pics de signal » apparaît. Vous pouvez ajuster la position du curseur vertical pour définir le seuil de détection des pics et spécifier la distance minimale entre les pics. Ensuite, cliquez sur « OK ».

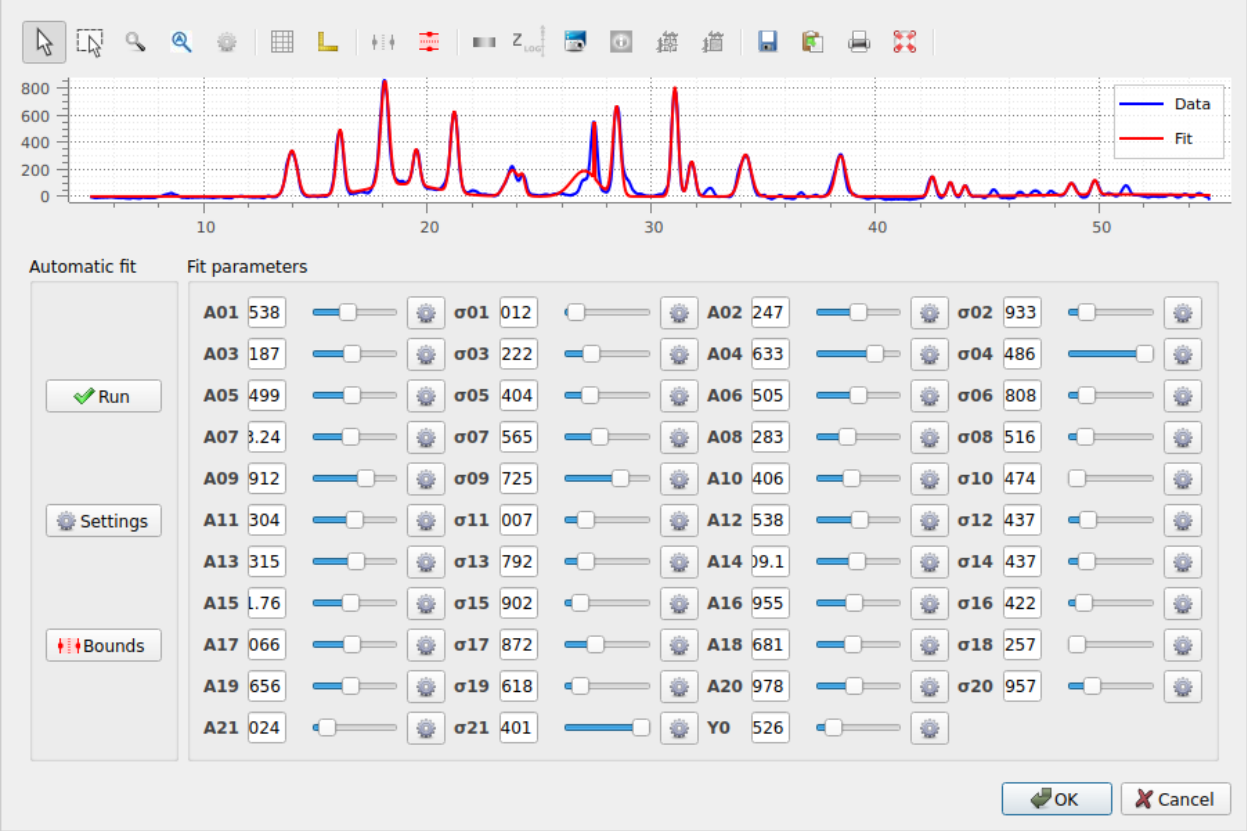


Fig. 27 – La boîte de dialogue « Ajustement multi-gaussien » apparaît. Un ajustement automatique est effectué par défaut. Cliquez sur « OK » (ou ajustez éventuellement les paramètres manuellement à l'aide des curseurs, ou modifiez les paramètres d'ajustement automatique).

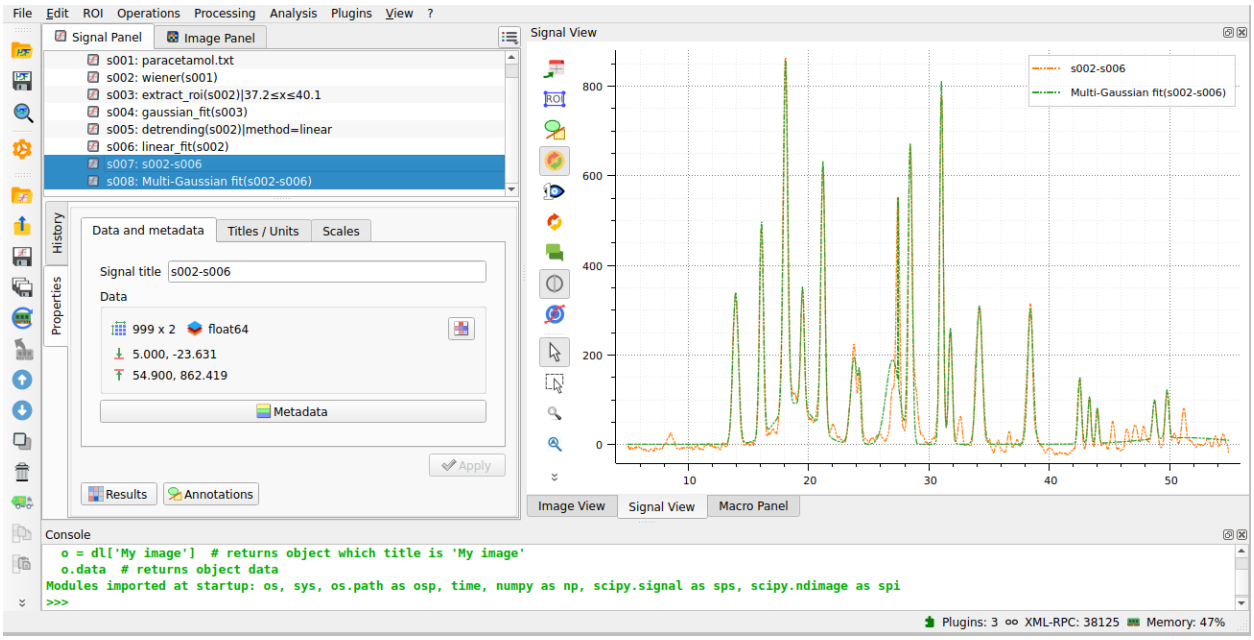


FIG. 28 – Le résultat de l'ajustement est affiché dans la fenêtre principale. Ici, nous avons sélectionné à la fois le spectre et l'ajustement dans le panneau « Signaux » à droite, donc les deux sont affichés dans le panneau de visualisation à gauche.

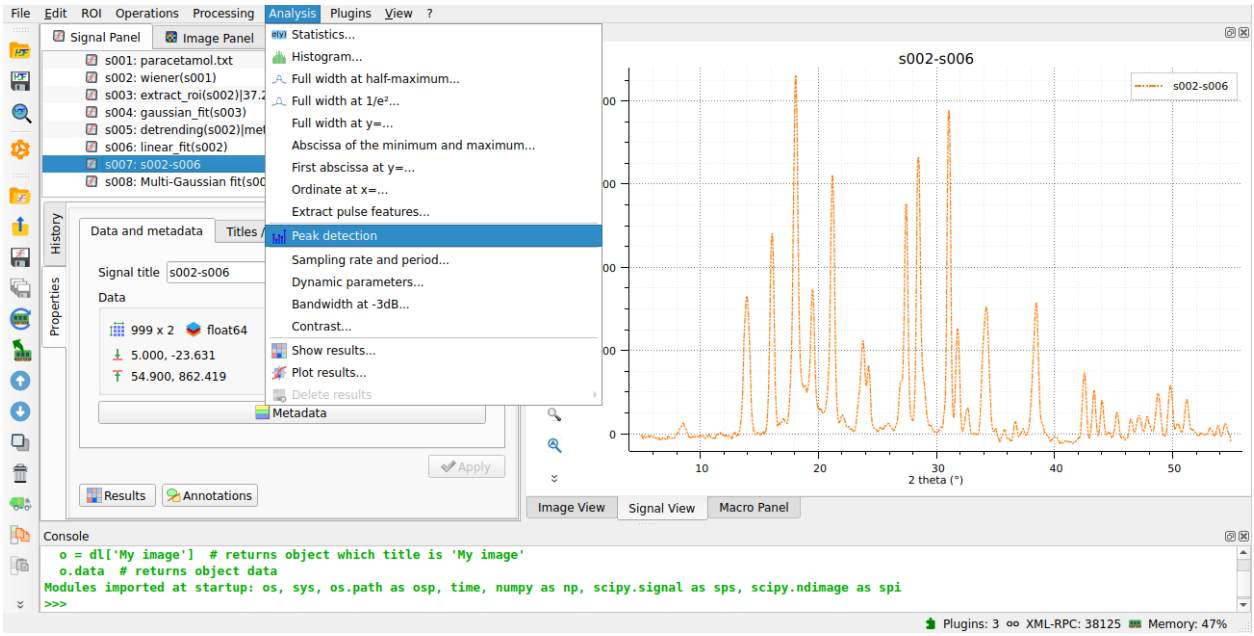


FIG. 29 – Ouvrir la fenêtre « Détection de pics » avec « Analyse > Détection de pics ».

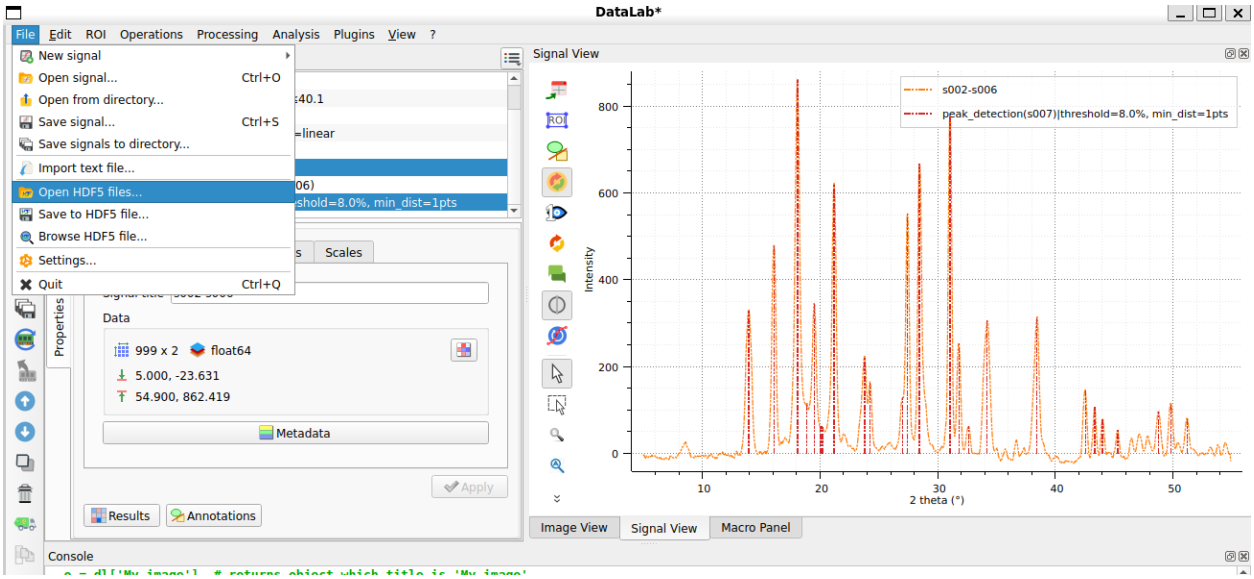


Fig. 30 – Après avoir ajusté les paramètres de détection de pics (en utilisant la même boîte de dialogue que pour l'ajustement multi-gaussien), cliquez sur « OK ». Ensuite, sélectionnez à la fois le résultat « détection_de_pics » et le spectre d'origine dans le panneau « Signaux » pour les afficher ensemble dans le panneau de visualisation à gauche.

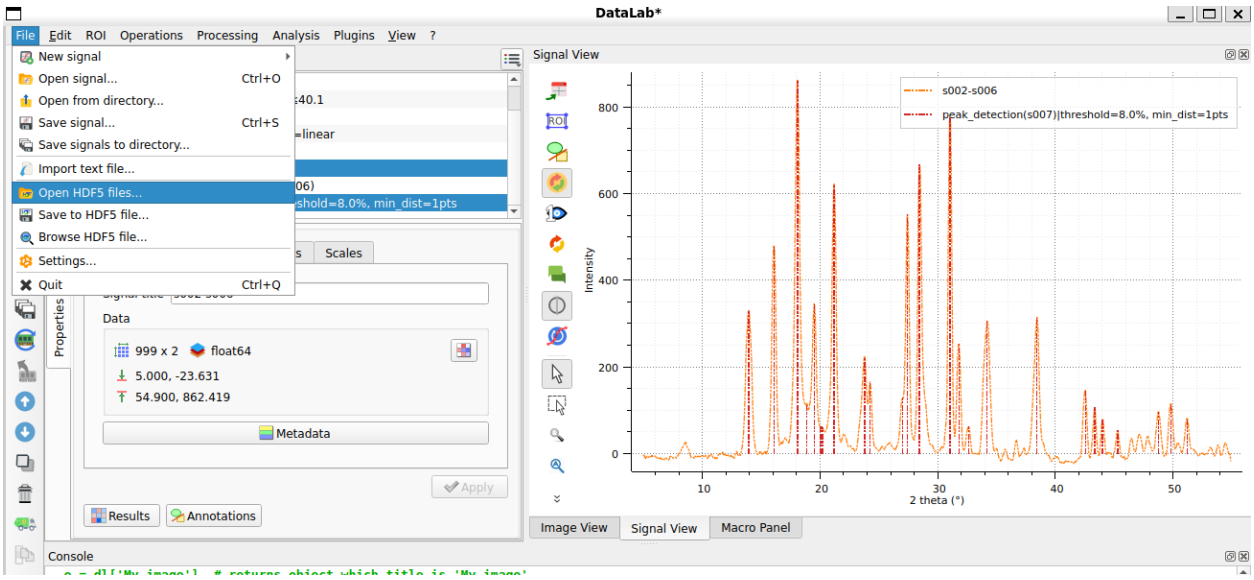


Fig. 31 – Sauvegarder l'espace de travail dans un fichier avec « Fichier > Sauvegarder dans un fichier HDF5… », ou le bouton dans la barre d'outils.

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).

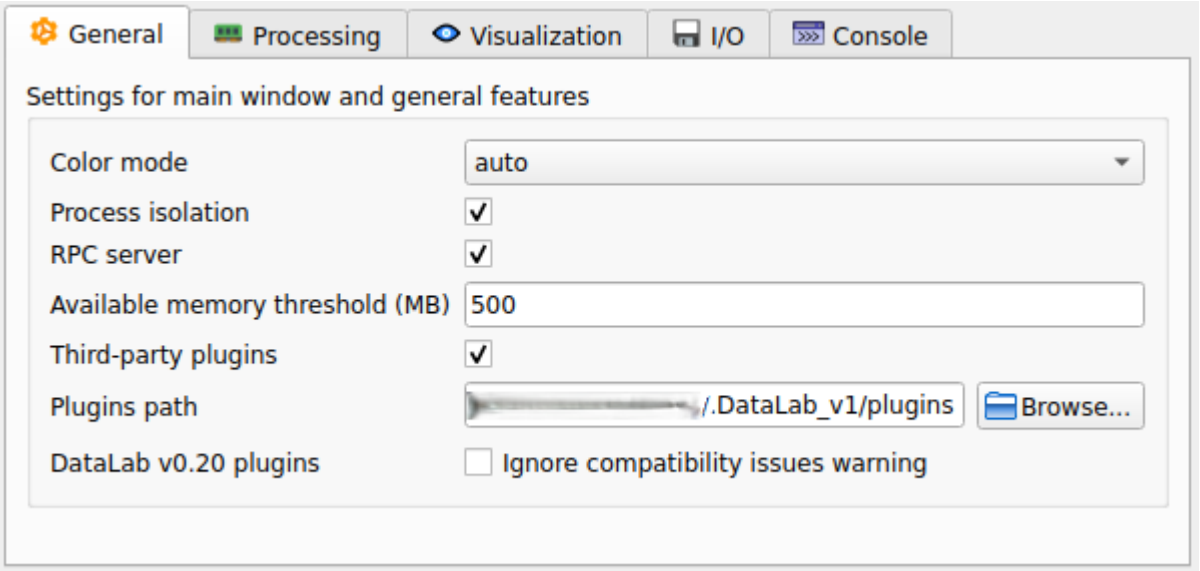


Fig. 32 – Dans l'onglet « Général », nous pouvons voir le champ « Chemin des plugins ». C'est le chemin où DataLab recherchera les plugins. Nous pouvons ajouter un nouveau plugin en copiant/collant le fichier du plugin dans ce répertoire.

Voir aussi :

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

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

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) ».

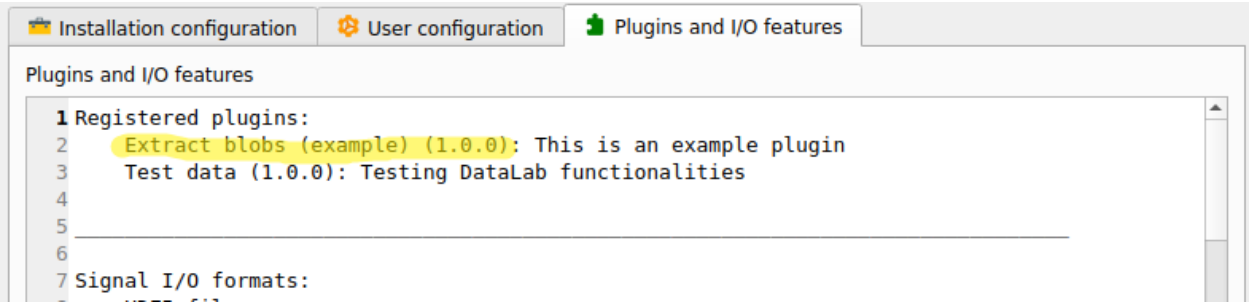


Fig. 33 – La boîte de dialogue « À propos de DataLab » affiche la liste des plugins disponibles.

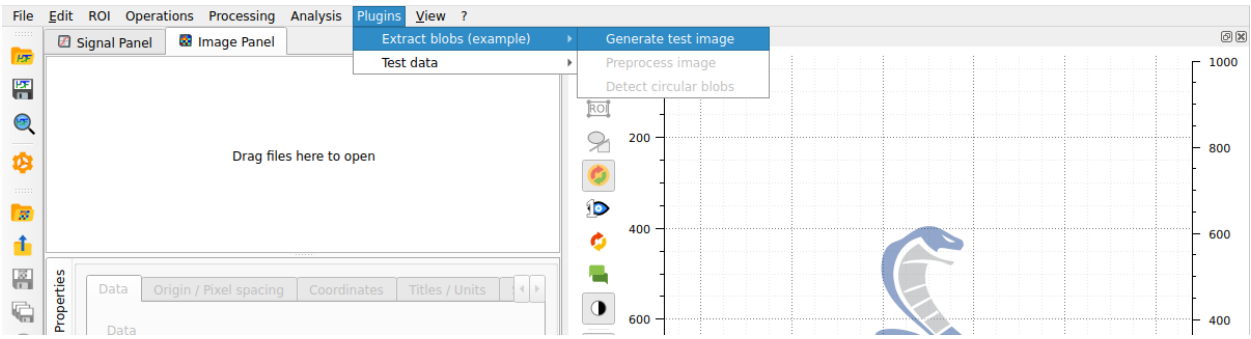


Fig. 34 – Cliquons sur « Extract blobs (example) > Generate test image »

Note : Les menus de DataLab changent entre les images et les signaux. Pour pouvoir voir le menu « Extract blobs (example) », assurez-vous que le panneau « Images » est sélectionné (cliquez sur l'onglet « Images » à gauche).

Une boîte de dialogue contextuelle apparaît, demandant les paramètres de l'image de test à générer (la taille de l'image et son titre). Nous pouvons simplement conserver les paramètres par défaut et cliquer sur « OK ».

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

```
def generate_test_image(self) -> None:
    """Generate test image"""
    newparam = self.edit_new_image_parameters(
        title="Test image", hide_dtype=True, shape=(2048, 2048)
    )
    if newparam is not None:
        # Create a NumPy array:
        shape = (newparam.height, newparam.width)
        arr = np.random.normal(10000, 1000, shape)
        for _ in range(10):
            row = np.random.randint(0, shape[0])
            col = np.random.randint(0, shape[1])
            rr, cc = skimage.draw.disk((row, col), min(shape) // 50, shape=shape)
            arr[rr, cc] -= np.random.randint(5000, 6000)
        center = (shape[0] // 2,) * 2
        rr, cc = skimage.draw.disk(center, min(shape) // 10, shape=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
        obj = sigima.objects.create_image(
            newparam.title, data, units=("mm", "mm", "lsb")
        )
        self.proxy.add_object(obj)
```

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

L'image est un peu bruitée, et aussi assez grande. De plus, les taches sont grandes par rapport à la taille des pixels.

Pour réduire le bruit, il existe plusieurs fonctions disponibles dans DataLab. En raison des considérations que nous ve-

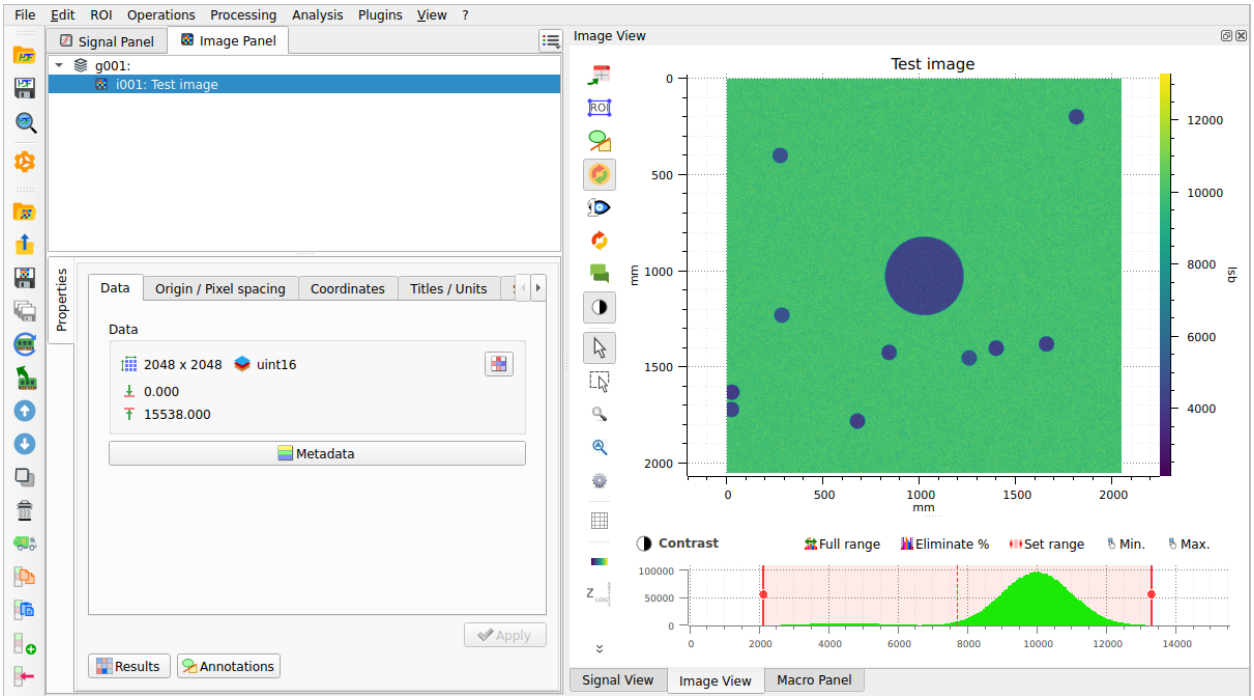


FIG. 35 – Le plugin a généré une image de test, et l'a ajoutée au panneau « Images ». L'image montre quelques taches, avec un disque central plus grand, et un fond bruité.

nons de faire, nous pouvons considérer qu'un binning réduirait le bruit sans perdre l'information que nous recherchons. Appliquons un binning à l'image par un facteur de 2 sur les deux axes. Cela réduira la taille de l'image d'un facteur de 4, et l'écart type du bruit d'un facteur de 2. Choisir de modifier la taille des pixels en conséquence maintiendra la taille des taches constante dans l'unité précédente (dans ce cas, les pixels, mais dans une application réelle, nous pouvons les calibrer pour respecter une taille physique réelle).

Une autre approche que nous pouvons adopter est d'appliquer un filtre médian mobile, pour réduire l'importance des pics. Nous le faisons avec une fenêtre de 5 ; bien sûr, en pratique, différentes tailles de fenêtre peuvent être testées pour trouver un bon compromis entre la réduction du bruit et la résolution. Voyons comment faire cela.

A présent, le débruitage paraît satisfaisant : le niveau de bruit est sensiblement réduit. Nous pouvons donc passer à l'étape suivante : détecter les taches sur l'image. Pour ce faire, nous pouvons utiliser la fonction « Détection de taches » disponible dans le menu « Analyse ». Différents *algorithmes* sont disponibles, ici nous utiliserons celui d'OpenCV.

Note : Si vous souhaitez afficher à nouveau les résultats d'analyse, vous pouvez sélectionner l'entrée « Afficher les résultats » dans le menu « Analyse », ou le bouton « Résultats » , dans le panneau de propriétés en dessous de la liste des images :

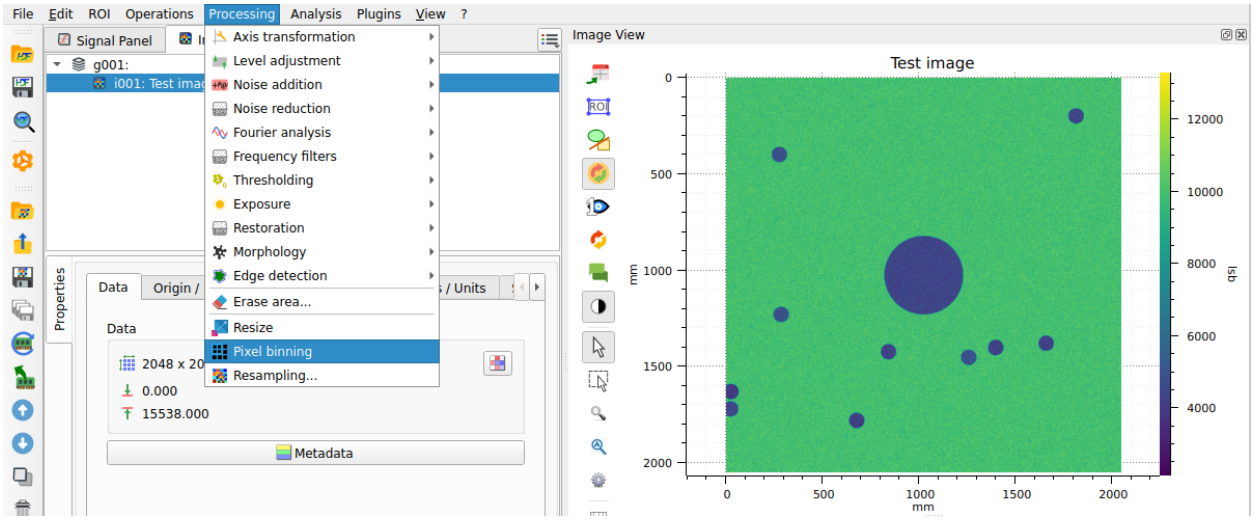


Fig. 36 – Cliquons sur « Opérations > Pixel binning ».

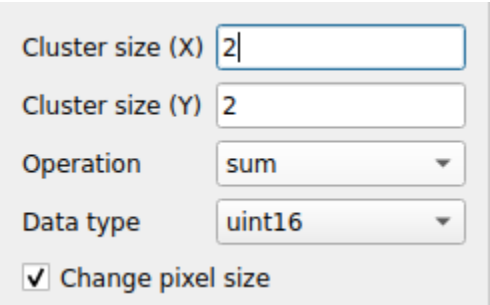


Fig. 37 – La boîte de dialogue « Binning » s'ouvre, avec plusieurs options. Réglez le facteur de binning à 2 pour les deux axes, sélectionnez la case « changer la taille des pixels » et cliquez sur « OK ».

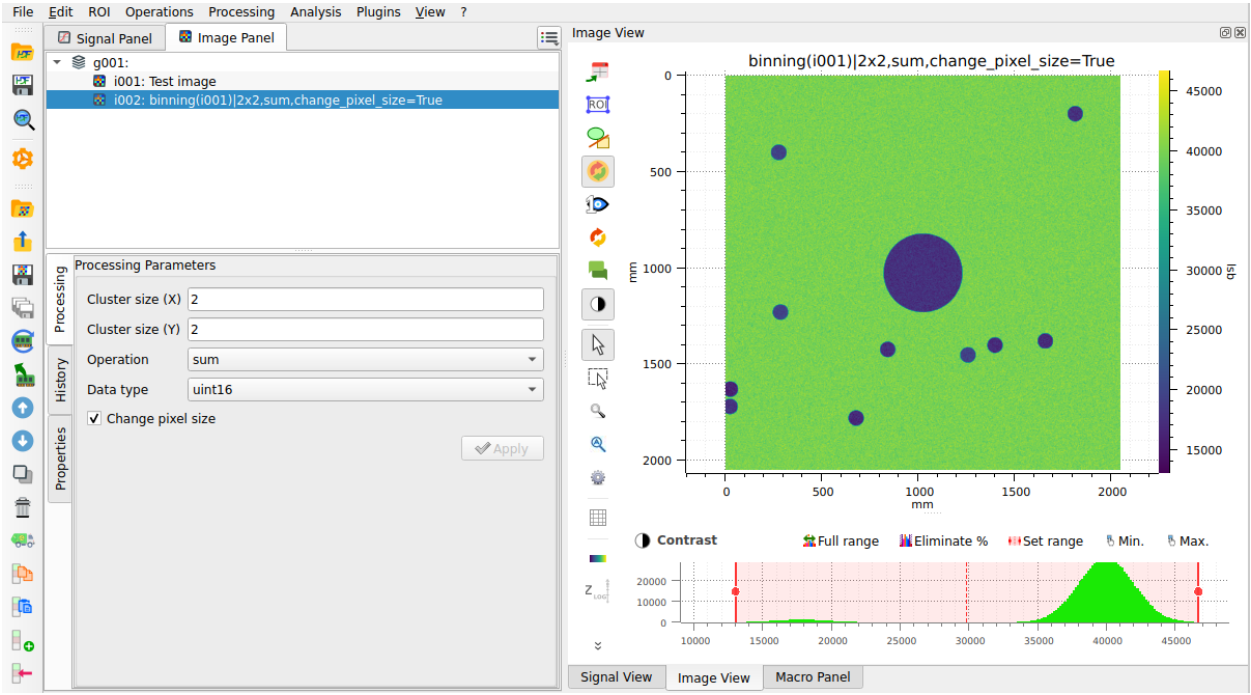


FIG. 38 – L'image binnée est ajoutée au panneau « Images ». Il est maintenant plus facile de voir les taches (même si elles étaient déjà assez visibles sur l'image d'origine : c'est juste un exemple), et l'image sera plus rapide à traiter.

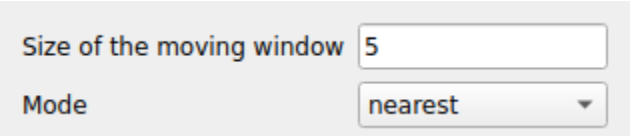


FIG. 39 – Cliquez sur l'entrée « Traitement > Réduction du bruit > Médiane mobile », et réglez la taille de la fenêtre sur 5.

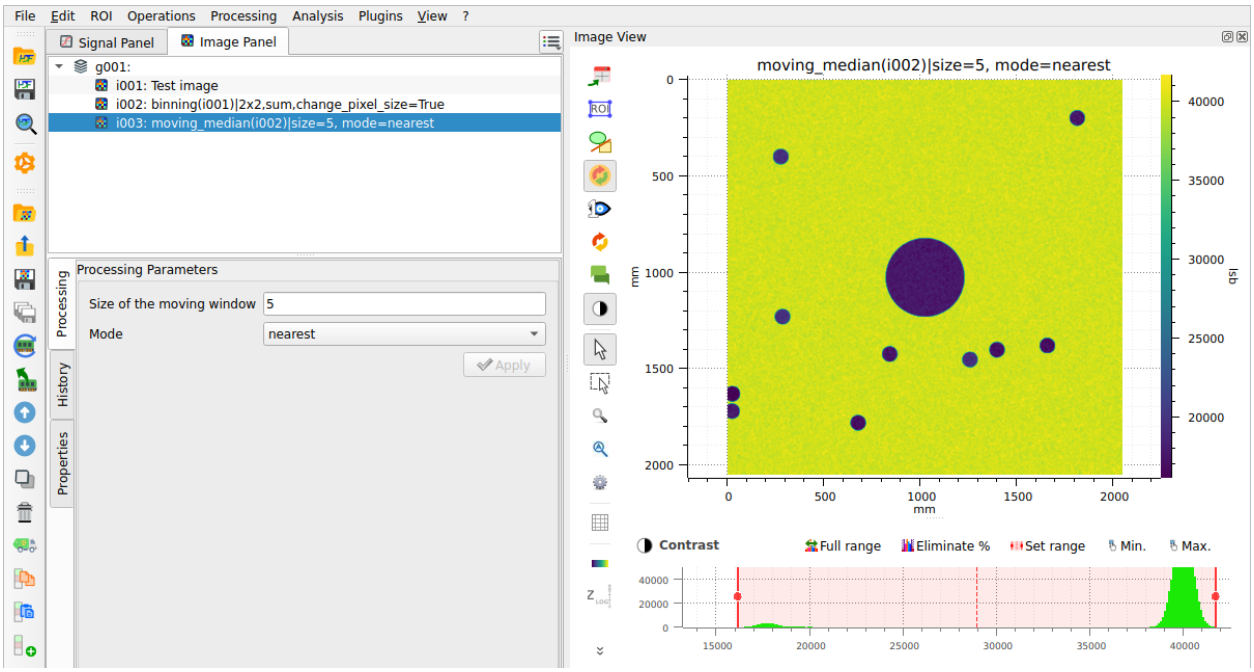


FIG. 40 – L'image filtrée est ajoutée au panneau « Images ». Le débruitage est assez efficace.

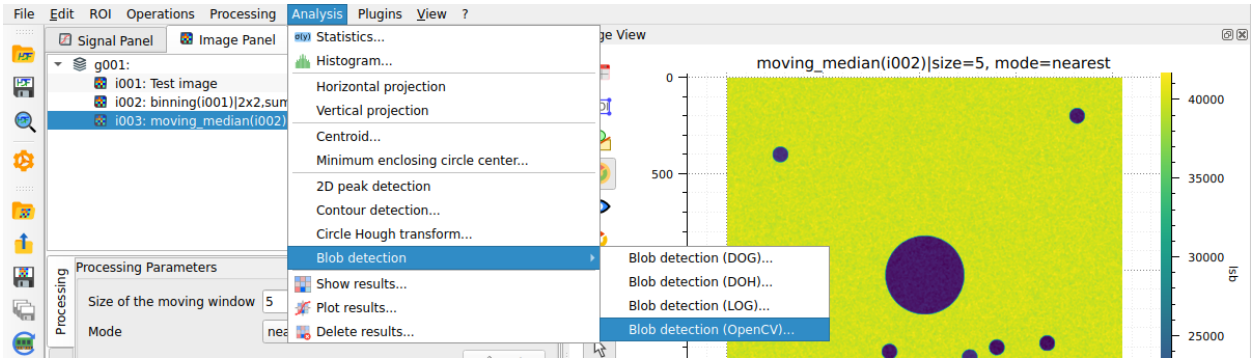


FIG. 41 – Cliquons sur « Analysis > Blob detection > Blob detection (OpenCV) ».

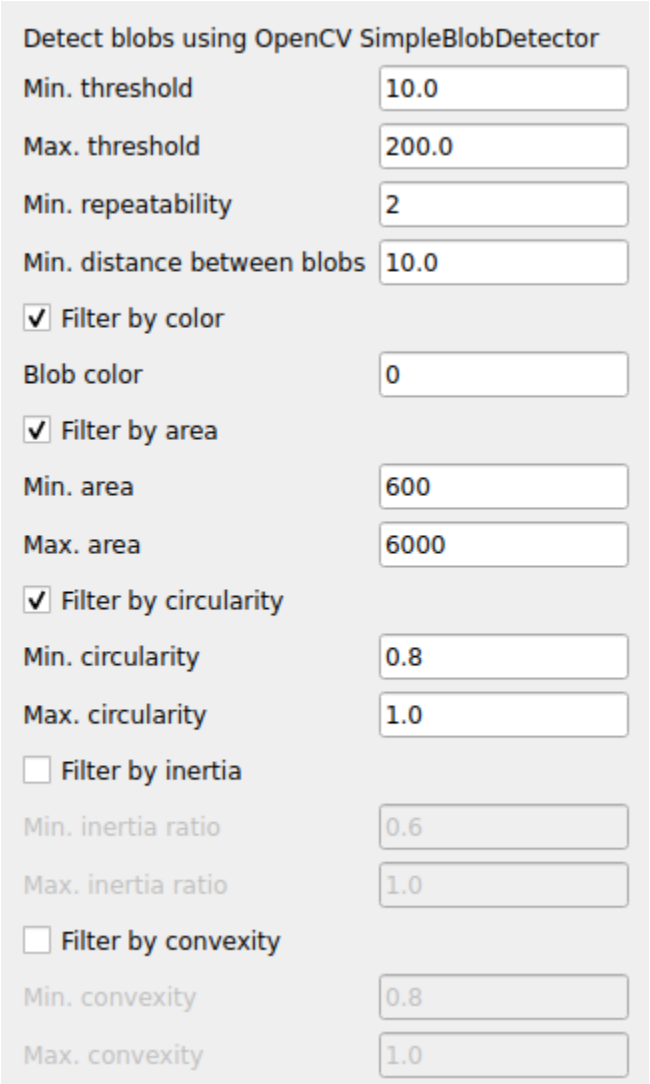


Fig. 42 – La boîte de dialogue « Blob detection (OpenCV) » s'ouvre. Réglez les paramètres comme indiqué sur la capture d'écran, et cliquez sur « OK ».

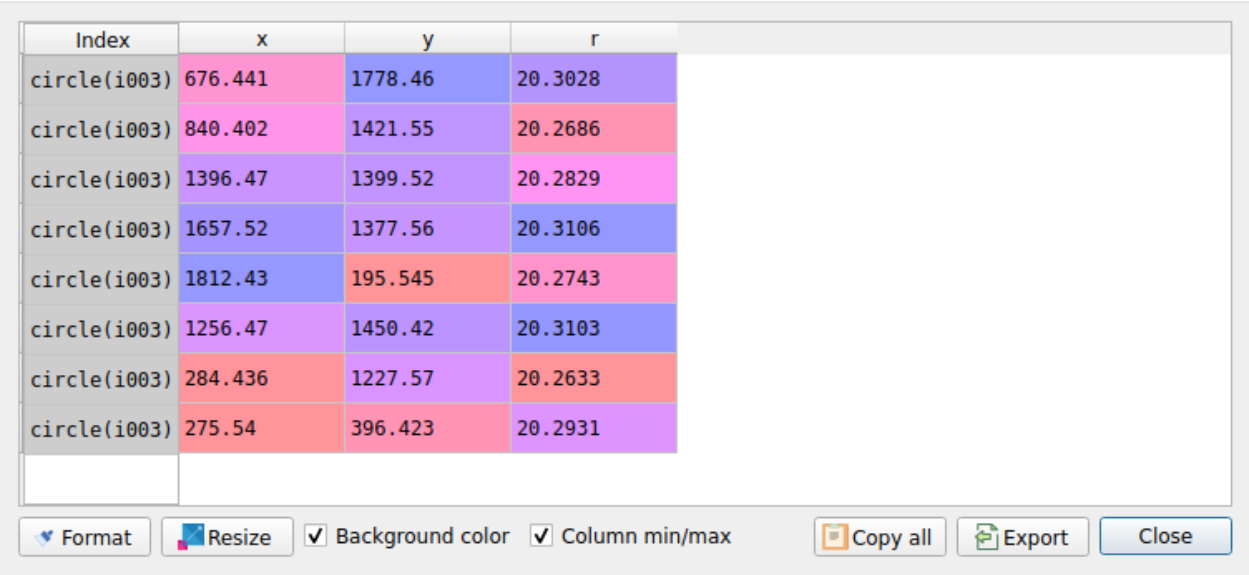


Fig. 43 – La boîte de dialogue « Résultats » s'ouvre, montrant les taches détectées : une ligne par tache, avec les coordonnées et le rayon de la tache.

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.

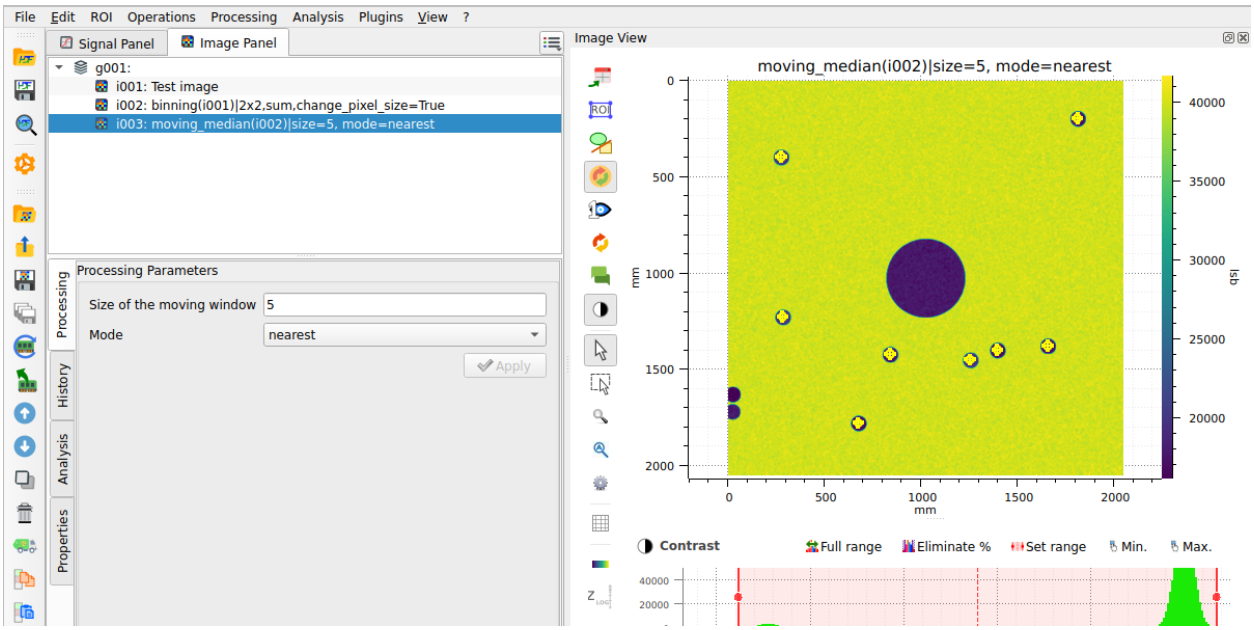


FIG. 44 – Les taches détectées sont également ajoutées aux métadonnées de l'image, et peuvent être visualisées dans le panneau de visualisation à droite.

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

Charger les images

Note : Les images utilisées dans ce tutoriel « fabry_perot1.jpg » et « fabry_perot2.jpg » sont disponibles dans le dossier de données du tutoriel du répertoire d'installation de DataLab (`<Répertoire d'installation de DataLab>/data/tutorials/`).

Alternativement, elles peuvent être téléchargées depuis la documentation en ligne à l'adresse https://datalab-platform.com.

Tout d'abord, nous ouvrons DataLab et chargeons les images depuis le menu « Fichier > Ouvrir une image… », avec le bouton dans la barre d'outils ou en faisant glisser-déposer les fichiers dans DataLab.

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

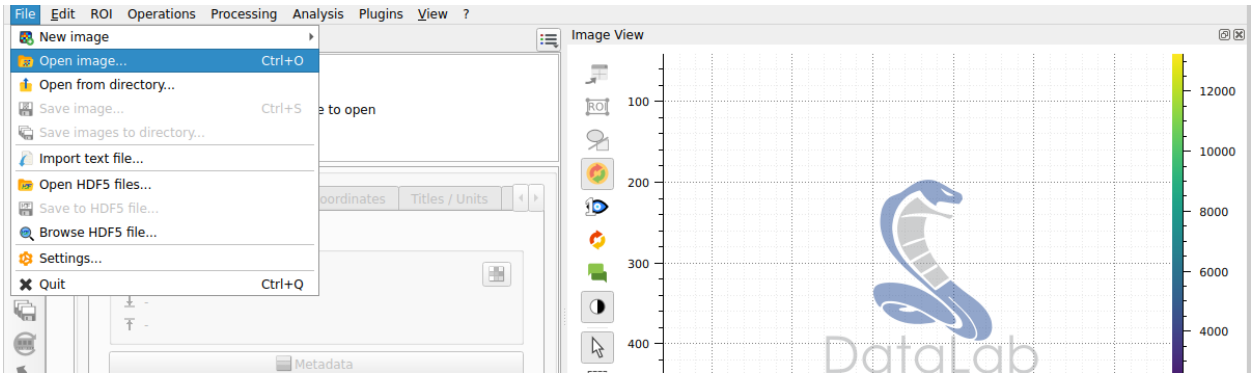


FIG. 45 – Ouvrez les fichiers d'image avec « Fichier > Ouvrir une image… », ou avec le bouton dans la barre d'outils, ou en faisant glisser-déposer les fichiers dans DataLab (sur le panneau de droite).

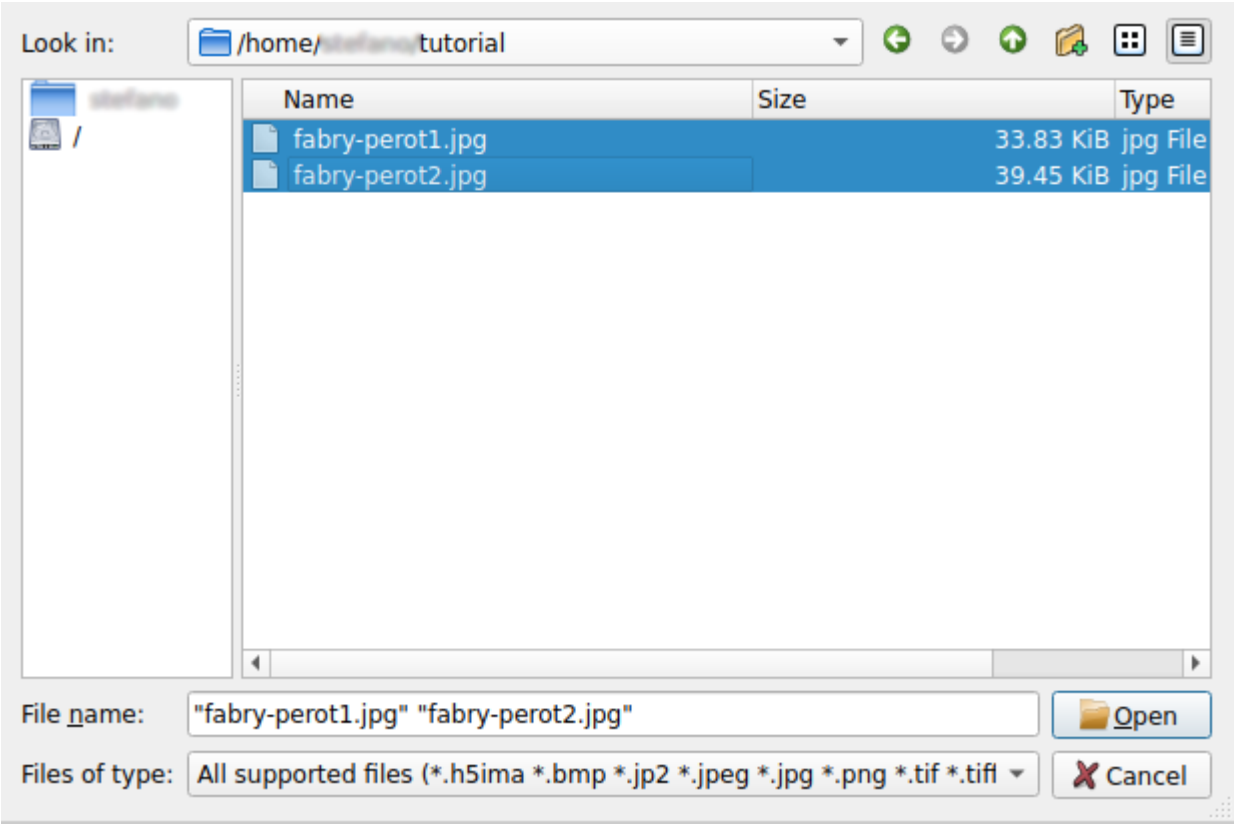


FIG. 46 – Allez dans le dossier de données du tutoriel dans le répertoire d'installation de DataLab (ou le dossier où vous avez placé les images), sélectionnez « fabry_perot1.jpg » et « fabry_perot2.jpg » et cliquez sur « Ouvrir ».

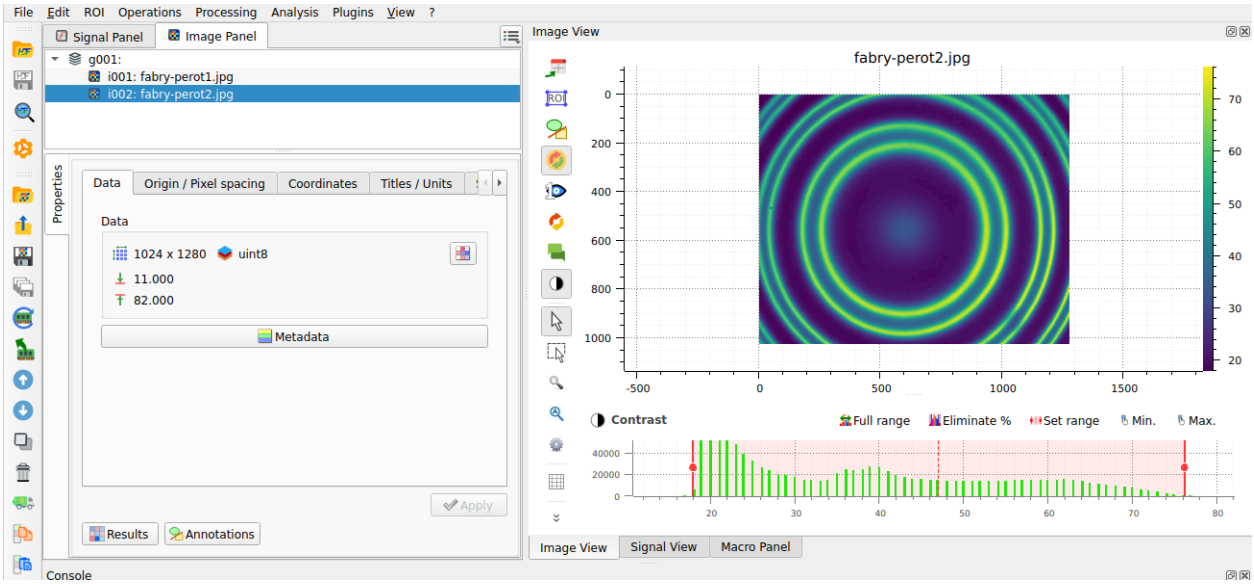


FIG. 47 – Images chargées dans DataLab. Zoomez avec le bouton droit de la souris, déplacez l'image avec le bouton du milieu de la souris.

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 palette de couleurs en une palette de couleurs en niveaux de gris. Si vous voyez une palette de couleurs 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 palette de couleurs dans la barre d'outils verticale à gauche du panneau de visualisation.

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

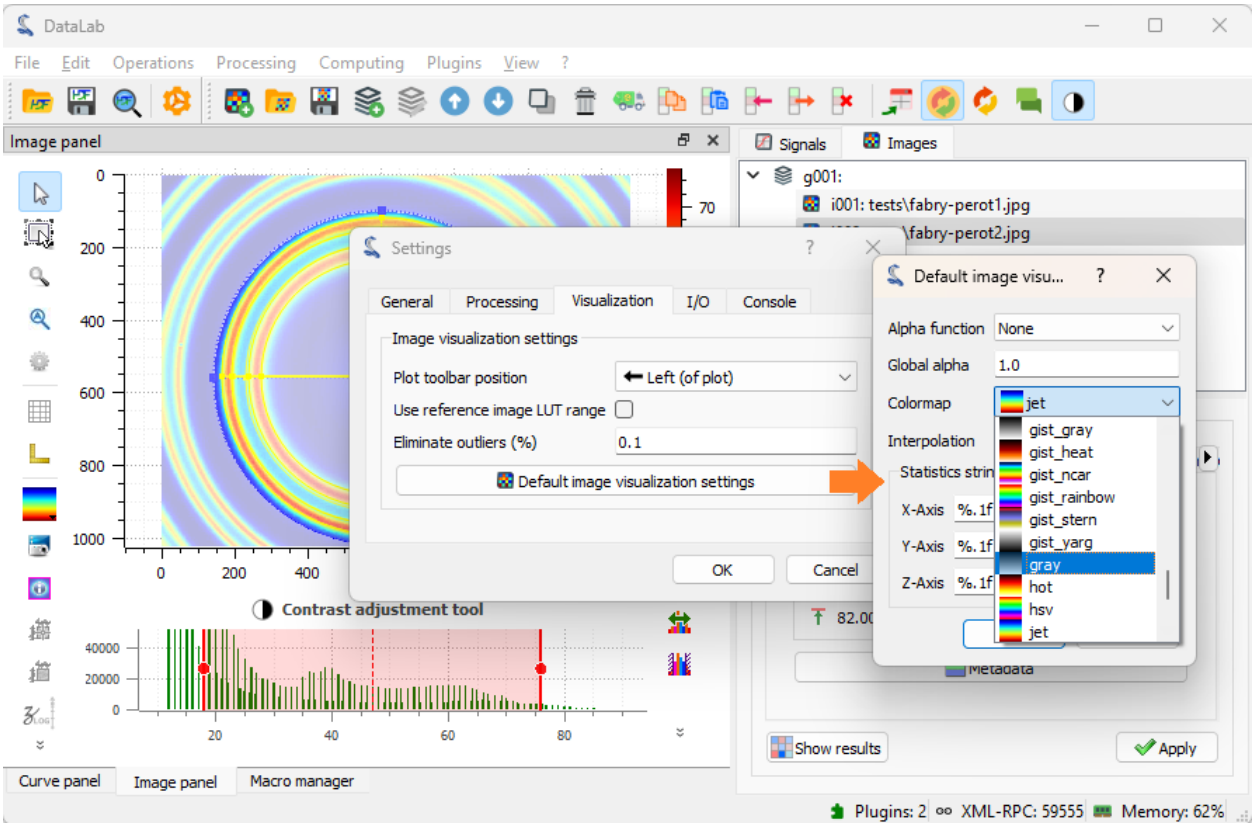


FIG. 48 – Sélectionnez l'onglet « Visualisation », et sélectionnez la palette de couleurs « gray ».

Définir des ROIs circulaires et ajuster des contours

Définissons une région d'intérêt circulaire (ROI) autour de la frange centrale. Pour ce faire, nous sélectionnons d'abord la première image dans le panneau « Images » (si elle n'est pas déjà sélectionnée), puis nous sélectionnons l'outil « Modifier les régions d'intérêt » dans le menu « ROI ».

Cela ouvre la boîte de dialogue « Régions d'intérêt », qui fournit tous les outils pour définir et éditer des ROIs. Vous pouvez définir des ROIs graphiquement ou via leurs coordonnées. Vous pouvez définir plusieurs ROIs sur la même image, et même les combiner en utilisant des opérations booléennes (union, intersection, différence).

Nous choisissons de définir une ROI circulaire graphiquement. Pour ce faire, nous cliquons sur le bouton « ROI graphique », et sélectionnons l'option de ROI circulaire. Nous pouvons maintenant dessiner la ROI circulaire sur l'image en cliquant et en faisant glisser la souris : le premier clic est d'un côté du cercle, et le deuxième clic est du côté opposé du cercle.

Une fois créée, la ROI peut être sélectionnée à l'aide de l'outil « Sélection », et déplacée ou redimensionnée en faisant glisser la ROI ou ses poignées. De plus, vous pouvez modifier le rayon de la ROI tout en gardant son centre fixe en appuyant sur la touche « Ctrl » tout en faisant glisser une poignée.

Une fois que vous êtes satisfait de la position et de la taille de la ROI, cliquez sur « OK » pour fermer la boîte de dialogue. Une boîte de dialogue de confirmation s'ouvre pour vous demander d'appliquer la ROI à l'image, et de confirmer ses paramètres. Ici, vous pouvez également choisir d'inverser le masque de la ROI (c'est-à-dire sélectionner la zone en dehors de la ROI au lieu de la zone à l'intérieur de la ROI) : nous ne voulons pas inverser la ROI dans ce cas, donc nous laissons la case à cocher « Inverser la ROI » décochée, et cliquons sur « OK ». Si vous souhaitez obtenir les

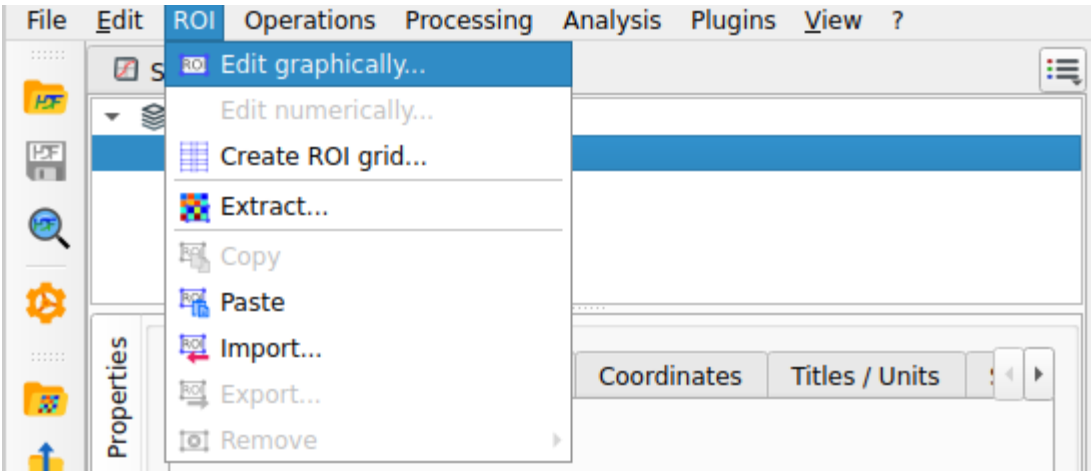


FIG. 49 – Sélectionnez l'outil « Édition graphique » dans le menu « ROI ».

mêmes résultats que ceux présentés dans ce tutoriel, assurez-vous que les paramètres de la ROI sont les mêmes que dans l'image.

Une fois confirmée, la ROI est appliquée à l'image. La ROI est affichée sur l'image, et la partie en dehors de la ROI est assombrie. Notez que, en interne, la ROI est définie par un masque binaire, c'est-à-dire que les données d'image sont représentées comme un tableau masqué NumPy.

A présent, détectons les contours dans la zone à l'intérieur de la ROI et ajustons-les à des cercles. Pour ce faire, nous sélectionnons l'outil « Détection de contour » dans le menu « Analyse ». La boîte de dialogue des paramètres de contour s'ouvre : nous sélectionnons la forme « Cercle » et cliquons sur « OK ». Vous verrez maintenant une boîte de dialogue « Résultats » s'ouvrir, qui affiche les paramètres du cercle ajusté. Cliquez sur « OK » pour fermer la boîte de dialogue. Les cercles ajustés sont affichés sur l'image.

Note : Si vous souhaitez afficher à nouveau les résultats de l'analyse, vous pouvez sélectionner le bouton « Afficher les résultats » dans le menu « Analyse », ou le bouton « Résultats » , dans le panneau d'analyse situé sous la liste des images :

Les images (ou signaux) peuvent également être affichées 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). Cela est utile pour comparer des images ou des signaux côte à côte, et vous permet d'ajouter des annotations (curseurs, étiquettes, formes, etc.) à l'image ou au signal : les annotations sont stockées dans les métadonnées de l'image ou du signal, et avec les données lorsque l'espace de travail est sauvegardé.

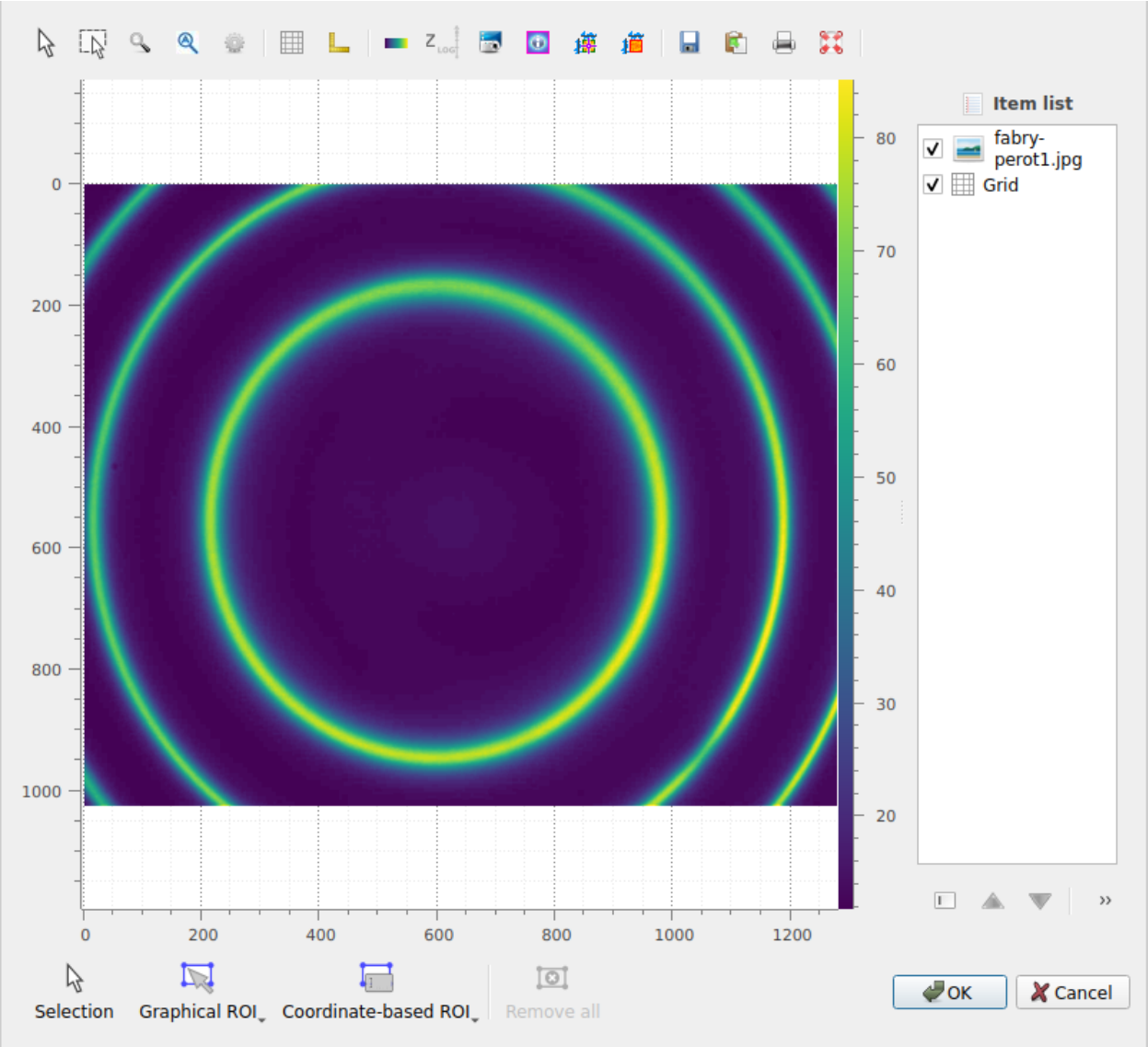


Fig. 50 – La boîte de dialogue « Régions d'intérêt » s'ouvre. Cliquez sur « Ajouter une ROI » et sélectionnez une ROI circulaire. Redimensionnez la ROI prédéfinie en faisant glisser les poignées. Notez que vous pouvez modifier le rayon de la ROI tout en gardant son centre fixe en appuyant sur la touche « Ctrl ». Cliquez sur « OK » pour fermer la boîte de dialogue.

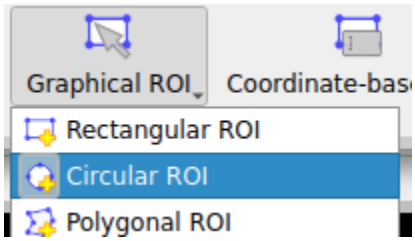


Fig. 51 – L'option pour définir graphiquement une ROI circulaire.

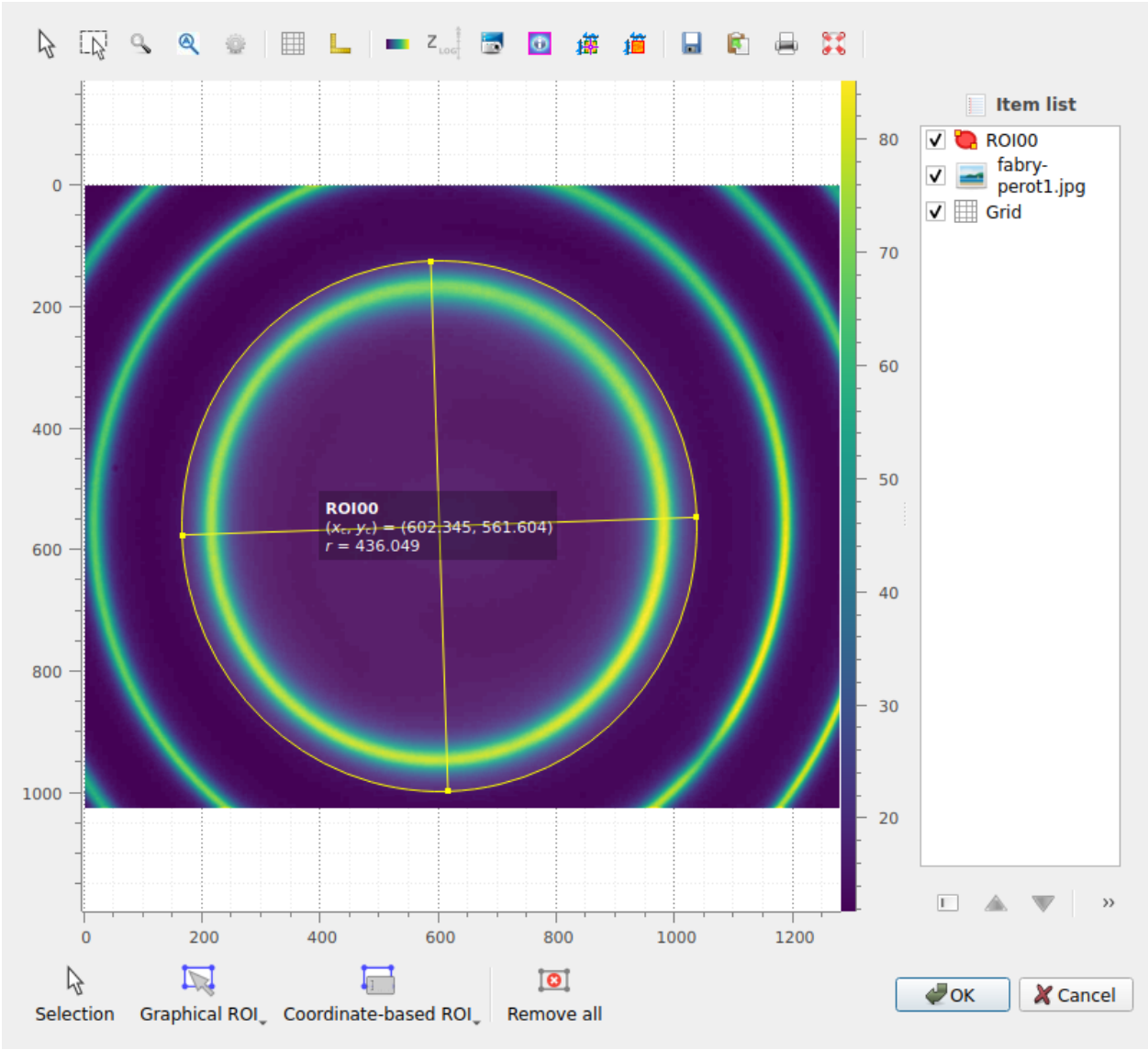


FIG. 52 – La ROI circulaire créée apparaît dans l'image.

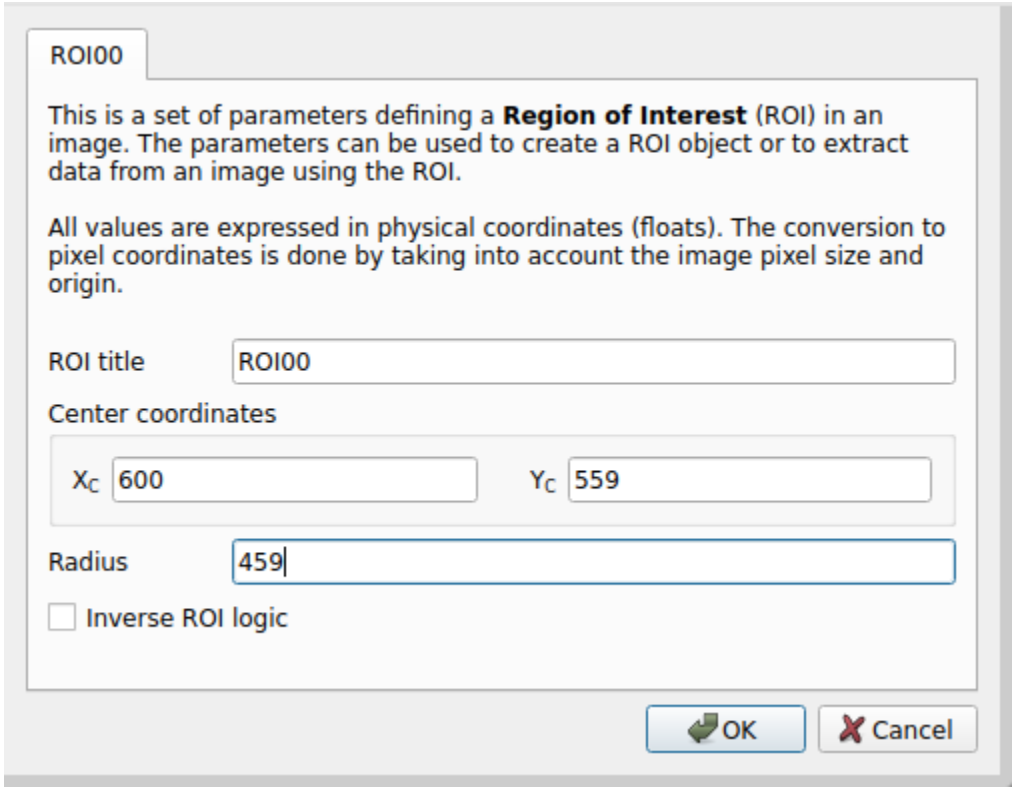


FIG. 53 – La boîte de dialogue de confirmation.

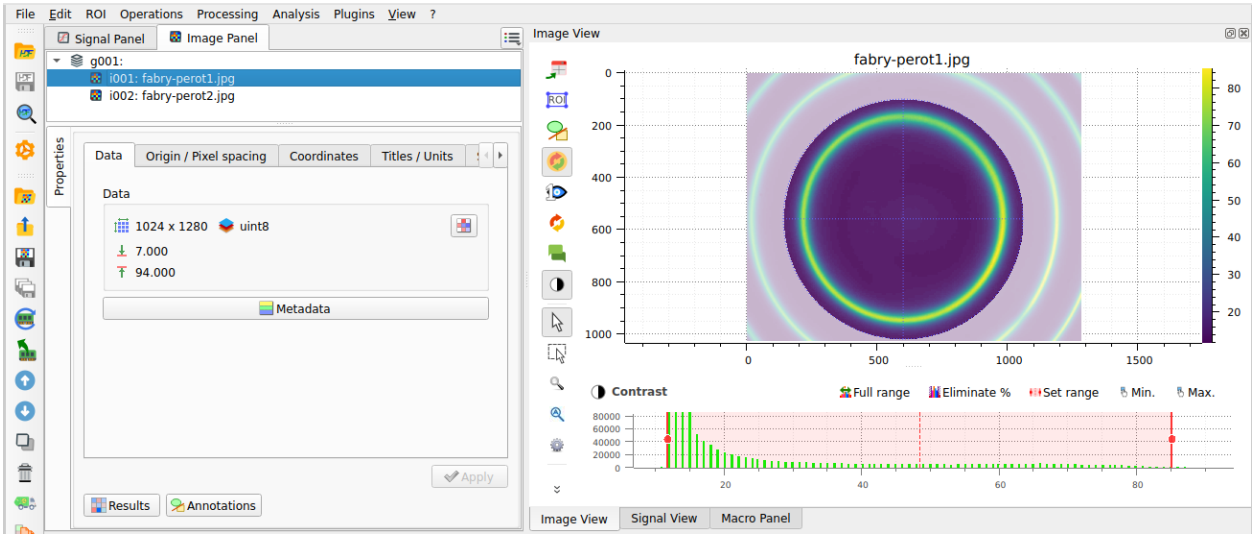


FIG. 54 – La ROI affichée sur l'image.

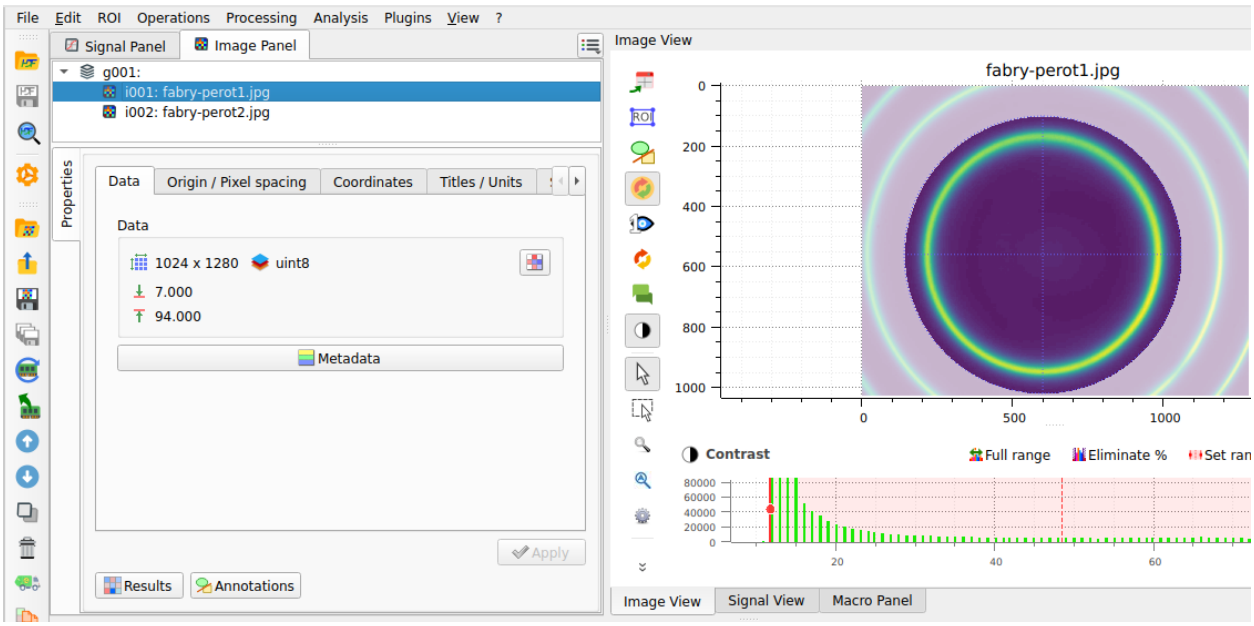


FIG. 55 – L'outil « Détection de contours » dans le menu « Analyse ».

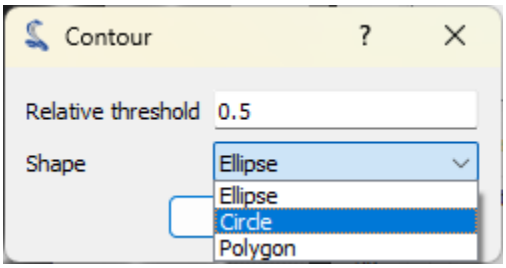
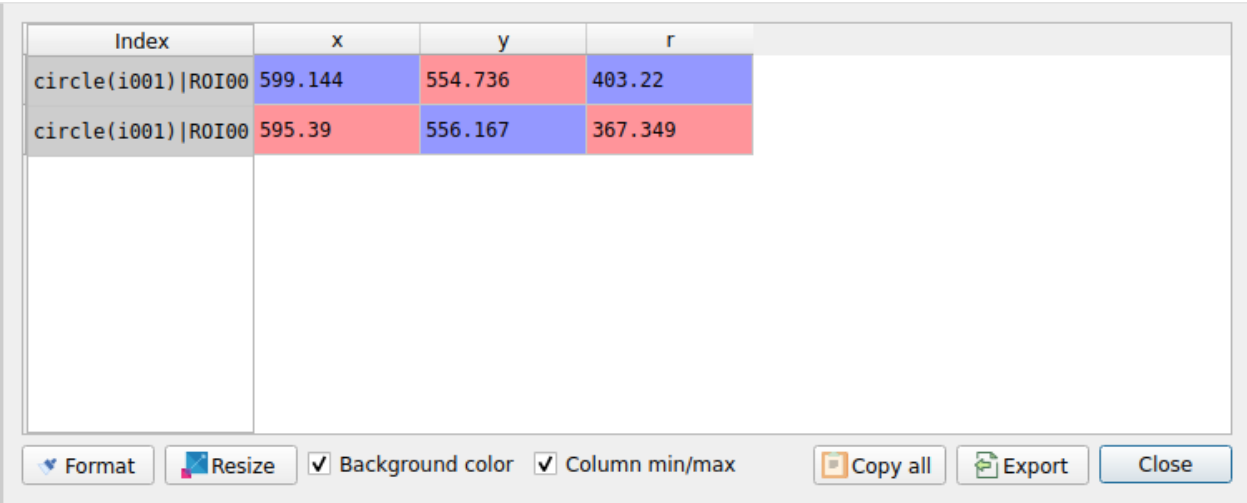


FIG. 56 – La boîte de dialogue des paramètres de contour.



Index	x	y	r
circle(i001)|ROI00	599.144	554.736	403.22
circle(i001)|ROI00	595.39	556.167	367.349

FIG. 57 – La boîte de dialogue des résultats.

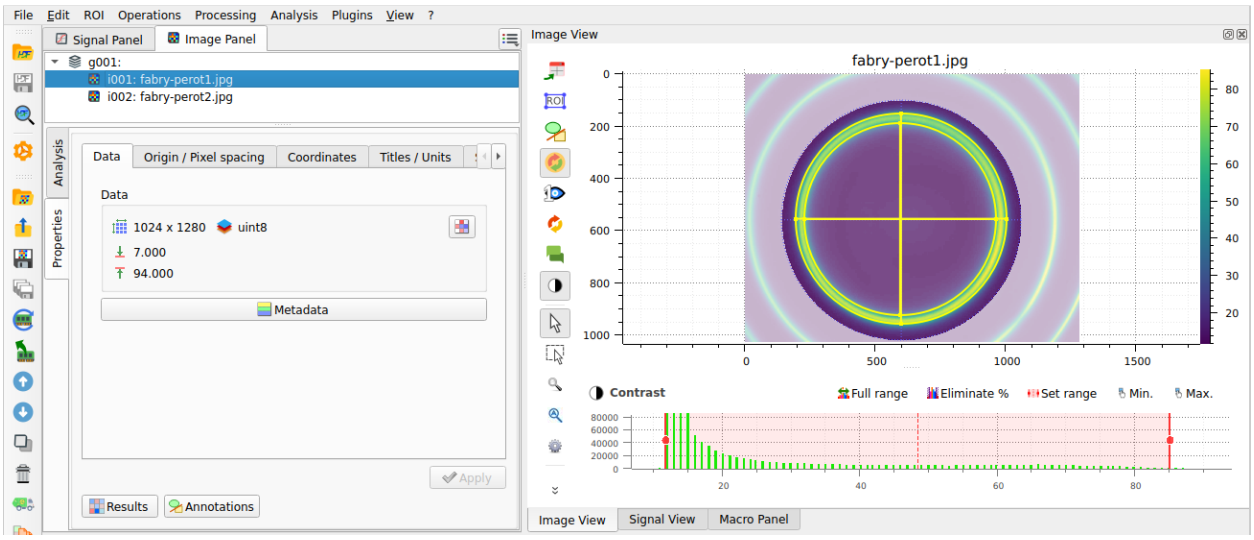


FIG. 58 – Les cercles ajustés sont affichés sur l'image.

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. Sélectionnez l'entrée « Supprimer les métadonnées de l'objet… » dans le menu « Edition », ou le bouton dans la barre d'outils, et répondez « Non » à la boîte de dialogue de confirmation pour conserver la ROI et supprimer le reste des métadonnées.

Si nous voulons définir la même ROI exacte sur la deuxième image, nous pouvons copier/coller la ROI de la première image à la deuxième image. La meilleure façon de le faire est d'utiliser l'option de copier/coller présente dans le menu « ROI », ou les boutons et dans la barre d'outils : sélectionnez la première image, puis cliquez sur l'entrée « Copier » dans le menu « ROI » (ou le bouton dans la barre d'outils), puis sélectionnez la deuxième image, et cliquez sur l'entrée « Coller la ROI » dans le menu « ROI » (ou le bouton dans la barre d'outils).

Note : Copier/coller les métadonnées copie/colle toutes les métadonnées, y compris les ROIs, mais cela aurait pour effet secondaire de copier/coller également le reste des métadonnées.

Sélectionnez l'outil « Détection de contours » dans le menu « Analyse », 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 ».

Extraire des profils d'intensité le long de l'axe X

Note : Pour des raisons de simplicité (et parce que nous voulons comparer deux méthodes d'extraction), nous allons extraire le profil d'intensité le long de l'axe X au centre de l'image. Dans une application réelle, vous voudriez probablement extraire le **profil d'intensité radial** à la place, ce qui peut être fait en utilisant l'entrée « Profil radial » dans le menu « Opérations > Profils d'intensité » : cela serait plus pertinent et plus simple pour analyser les franges de Fabry-Perot.

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

- 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.

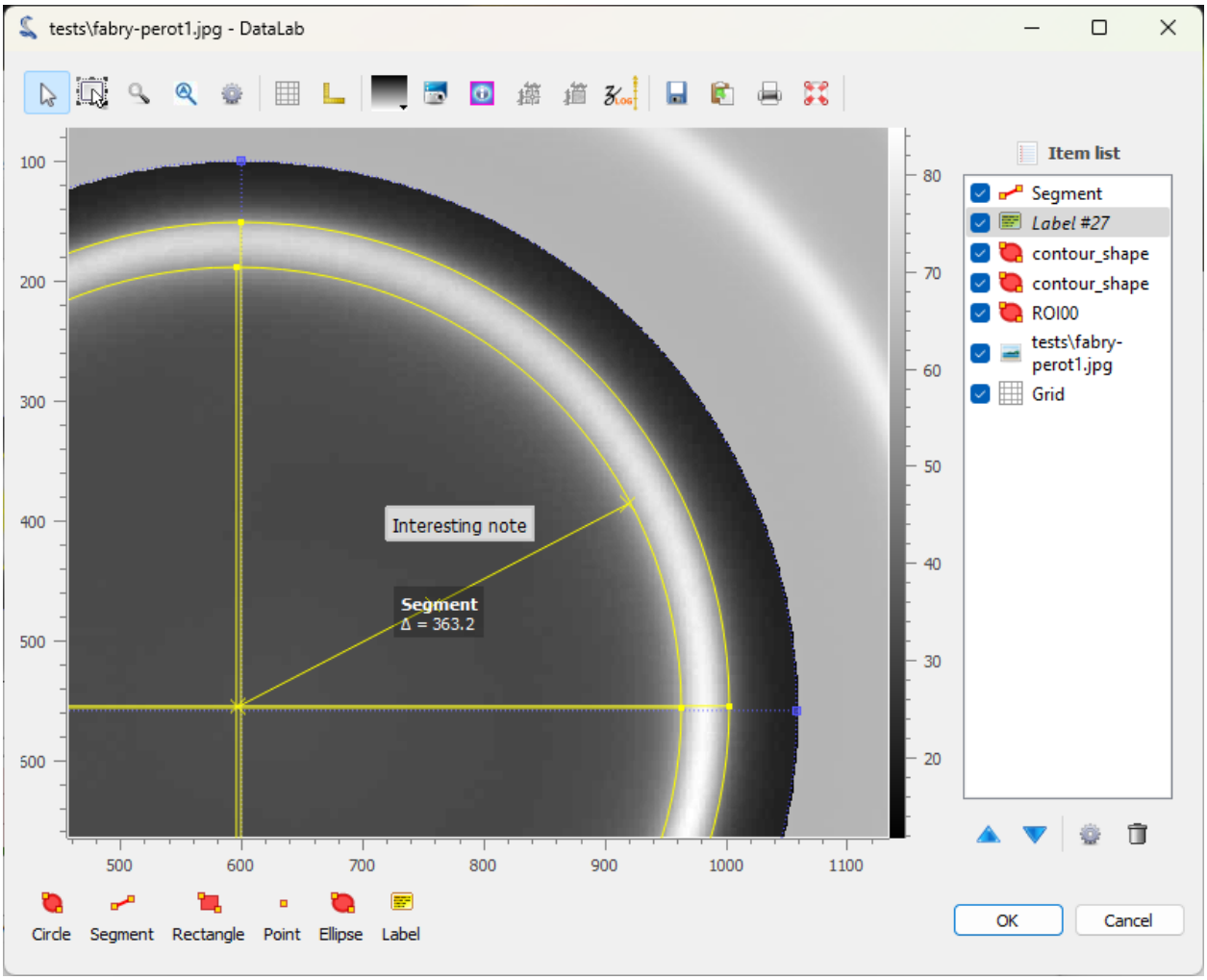


FIG. 59 – L'image affichée dans une fenêtre séparée. La ROI et les cercles ajustés sont également affichés. Des annotations peuvent être ajoutées à l'image en cliquant sur les boutons en bas de la fenêtre. Les annotations sont stockées dans les métadonnées de l'image, et avec les données de l'image lorsque l'espace de travail est sauvegardé. Cliquez sur « OK » pour fermer la fenêtre.

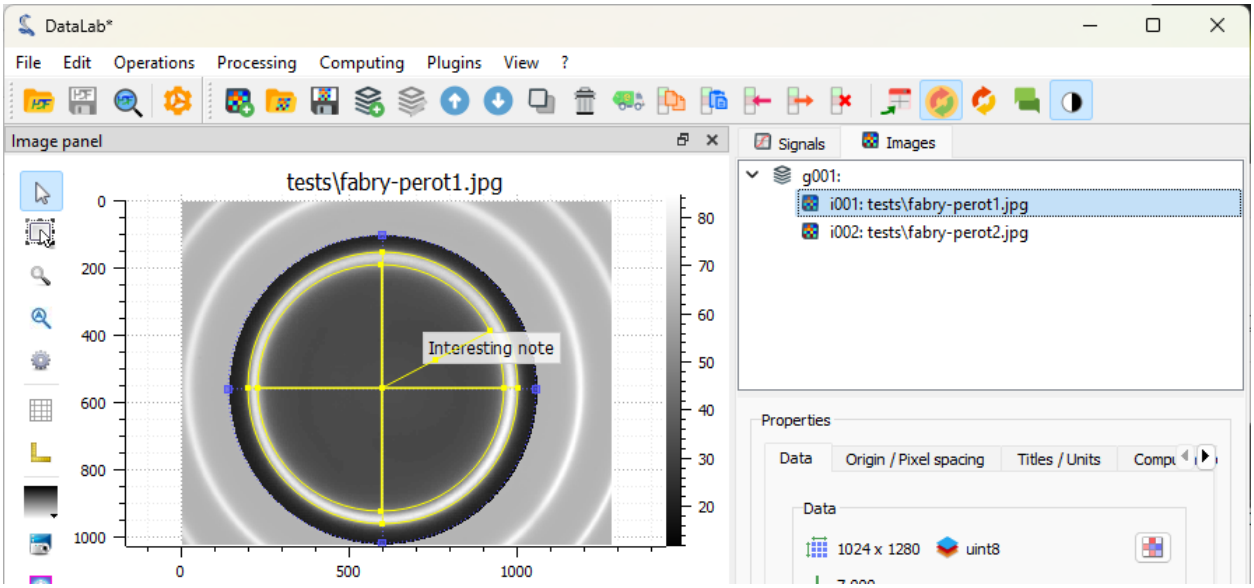


FIG. 60 – L'image est affichée dans la fenêtre principale, avec les annotations.

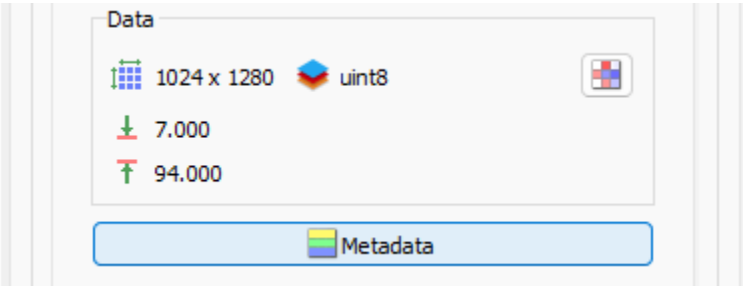


FIG. 61 – Le bouton « Métadonnées » est situé en dessous de la liste des images.

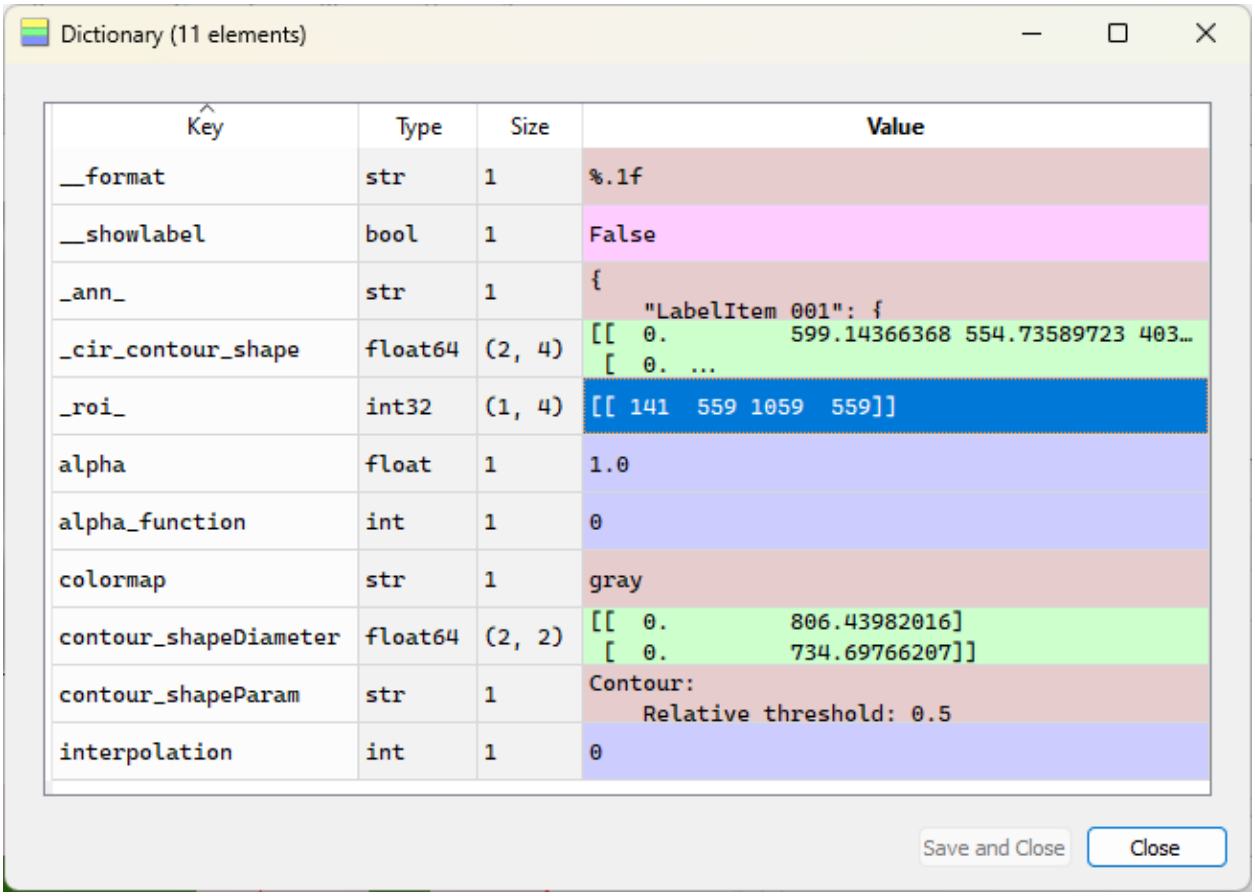


Fig. 62 – La boîte de dialogue « Métadonnées » s'ouvre. Parmi d'autres informations, elle affiche les annotations (dans un format JSON), des informations de style (par exemple la palette de couleurs), et la ROI.

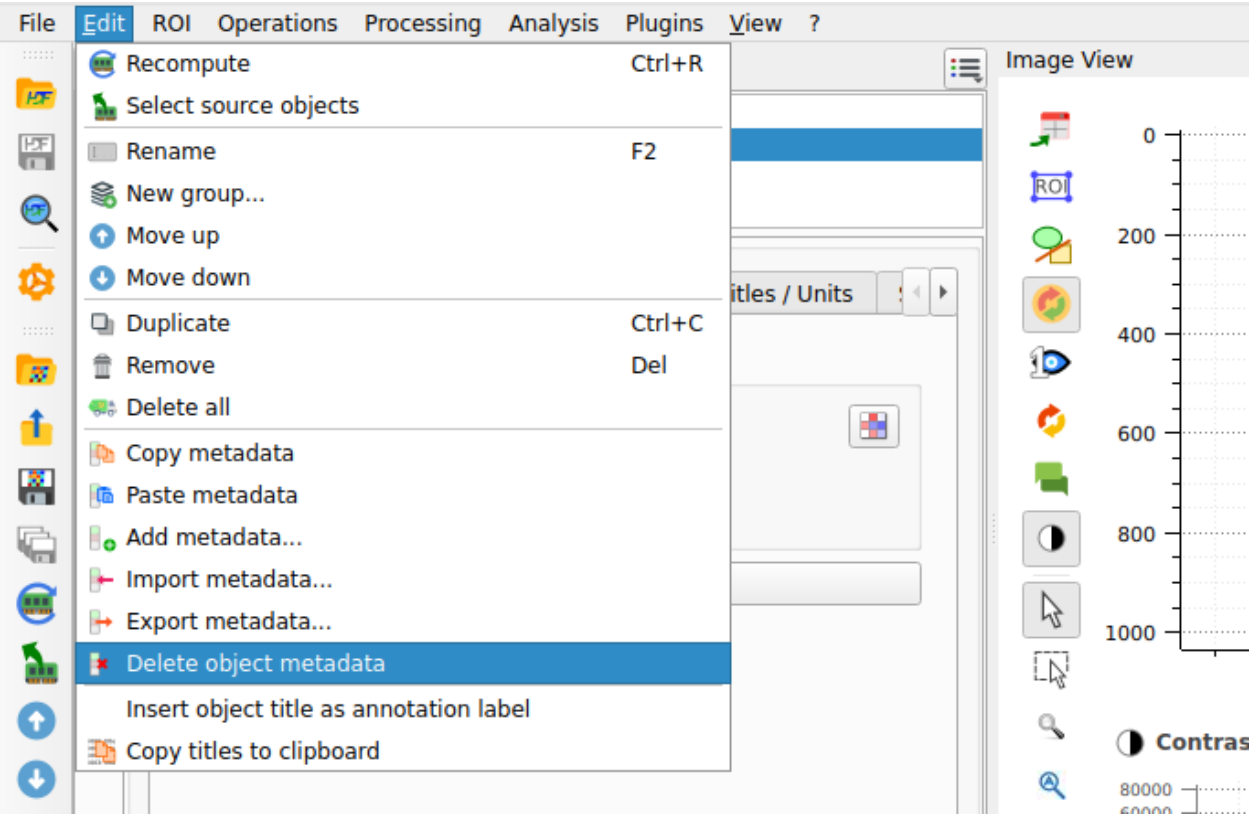


Fig. 63 – Sélectionnez l'entrée « Supprimer les métadonnées de l'objet » dans le menu « Édition ».

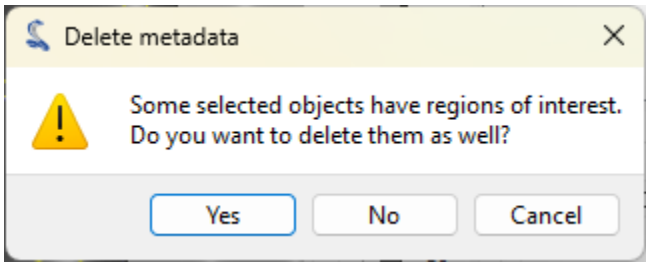


Fig. 64 – La boîte de dialogue « Supprimer les métadonnées ». Cliquez sur « Non » pour conserver la ROI et supprimer le reste des métadonnées.

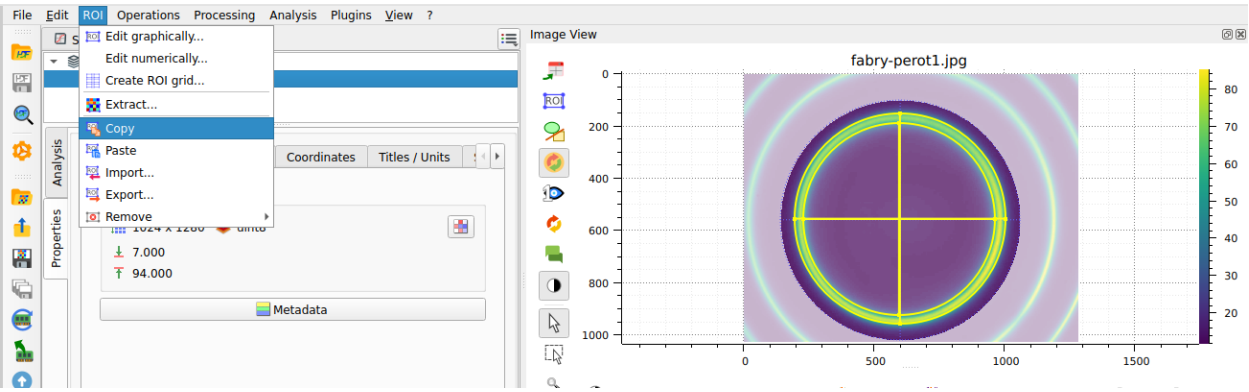


Fig. 65 – L'entrée « Copier » dans le menu « ROI ».

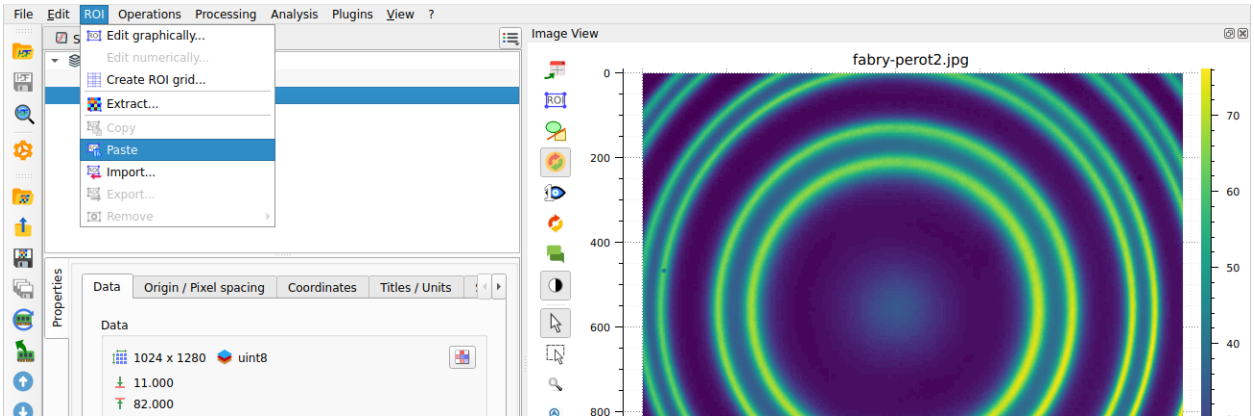


FIG. 66 – L'entrée « Coller » dans le menu « ROI », ou le bouton dans la barre d'outils.

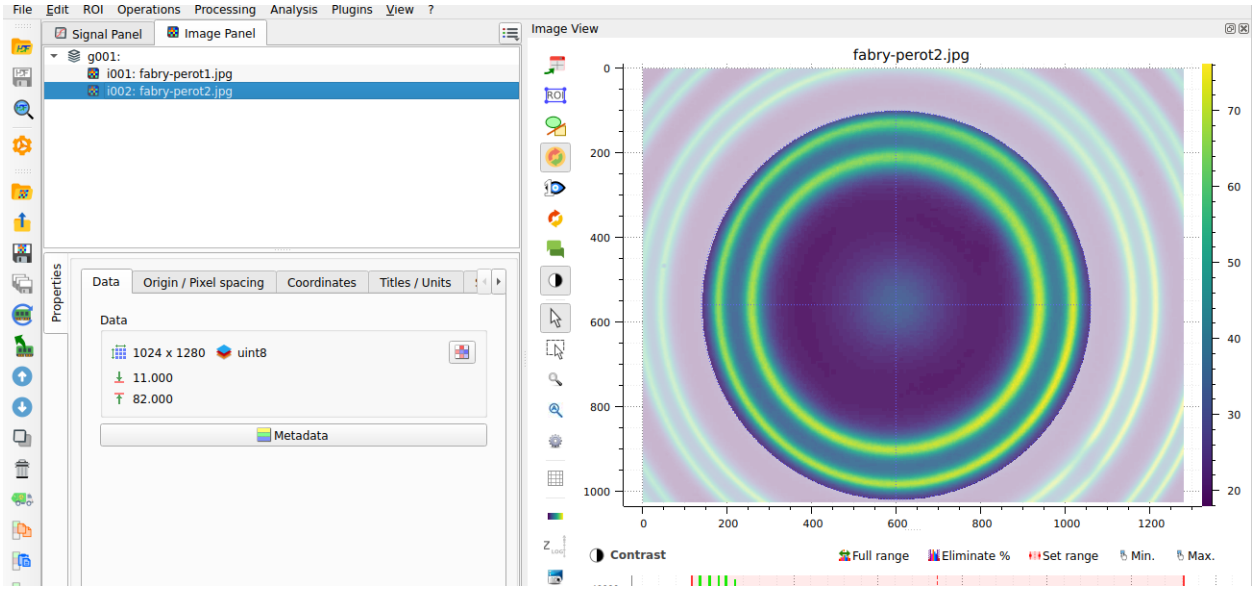


FIG. 67 – La ROI est ajoutée à la deuxième image.

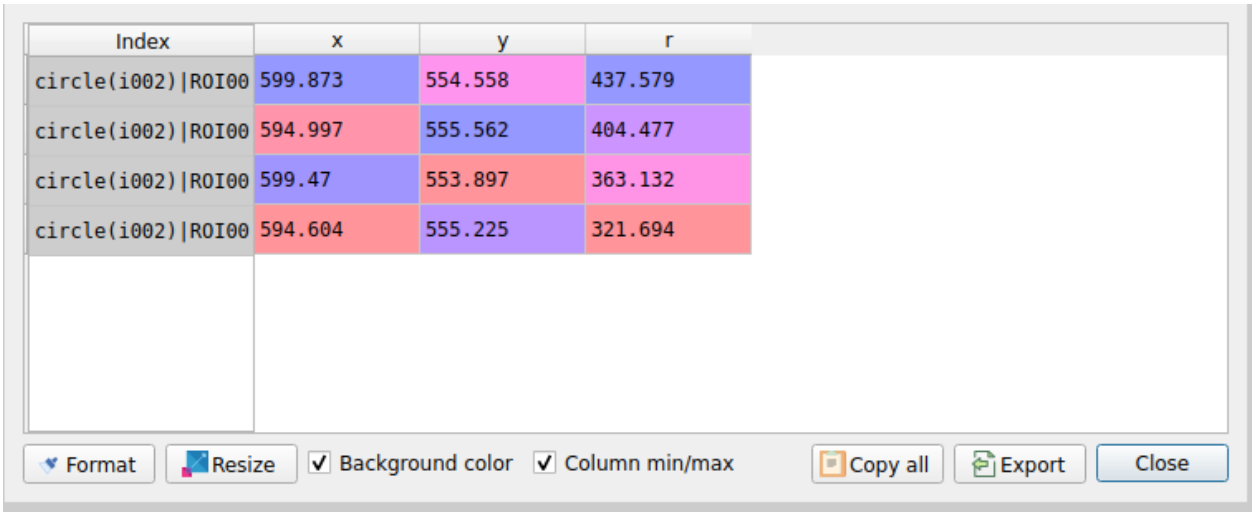


FIG. 68 – La boîte de dialogue « Résultats » pour la deuxième image.

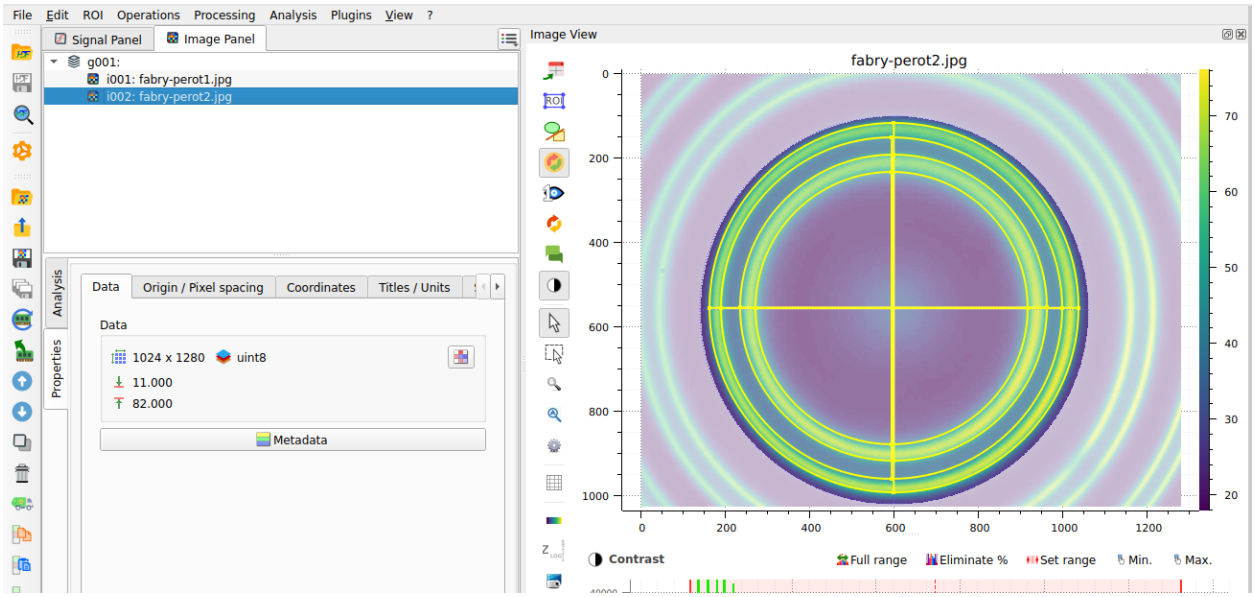


FIG. 69 – Les cercles ajustés sont affichés sur l'image.

Essayons la première option : sélectionnez l'entrée « Profil rectiligne… » . C'est la façon la plus simple d'extraire un profil d'une image, et cela correspond à la fonctionnalité de calcul `calc("line_profile")` de l'API de DataLab (donc elle peut être utilisée dans un script, un plugin ou une macro). Sélectionnez l'entrée « Profil rectiligne… » dans le menu « Opérations » et la boîte de dialogue « Profil rectiligne » s'ouvrira, vous permettant de sélectionner la ligne du profil (pour un profil horizontal) ou la colonne (pour un profil vertical) sur l'image. Vous pouvez également entrer directement le numéro de ligne/colonne en cliquant sur « paramètres… » dans la boîte de dialogue « Profil rectiligne ».

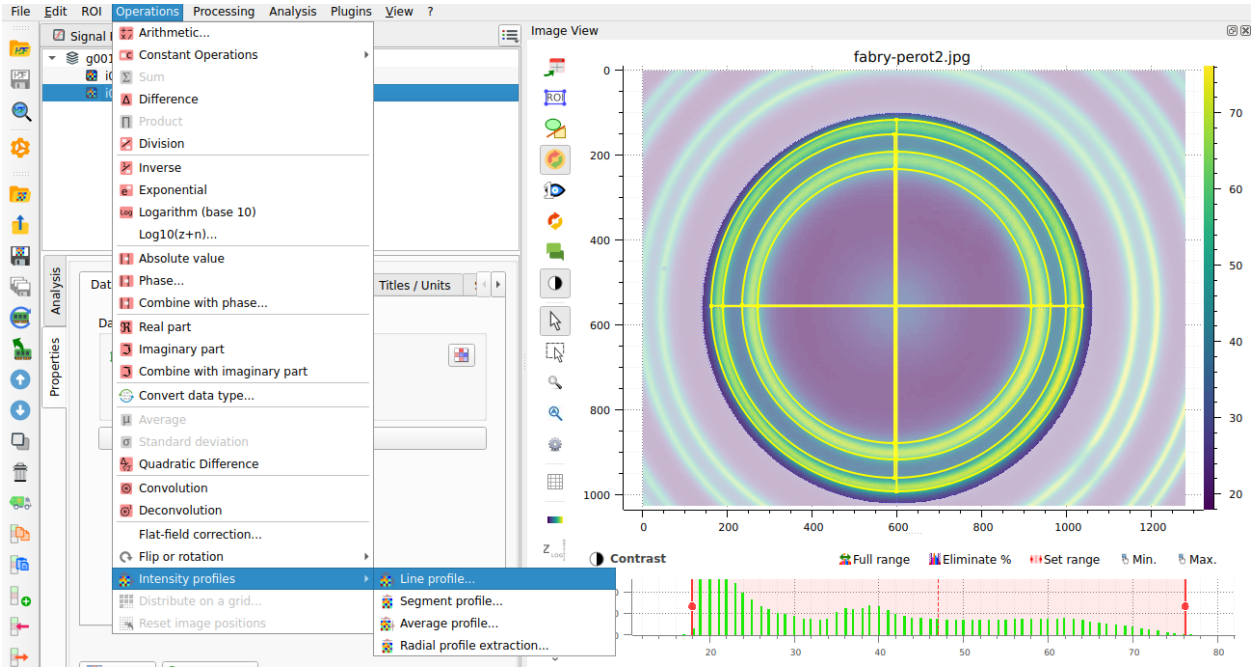


Fig. 70 – Sélectionnez l'entrée « Profil rectiligne… » dans le menu « Opérations ».

Après avoir cliqué sur « OK », le profil d'intensité le long de l'axe X est calculé et affiché dans le panneau « Signaux ». DataLab bascule automatiquement vers le panneau « Signaux » pour afficher le profil.

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).

Essayons maintenant la deuxième option pour extraire le profil d'intensité le long de l'axe X, en utilisant 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 grisé). Ensuite, nous pouvons cliquer sur l'image pour afficher les profils d'intensité le long des axes X et Y. DataLab intègre une version modifiée de cet outil qui vous permet de transférer le profil vers le panneau « Signaux » pour un traitement ultérieur. Sélectionnez l'outil « Profil rectiligne » dans la barre d'outils verticale et cliquez sur l'image pour afficher les profils d'intensité le long des axes X et Y.

Cliquez sur le bouton « Traiter le signal » dans la barre d'outils près du profil pour transférer le profil vers le panneau « Signaux ». Le profil d'intensité est ajouté au panneau « Signaux », et DataLab bascule vers ce panneau pour afficher le profil.

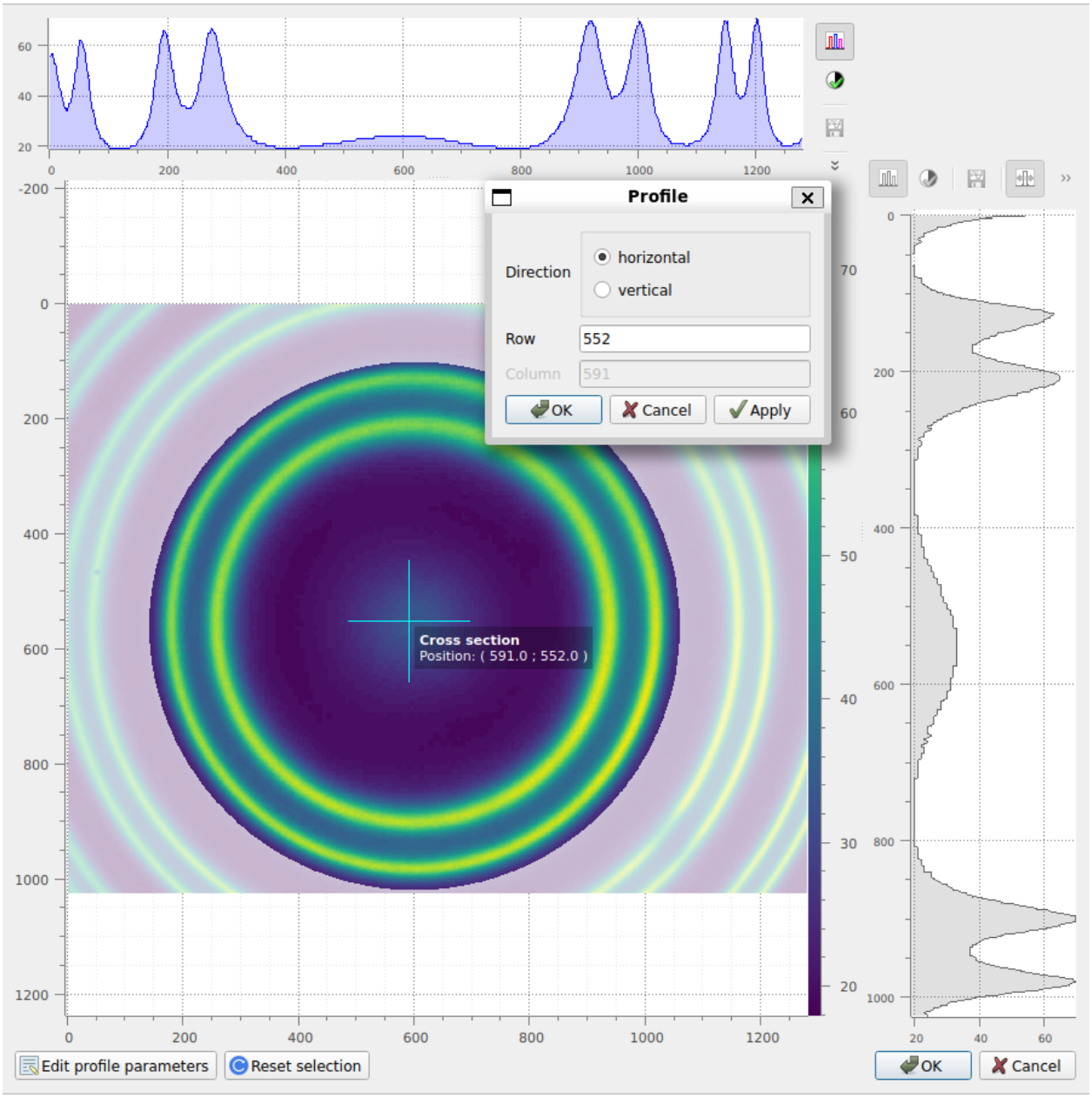


FIG. 71 – La boîte de dialogue « Profil » s'ouvre. Entrez la ligne du profil horizontal (ou la colonne du profil vertical) dans la boîte de dialogue qui s'ouvre. Cliquez sur « OK ».

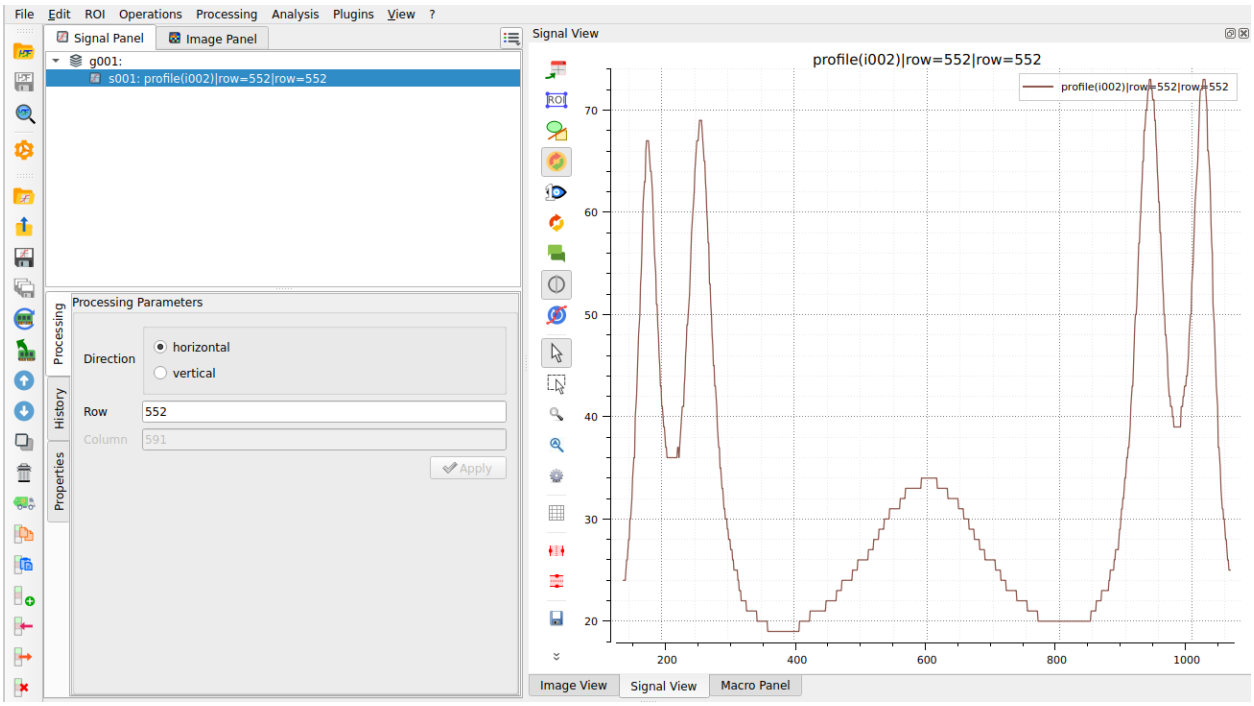


Fig. 72 – Le profil d'intensité est ajouté au panneau « Signaux », et DataLab bascule vers ce panneau pour afficher le profil.

Sauvegarder l'espace de travail

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 d'analyse, 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.

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

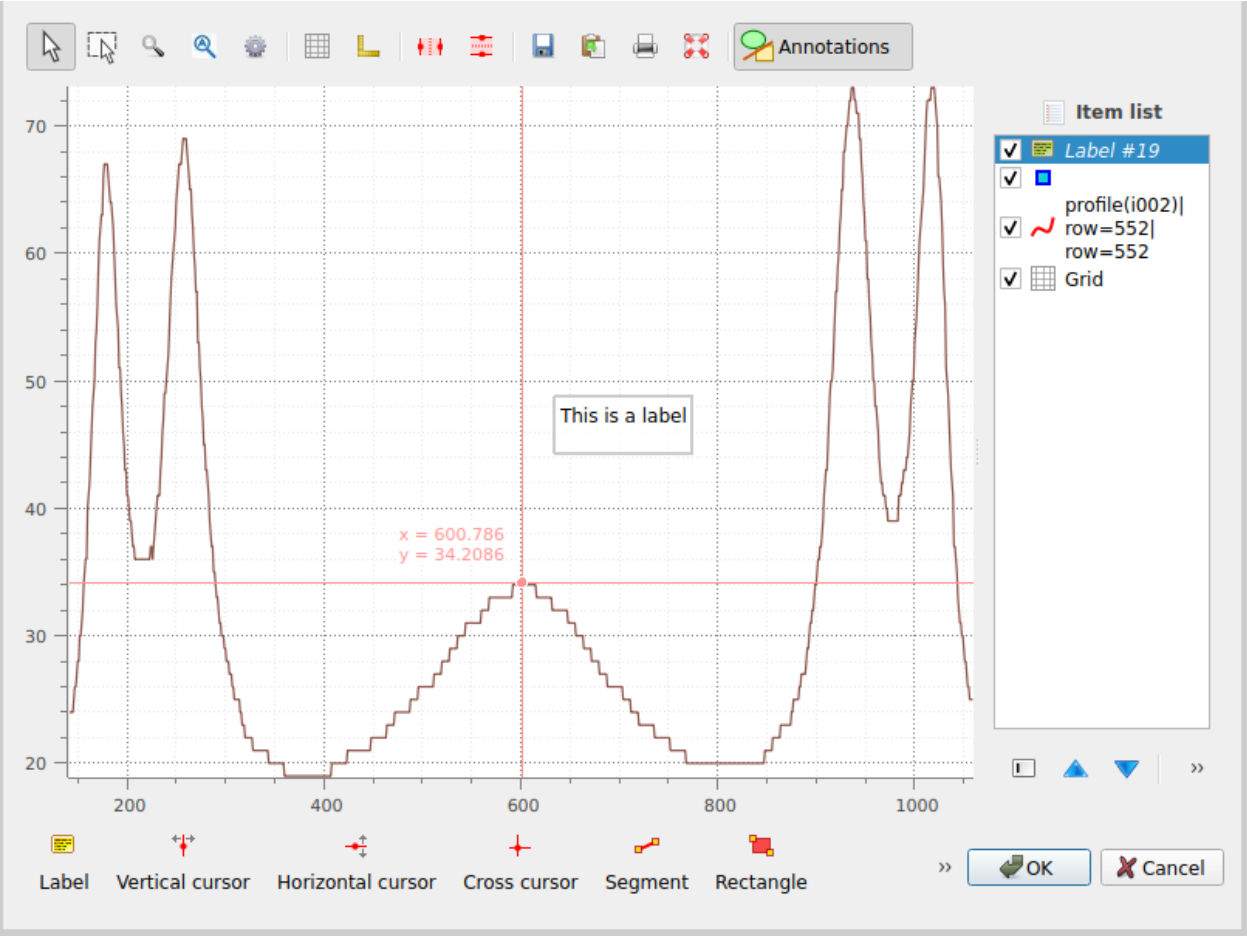


FIG. 73 – Le signal est affiché dans une fenêtre séparée. Ici, nous avons ajouté des curseurs verticaux et une étiquette de texte très intéressante. Comme pour les images, les annotations sont stockées dans les métadonnées du signal, et avec les données du signal lorsque l'espace de travail est sauvegardé. Cliquez sur « OK » pour fermer la fenêtre.

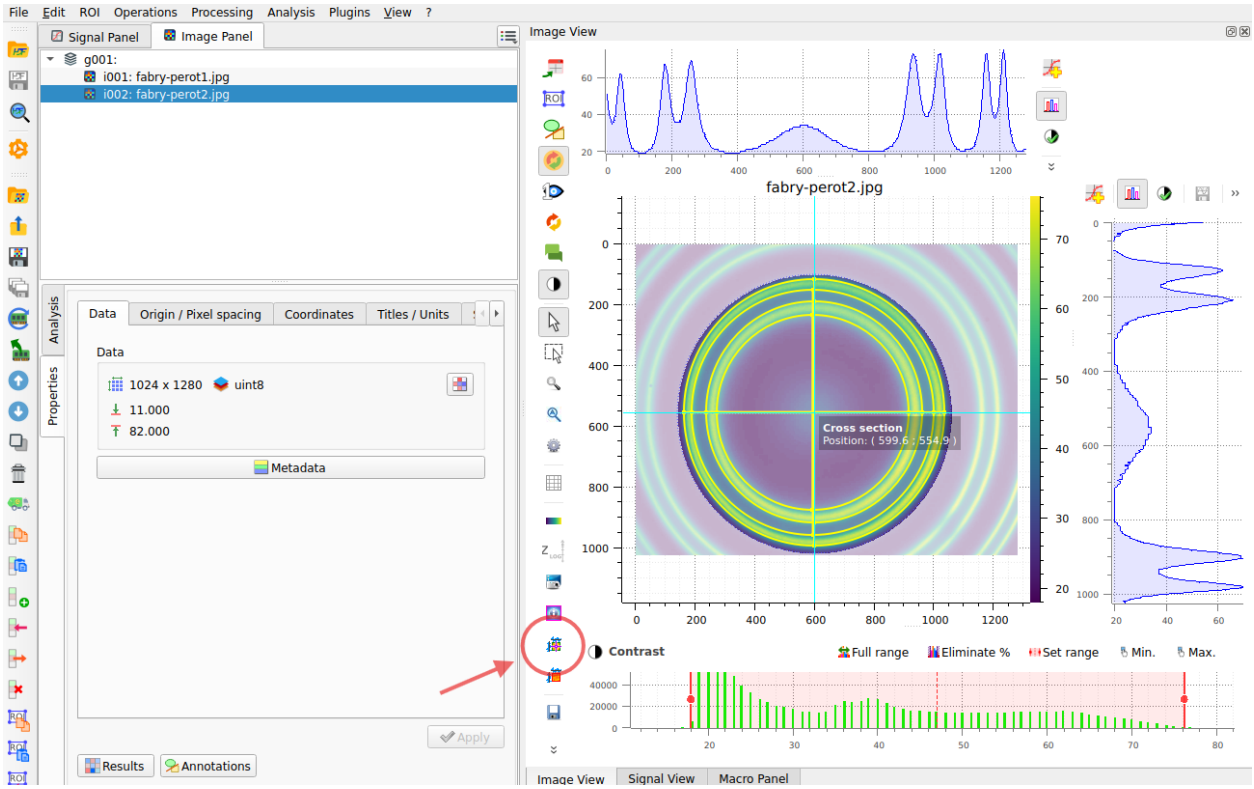


Fig. 74 – L'outil de profil rectiligne dans le panneau de visualisation. Dans le cercle rouge, le bouton pour l'activer. Dans le cercle bleu, le bouton pour transférer le profil vers le panneau « Signaux ».

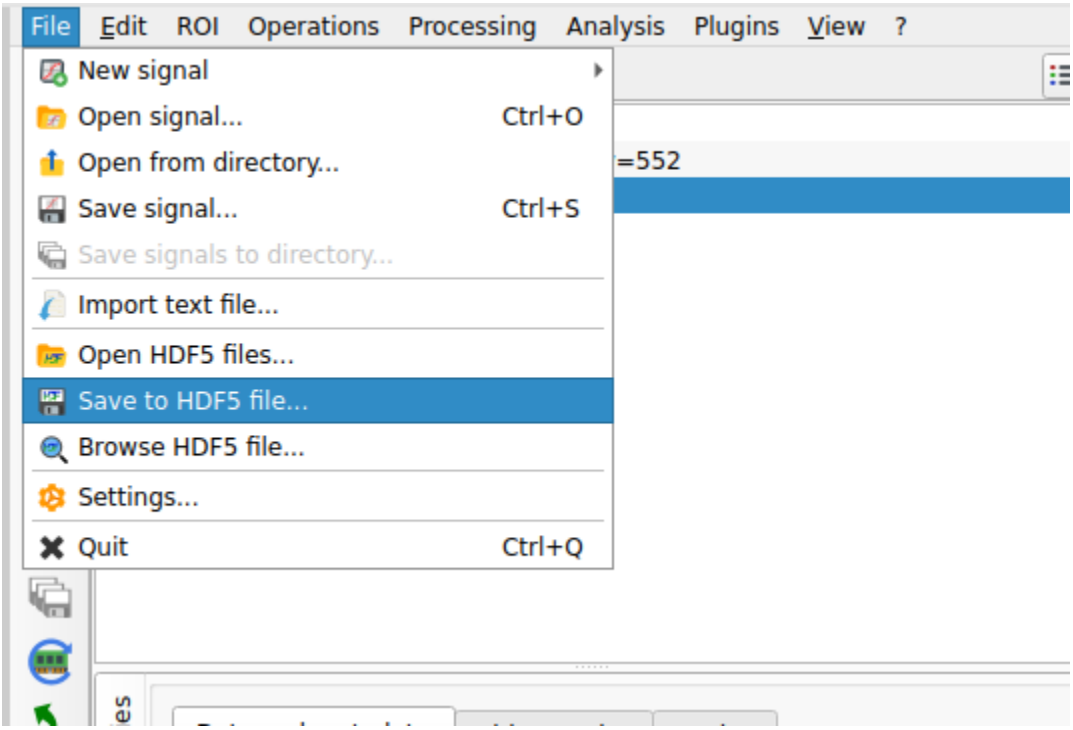


Fig. 75 – Sauvegardez l'espace de travail dans un fichier avec « Fichier > Sauvegarder dans un fichier HDF5… », ou le bouton dans la barre d'outils.

Charger les images

Note : Les images utilisées dans ce tutoriel « TEM00_z_*.jpg » sont disponibles dans le dossier de données du tutoriel du répertoire d'installation de DataLab (<`Répertoire d'installation de DataLab`>`/data/tutorials/`).

Alternativement, elles peuvent être téléchargées depuis la documentation en ligne à l'adresse https://datalab-platform.com.

Tout d'abord, nous ouvrons DataLab et chargeons les images. Lorsque vous travaillez avec plusieurs images, l'approche la plus efficace consiste à utiliser l'option « Ouvrir depuis un répertoire… » du menu « Fichier » (ou cliquer sur le bouton dans la barre d'outils). Cette fonction charge toutes les images d'un répertoire sélectionné en une seule fois.

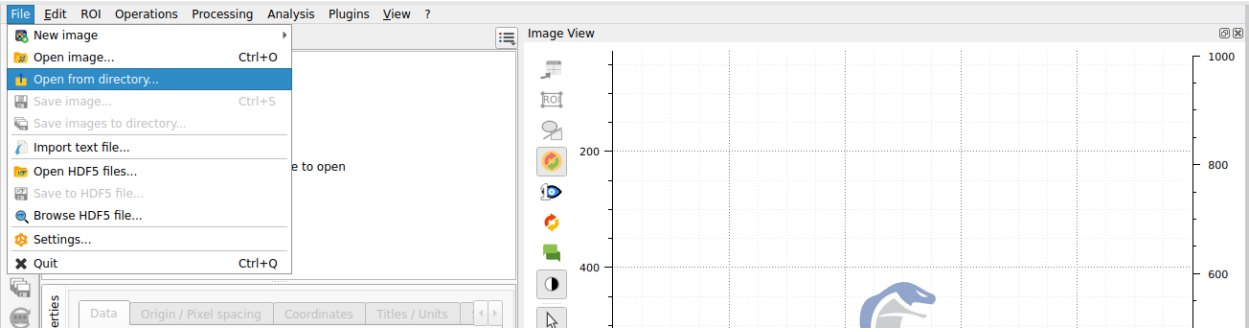


Fig. 76 – Le menu « Fichier > Ouvrir une image… »

Les images sont maintenant chargées dans le panneau « Images ». Vous pouvez zoomer en cliquant avec le bouton droit et en faisant glisser la souris verticalement. Pour déplacer l'image, utilisez le bouton du milieu de la souris tout en faisant glisser.

Note : Pour voir plusieurs images simultanément, sélectionnez le groupe d'images et choisissez « Afficher les images côte à côte » dans le menu « Affichage ».

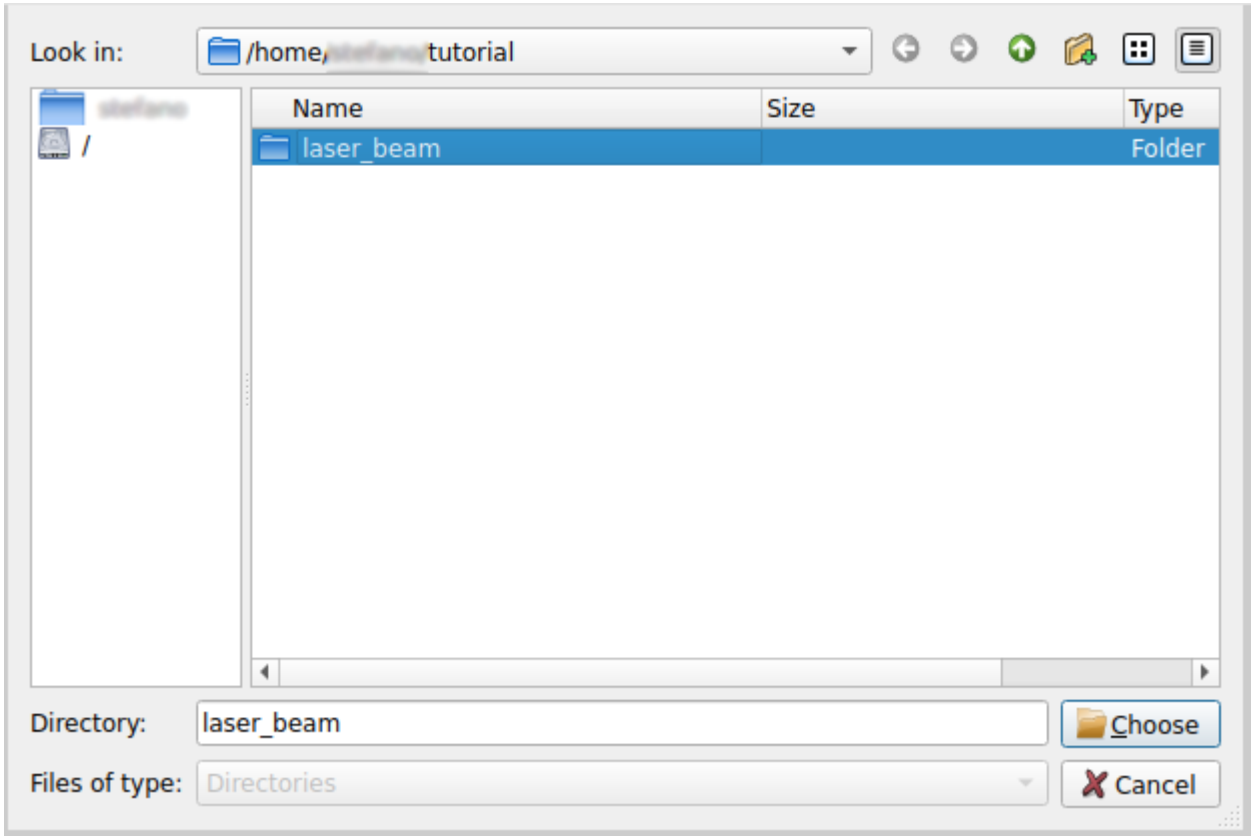


FIG. 77 – Sélectionnez le dossier contenant les images et cliquez sur « Choisir ».

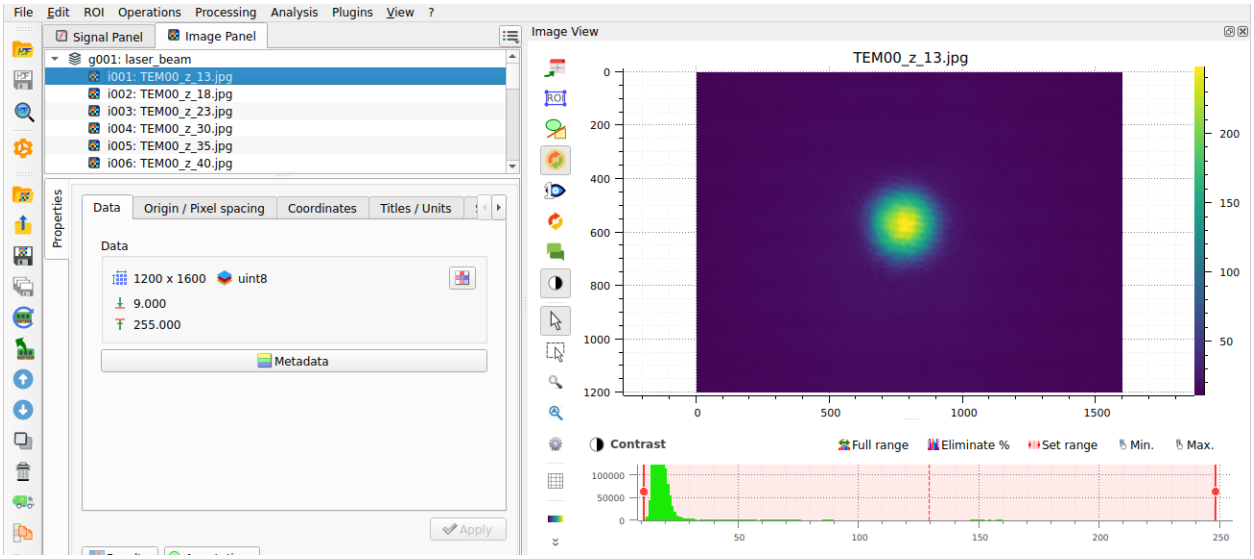


FIG. 78 – Zoomer avec le bouton droit de la souris. Déplacer l'image avec le bouton du milieu de la souris.

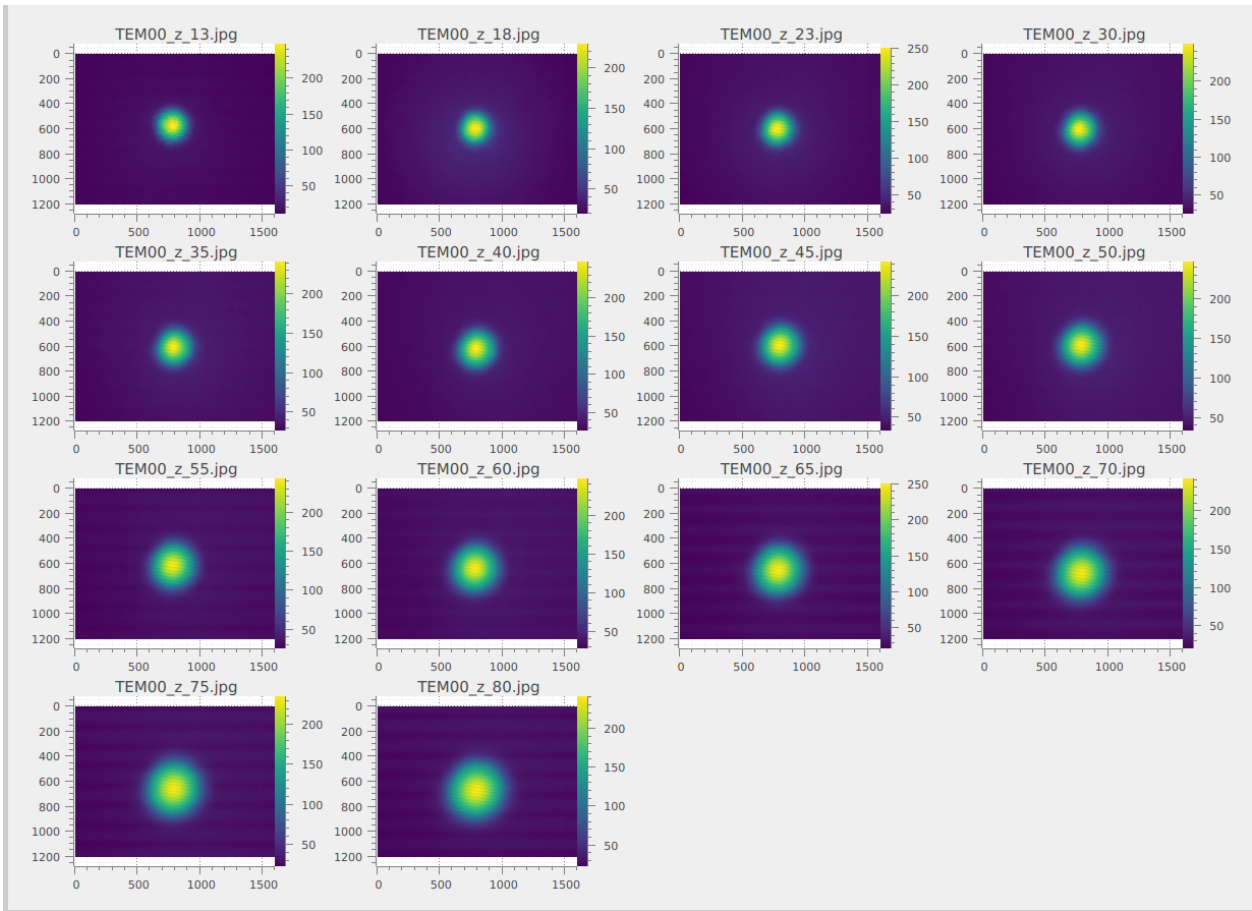
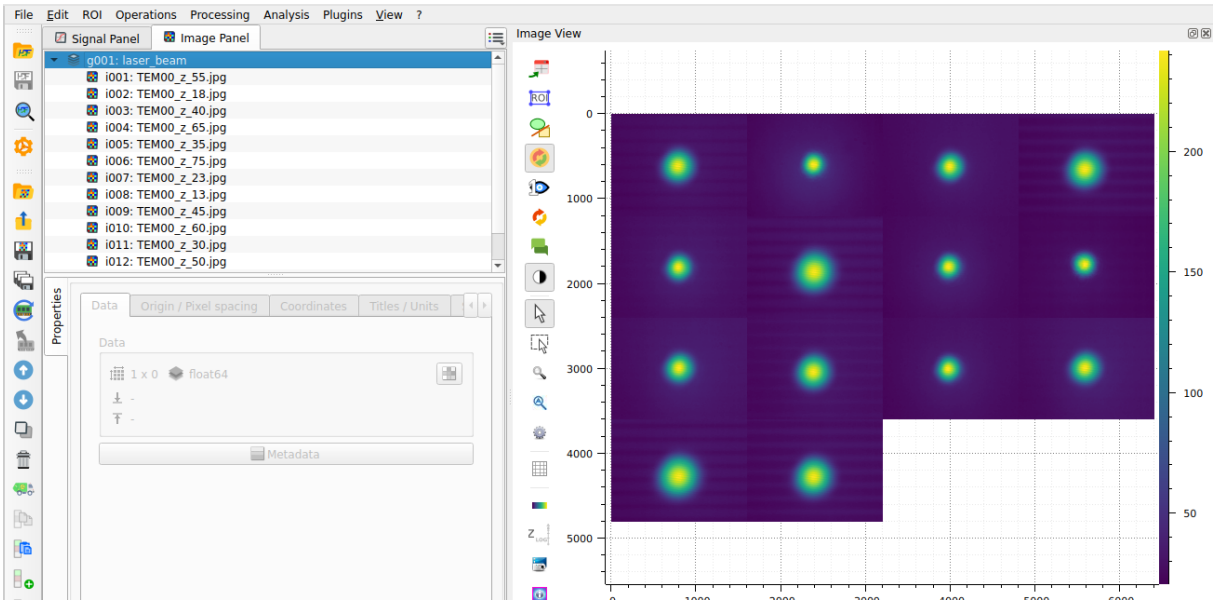


FIG. 79 – Affichage des images côte à côte.

Avertissement : Le menu « Traitement > Géométrie » inclut une option « Distribuer sur une grille » . Cette fonction repositionne les images en appliquant des décalages à leurs coordonnées X et Y, les organisant dans une disposition en grille pour une visualisation côte à côte. Notez que cette opération modifie les coordonnées de l'image.



Images distribuées sur une grille de 4 colonnes.

Pour restaurer les positions d'image d'origine, utilisez l'option « Réinitialiser les positions des images » du menu « Traitement > Géométrie ». Notez que cette opération définit toutes les origines d'image pour qu'elles correspondent à l'origine de la première image, ce qui signifie que toutes les différences initiales dans les origines d'image seront perdues.

Supprimer le bruit de fond

Lorsque nous sélectionnons l'une des images, nous remarquons la présence de bruit de fond, ce qui rend utile l'application d'un seuillage aux images. Plusieurs méthodes sont disponibles pour estimer le niveau de bruit de fond.

Une approche utilise l'outil « Profil rectiligne », qui est fourni par la bibliothèque *PlotPy <https ://github.com/PlotPyStack/plotpy>* que DataLab utilise pour la visualisation de signaux et d'images. Sélectionnez une image dans le panneau « Images », choisissez l'image correspondante dans le panneau de visualisation, et activez l'outil « Profil rectiligne » depuis la barre d'outils verticale sur le côté gauche du panneau de visualisation. Cela révèle que le niveau de bruit de fond est d'environ 30 lsb.

Une autre méthode pour mesurer le bruit de fond, également fournie par *PlotPy <https ://github.com/PlotPyStack/plotpy>*, consiste à utiliser l'outil « Statistiques de l'image » de la barre d'outils verticale sur le côté gauche du panneau de visualisation. Cet outil affiche des informations statistiques pour une région rectangulaire que vous définissez en faisant glisser la souris sur l'image. Cette analyse confirme que le niveau de bruit de fond est d'environ 30 lsb.

Notez que ces outils ne sont pas persistants : les résultats d'analyse disparaissent lorsque vous sélectionnez une autre image, ils sont destinés à fournir un aperçu rapide des données d'image.

Nous pouvons maintenant écrêter l'image à 35 lsb pour supprimer le bruit de fond en utilisant le menu « Traitement > Ajustement de niveau > Écrêtage… ».

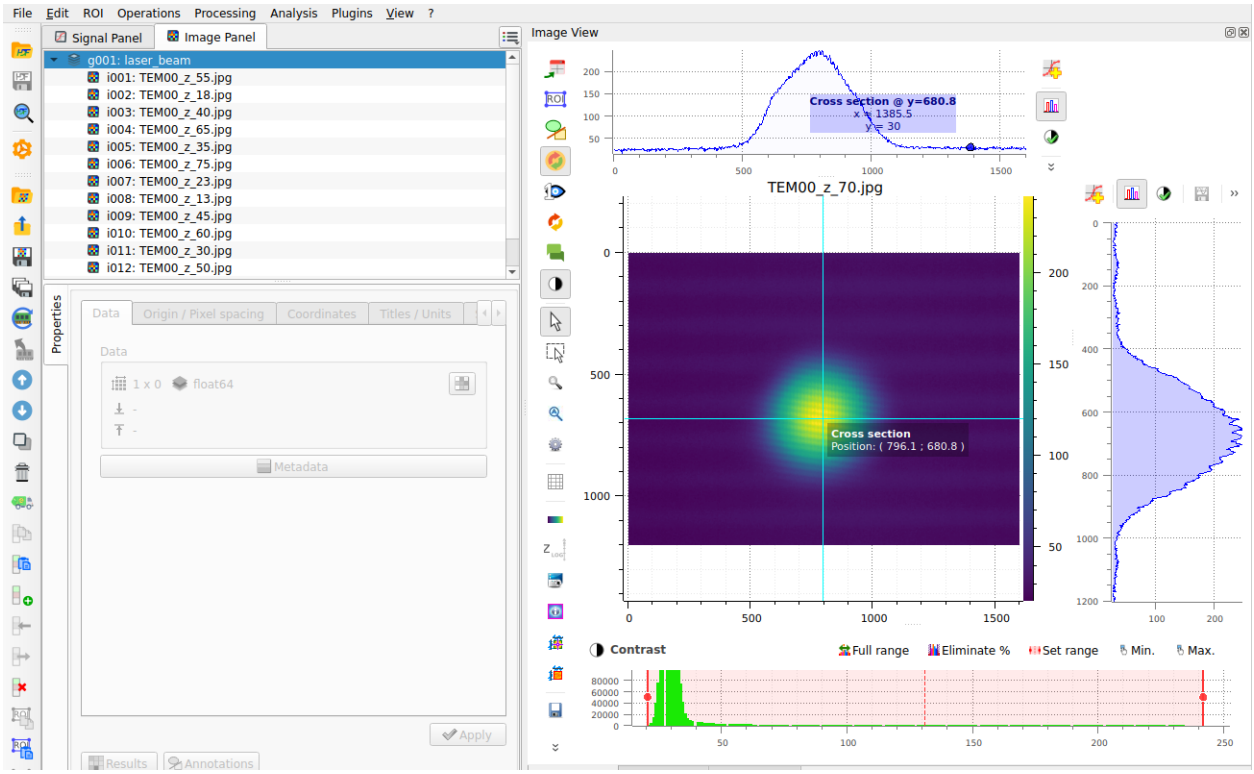


FIG. 80 – Une image du panneau « Images ». Pour afficher le marqueur de courbe, sélectionnez la courbe de profil et cliquez avec le bouton droit pour ouvrir le menu contextuel, puis choisissez « Marqueurs > Lié à l'élément actif ».

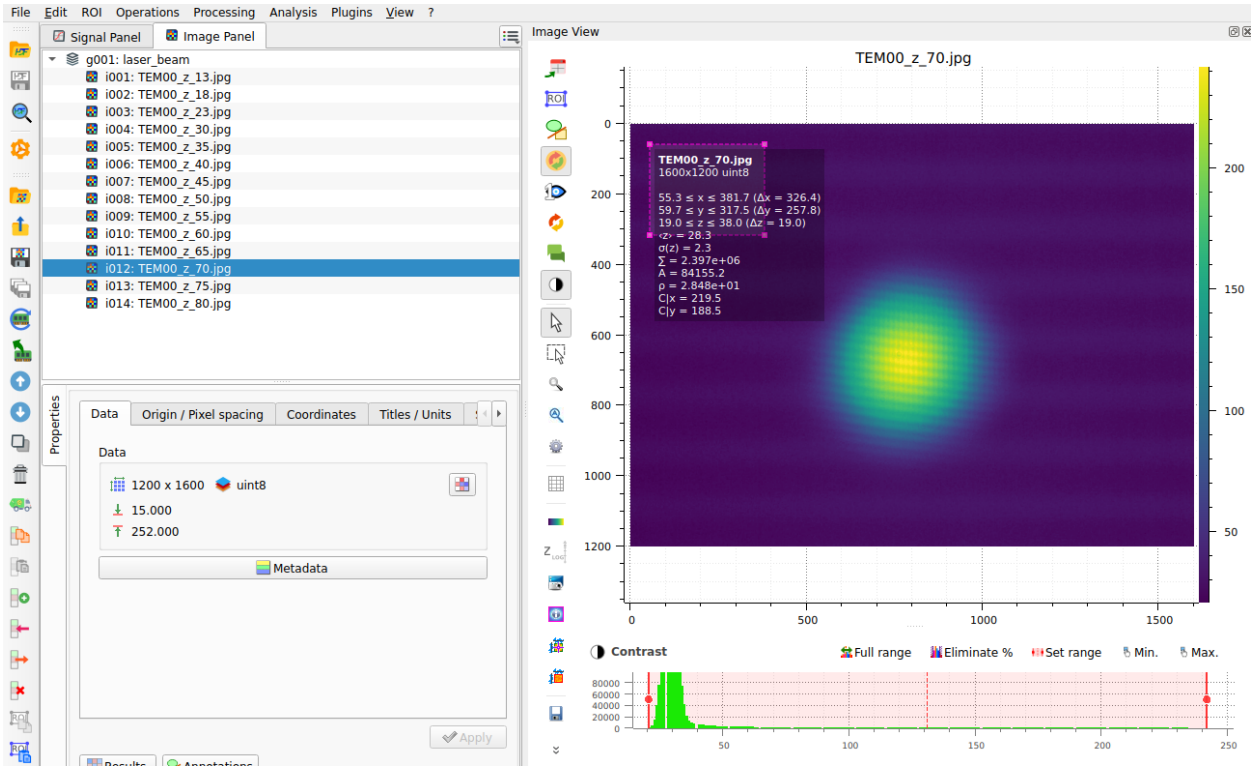


FIG. 81 – L'outil « Statistiques de l'image » dans la barre d'outils verticale.

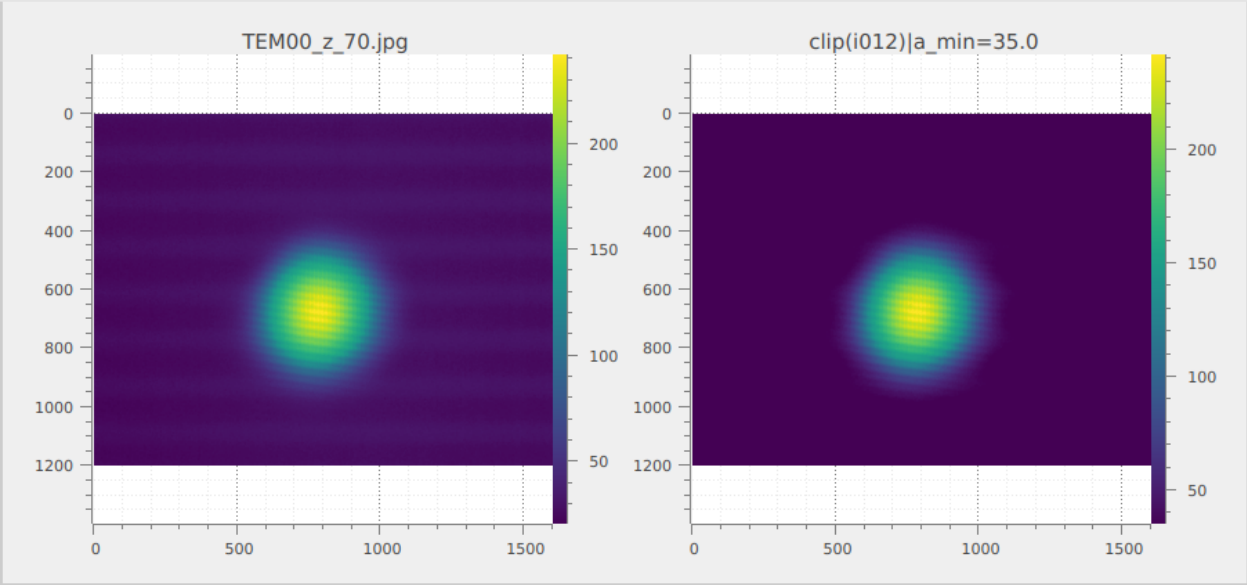


Fig. 82 – Les deux images d'origine et l'image écrêtée sont affichées côte à côte dans la « Vue Image » (en utilisant la fonction « Distribuer sur une grille » vue précédemment).

Mesure de la taille du faisceau

Nous pouvons maintenant calculer le centroïde du faisceau—c'est-à-dire la position de son centre de masse. Pour ce faire, sélectionnez « Analyse > Centroïde » dans le menu.

Ensuite, nous pouvons extraire un profil de ligne le long de l'axe horizontal en utilisant « Opérations > Profils d'intensité > Profil rectiligne ». Définissez la position de la ligne sur la position du centroïde calculée précédemment (c'est-à-dire 668) en utilisant le bouton « Définir les paramètres ». Voir *Mesure de franges de Fabry-Perot* pour plus de détails sur l'extraction de profil d'intensité.

Le profil d'intensité sera affiché dans le panneau « Signaux ». Nous pouvons ensuite ajuster le profil à une fonction gaussienne en utilisant « Traitement > Ajustement > Ajustement gaussien ». Ici, nous avons sélectionné les deux signaux pour comparaison.

Explorons maintenant une autre méthode pour calculer la FWHM. En revenant au signal de profil d'intensité, nous pouvons calculer directement la FWHM en utilisant « Analyse > Largeur à mi-hauteur ». Vous pouvez choisir la méthode d'estimation en fonction des caractéristiques de votre courbe et spécifier éventuellement l'intervalle à considérer pour ce calcul. Une fois terminé, la fenêtre de résultats affichera la valeur FWHM, qui est stockée dans les métadonnées et affichée sur la courbe.

Essayons également une autre méthode pour mesurer la taille du faisceau en revenant à l'image.

Depuis le panneau « Images », nous pouvons extraire le profil d'intensité radial en utilisant « Opérations > Profils d'intensité > Profil radial ». Le profil d'intensité radial peut être calculé autour de la position du centroïde, du centre de l'image ou d'une position définie par l'utilisateur, selon vos données. Pour nos données, qui semblent avoir une symétrie radiale avec un centre pas nécessairement identique au centre de l'image, la meilleure option est la position du centroïde.

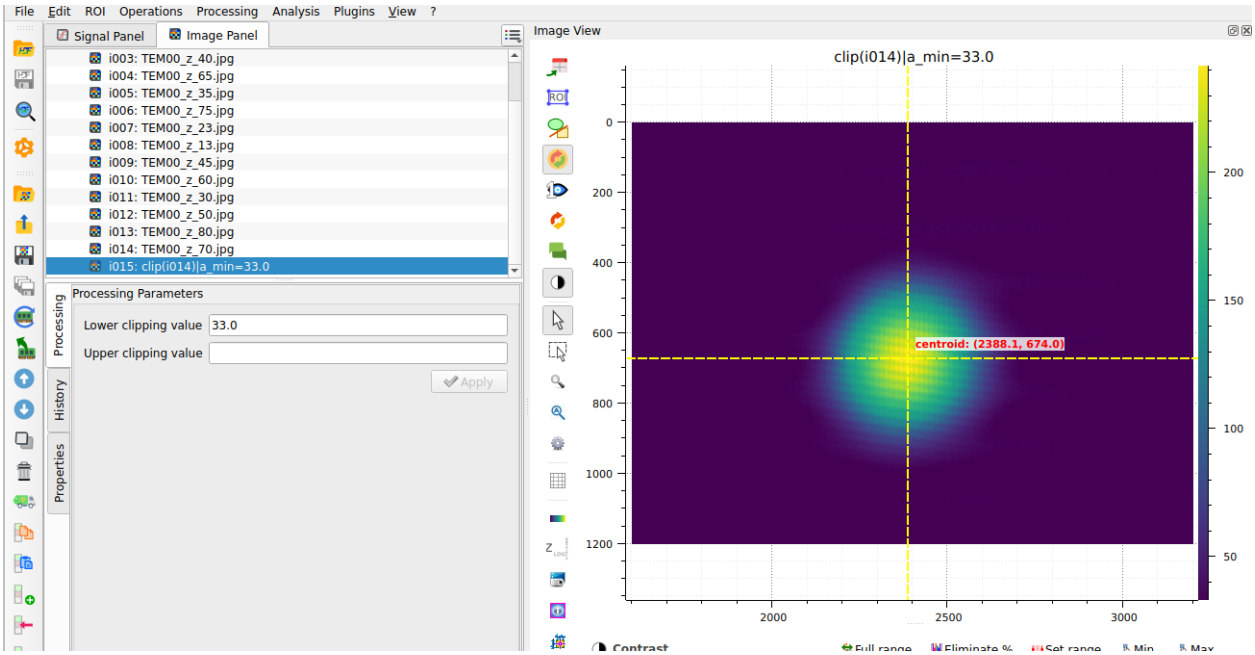


FIG. 83 – La position du centroïde est affichée sur l'image.

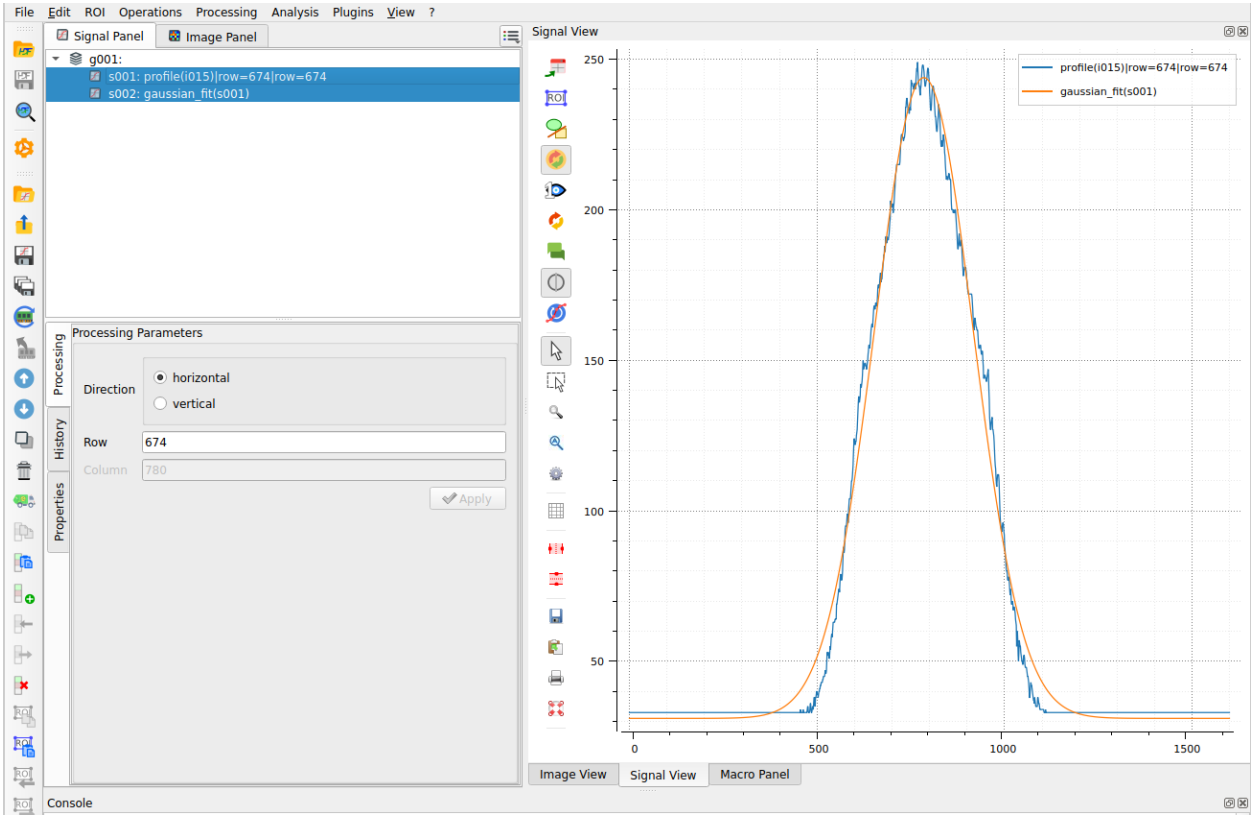


FIG. 84 – Le profil d'intensité ajusté à une fonction gaussienne. Ici, les deux signaux sont sélectionnés.

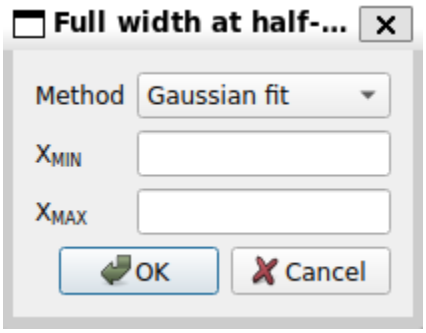


FIG. 85 – La fenêtre contextuelle qui permet de choisir la méthode pour estimer la FWHM.

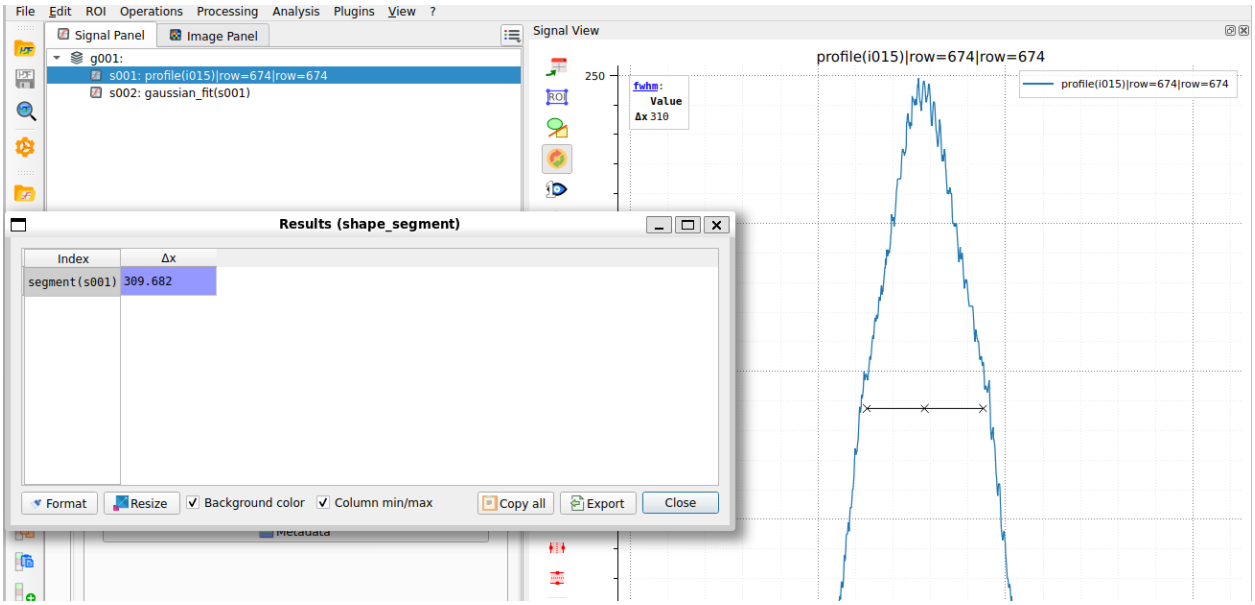


FIG. 86 – Le résultat de la FWHM affiché dans la fenêtre contextuelle et sur la courbe.

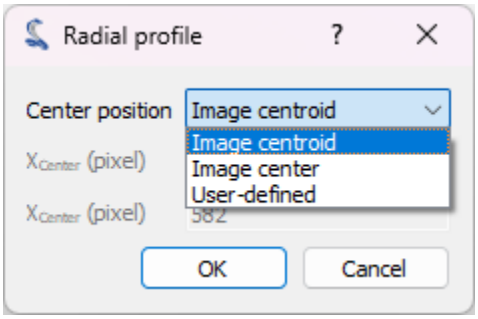


FIG. 87 – Les options disponibles pour le calcul du profil radial.

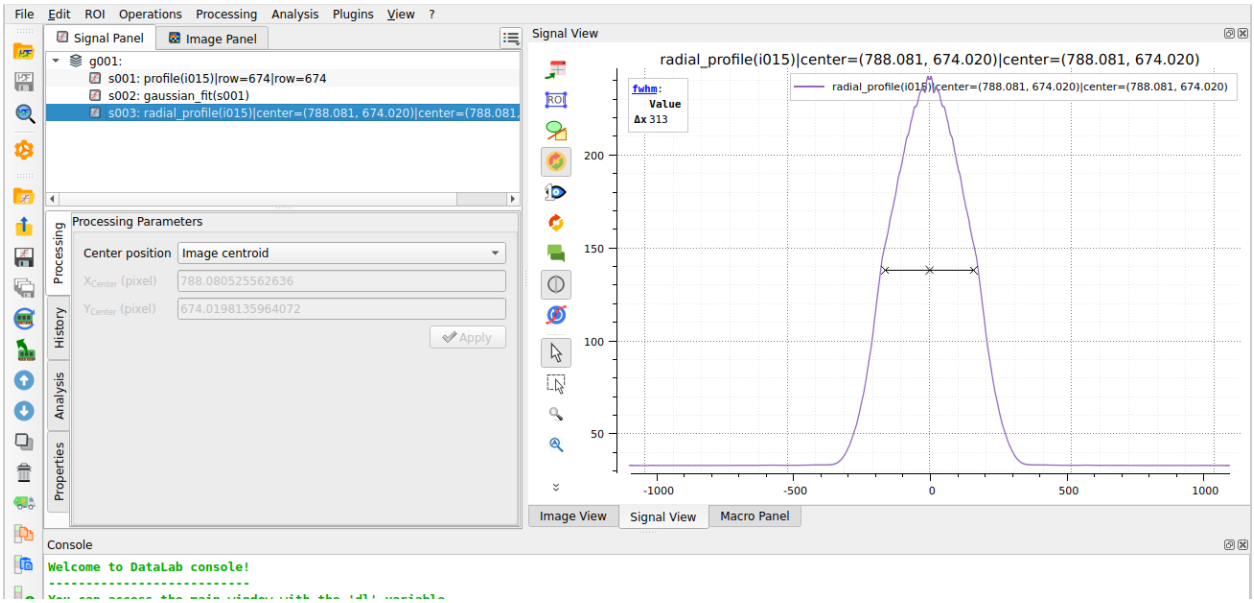


FIG. 88 – Le profil d'intensité radial affiché dans le panneau « Signaux ». Il est plus lisse que le profil de ligne, car il est calculé à partir d'un plus grand nombre de pixels, ce qui permet de moyenner le bruit.

Appliquer ces opérations à toutes les images

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

Pour commencer, nettoyez le panneau « Signaux » en utilisant « Édition > Tout supprimer » ou le bouton dans la barre d'outils. Supprimez également les résultats intermédiaires du panneau « Images » en sélectionnant les images créées lors du prototypage et en les supprimant individuellement en utilisant « Édition > Supprimer » ou le bouton .

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

Appliquez l'opération d'écrêtage à toutes les images, puis extrayez les profils d'intensité radiaux pour toutes les images (après avoir sélectionné l'ensemble du groupe « g002 »—il devrait être automatiquement sélectionné si vous aviez « g001 » sélectionné avant d'appliquer le seuil).

Nous pouvons maintenant calculer la FWHM pour tous les profils d'intensité radiaux. La boîte de dialogue « Résultats » affichera les valeurs FWHM pour tous les profils.

Note : Pour afficher à nouveau les résultats d'analyse, sélectionnez « Afficher les résultats » dans le menu « Analyse », ou cliquez sur le bouton « Afficher les résultats » en dessous de la liste des images :

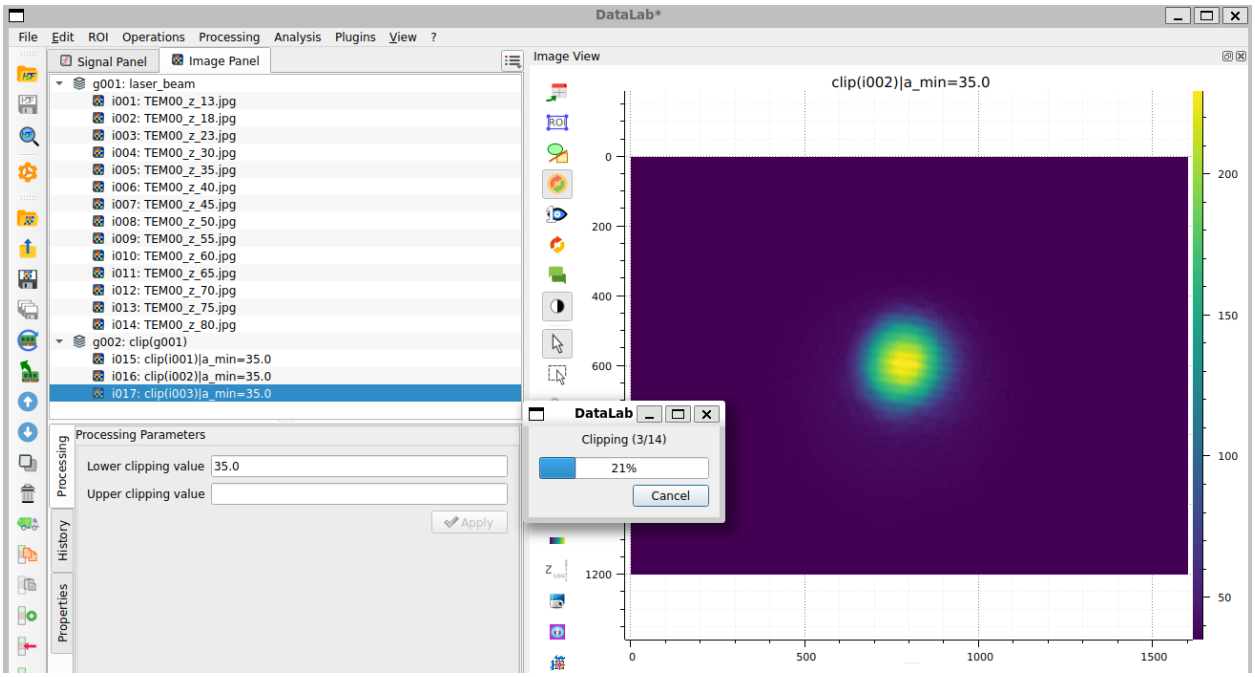


FIG. 89 – L'écrêtage s'applique à toutes les images du groupe.

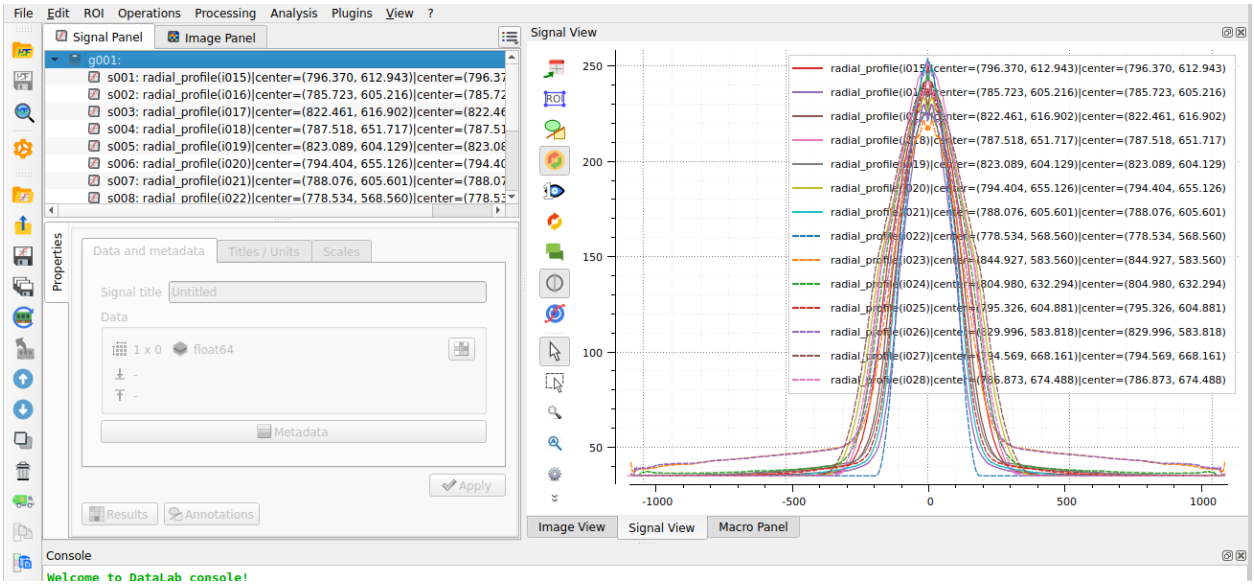
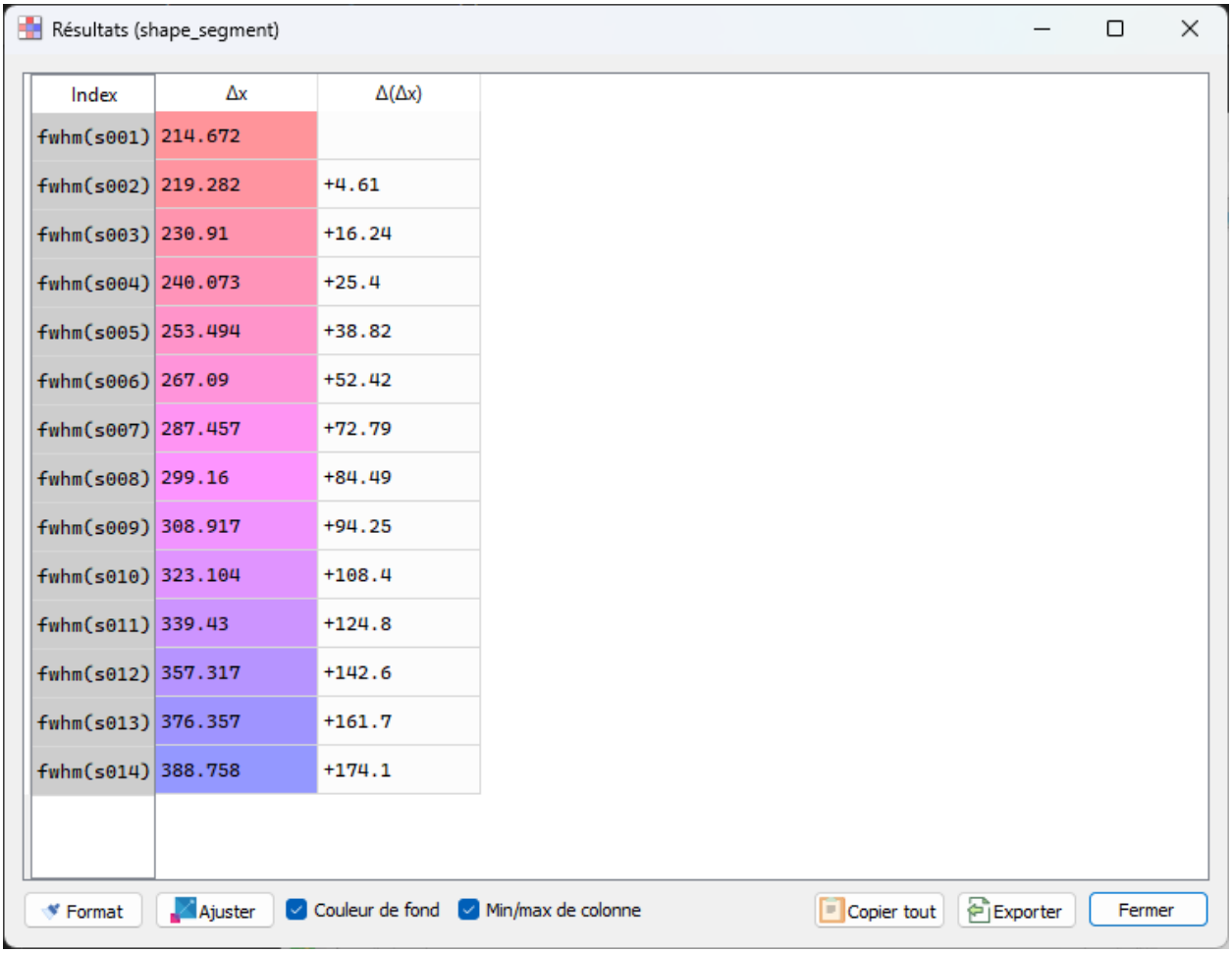


FIG. 90 – Le panneau « Signaux » contient maintenant tous les profils d'intensité radiaux.



Index	Δx	Δ(Δx)
fwhm(s001)	214.672	
fwhm(s002)	219.282	+4.61
fwhm(s003)	230.91	+16.24
fwhm(s004)	240.073	+25.4
fwhm(s005)	253.494	+38.82
fwhm(s006)	267.09	+52.42
fwhm(s007)	287.457	+72.79
fwhm(s008)	299.16	+84.49
fwhm(s009)	308.917	+94.25
fwhm(s010)	323.104	+108.4
fwhm(s011)	339.43	+124.8
fwhm(s012)	357.317	+142.6
fwhm(s013)	376.357	+161.7
fwhm(s014)	388.758	+174.1

FIG. 91 – La boîte de dialogue « Résultats » affiche les valeurs FWHM pour tous les profils.

Enfin, nous pouvons tracer la taille du faisceau en fonction de la position le long de l'axe de propagation en utilisant la fonction « Tracer les résultats » du menu « Analyse ». Cette fonction vous permet de tracer des ensembles de données de résultats en sélectionnant les axes x et y parmi les colonnes de résultats disponibles. Ici, nous allons tracer les valeurs FWHM (L) en fonction de l'index de l'image (*Indices*).

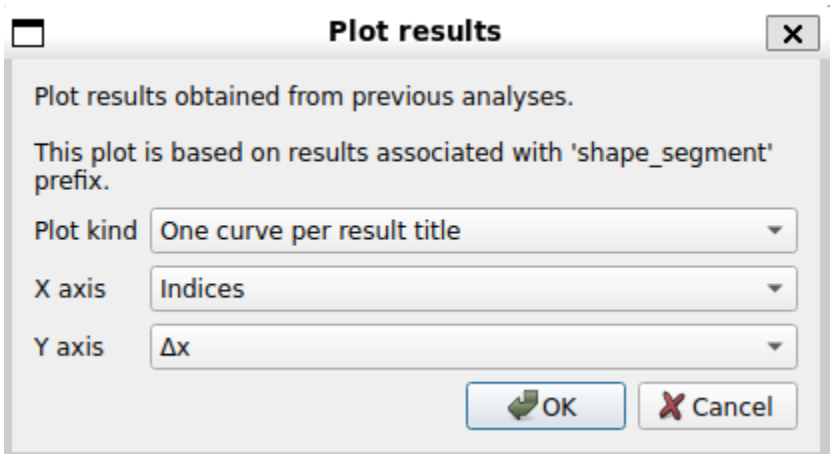


FIG. 92 – La fonction « Tracer les résultats » dans le menu « Analyse ».

Nous pouvons également étalonner les axes X et Y en utilisant « Traitement > Étalonnage linéaire ». Ici, nous définissons l'axe X sur la position en mm (en entrant le titre et l'unité dans le groupe « Propriétés ») en utilisant la formule : $X[\text{mm}] = 0.5 \cdot i + 10$ où i est l'index de l'image.

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 d'analyse, les paramètres de 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.

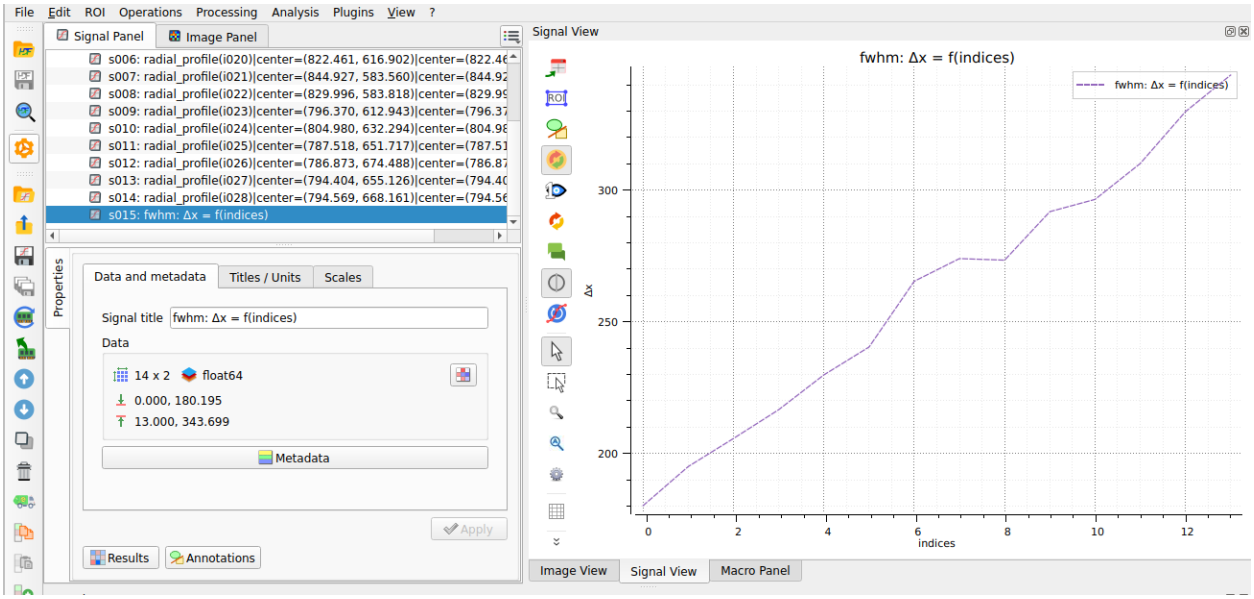


FIG. 93 – La taille du faisceau en fonction de la position le long de l'axe de propagation (la position est en unités arbitraires—l'index de l'image).

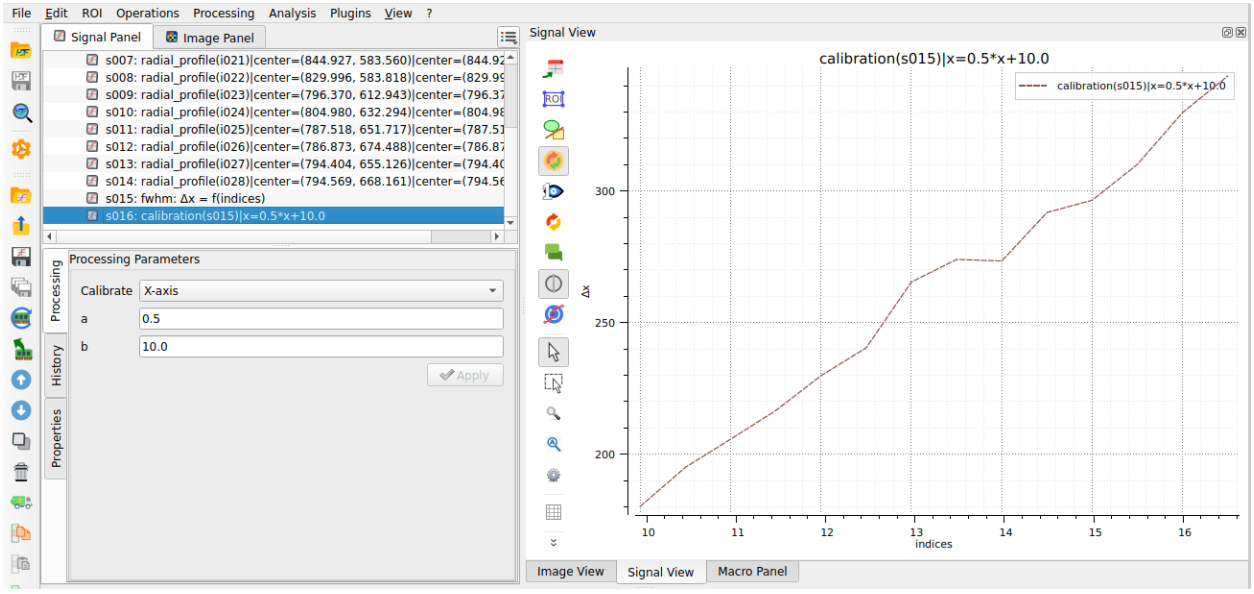


FIG. 94 – La taille du faisceau étalonné en fonction de la position le long de l'axe de propagation (la position est maintenant en mm).

Prototypage d'une chaîne de traitement personnalisée

Pour votre application spécifique, il est possible que vous ayez besoin d'une fonction qui n'est pas disponible dans DataLab par défaut. Dans ce cas, vous pouvez définir votre propre fonction et l'intégrer dans DataLab pour analyser vos données de manière pratique. DataLab a été conçu pour être flexible et extensible, et le but de ce tutoriel est de vous montrer comment faire cela.

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 allons utiliser 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 moyenne et de détection de contours. Ce filtre va moyenner les valeurs des 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 contour ou du bruit. Une fois que vous aurez compris comment implémenter un traitement personnalisé dans DataLab, vous pourrez définir vos propres fonctions adaptées à vos besoins spécifiques.

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)
```

Configurer l'environnement de test

Pour tester notre fonction de traitement, nous allons utiliser une image générée à partir d'un exemple de plugin DataLab (*plugins/examples/datalab_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* pour comprendre comment le faire).

Nous réorganisons ensuite la disposition de la fenêtre de DataLab pour avoir un environnement confortable pour développer et tester notre fonction : nous réorganisons la disposition de la fenêtre de DataLab pour avoir le « Panneau Image » à gauche et le « Gestionnaire de Macros » à droite. Pour ce faire, vous pouvez simplement faire glisser et déposer les panneaux à la position souhaitée. Si le « Gestionnaire de Macros » n'est pas visible, vous pouvez l'activer depuis le menu « Affichage > Panneaux > Gestionnaire de Macros ».

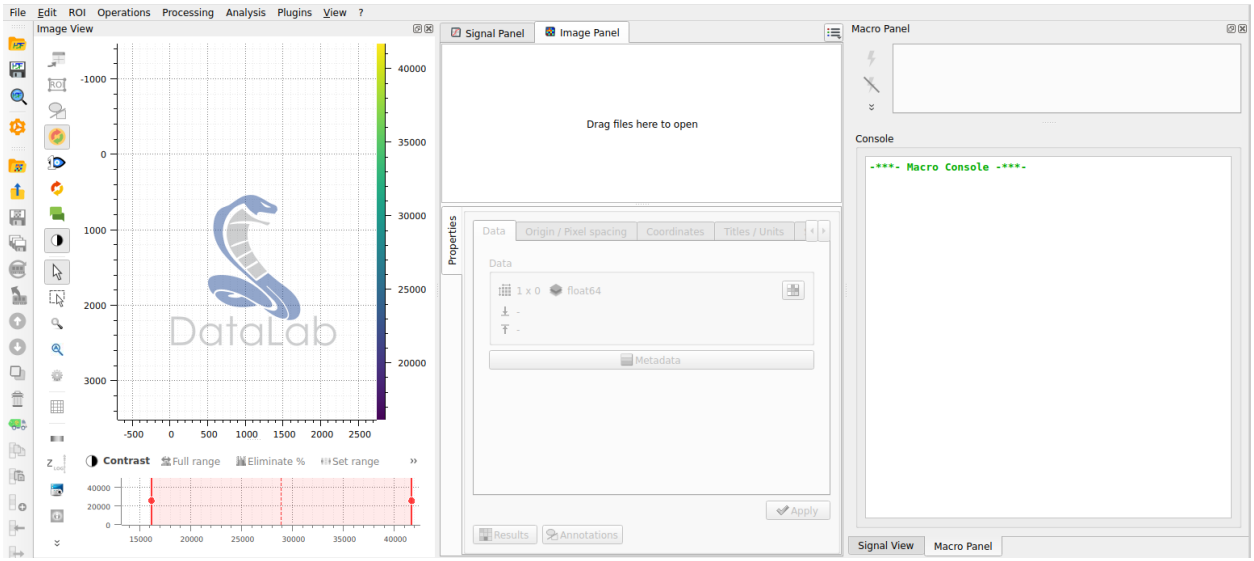


Fig. 95 – La disposition de la fenêtre de DataLab réorganisée pour avoir le « Panneau Image » à gauche et le « Gestionnaire de Macros » à droite.

Nous pouvons maintenant générer une image de test en utilisant le plugin que nous avons installé précédemment. Pour ce faire, nous cliquons sur le menu « Plugins > Extract blobs (example) > Generate test image ». La fonction que nous développons est assez lente, nous choisissons donc une taille limitée pour l'image (par exemple 512x512 pixels).

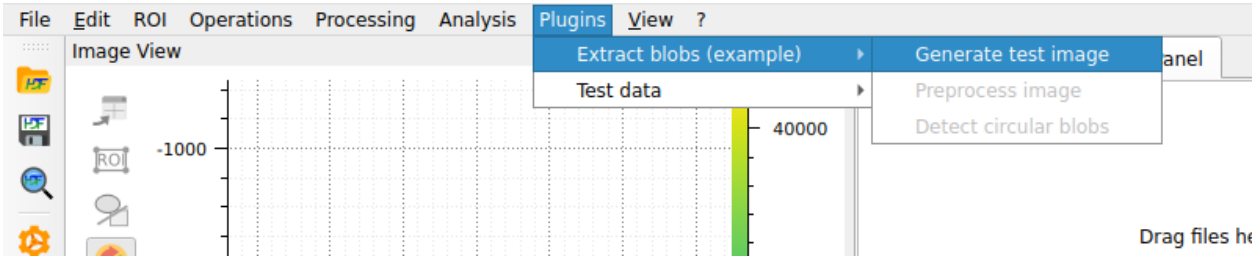


Fig. 96 – Nous générons une nouvelle image en utilisant le menu « Plugins > Extract blobs (example) > Generate test image ».

Le plugin génère une image de test et l'ajoute au « Panneau Image ».

Créer une macro-commande

Une macro est un code qui peut être exécuté directement dans DataLab pour automatiser des tâches. Les macros sont écrites en Python et utilisent l'API de DataLab pour interagir avec DataLab.

De plus, les macros ont les caractéristiques importantes suivantes :

- 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 nous 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.

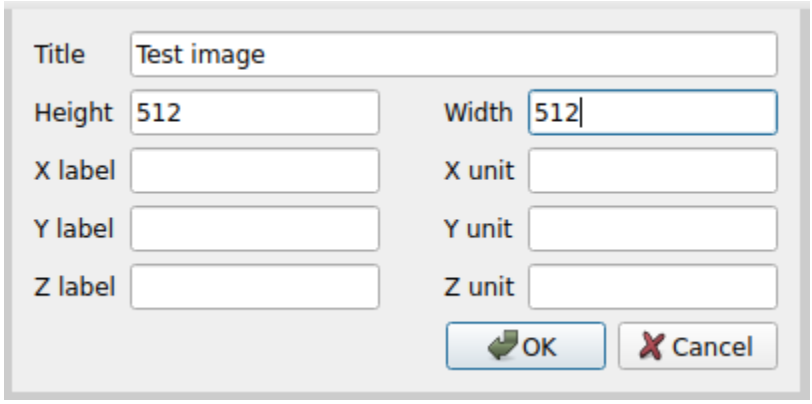


FIG. 97 – Nous sélectionnons la taille de l'image et cliquons sur « OK ».

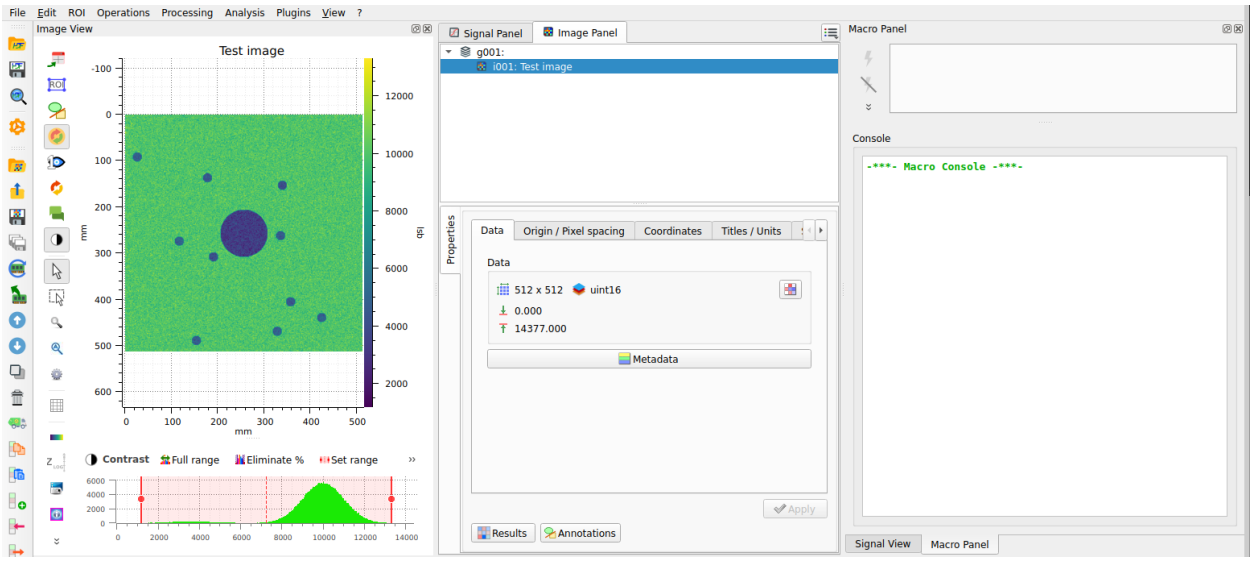


FIG. 98 – L'image générée dans le « Panneau Image ».

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. Une liste des modules disponibles dans l'environnement Python de votre DataLab est disponible dans la boîte de dialogue « ? > Installation et configuration > Configuration de l'installation ».

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.

Pour éditer les macros, nous utilisons le « Gestionnaire de Macros » disponible dans DataLab. Ce panneau est divisé en deux parties : la partie supérieure est l'éditeur de macros, où nous pouvons écrire et éditer le code de la macro ; la partie inférieure est la console, où nous pouvons voir la sortie de la macro. Si l'éditeur de macros est trop petit pour afficher tous les boutons de la barre d'outils des macros, certains d'entre eux sont cachés et peuvent être accessibles en cliquant sur le bouton de dépliage. Alternativement, vous pouvez redimensionner l'éditeur de macros en faisant glisser le séparateur entre l'éditeur et la console.

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 ».

Dans ce code, nous pouvons remarquer la présence de la classe `RemoteProxy` qui est utilisée pour communiquer avec DataLab. Cette classe fait partie de l'API de DataLab, et est utilisée pour accéder aux objets et fonctions de DataLab à partir d'une macro-commande, d'un plugin ou d'un IDE externe. Vous pouvez trouver la documentation de l'API de DataLab dans la section *API*.

Nous pouvons supprimer ce code et le remplacer par notre propre code :

```
# Import the necessary modules
import numpy as np
import scipy.ndimage as spi
from datalab.control.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]
```

(suite sur la page suivante)

(suite de la page précédente)

```
    differences = np.abs(values - central_pixel)
    weights = np.exp(-differences / np.mean(differences))
    return np.average(values, weights=weights)

# Initialize the proxy to DataLab
proxy = RemoteProxy()

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

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

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

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

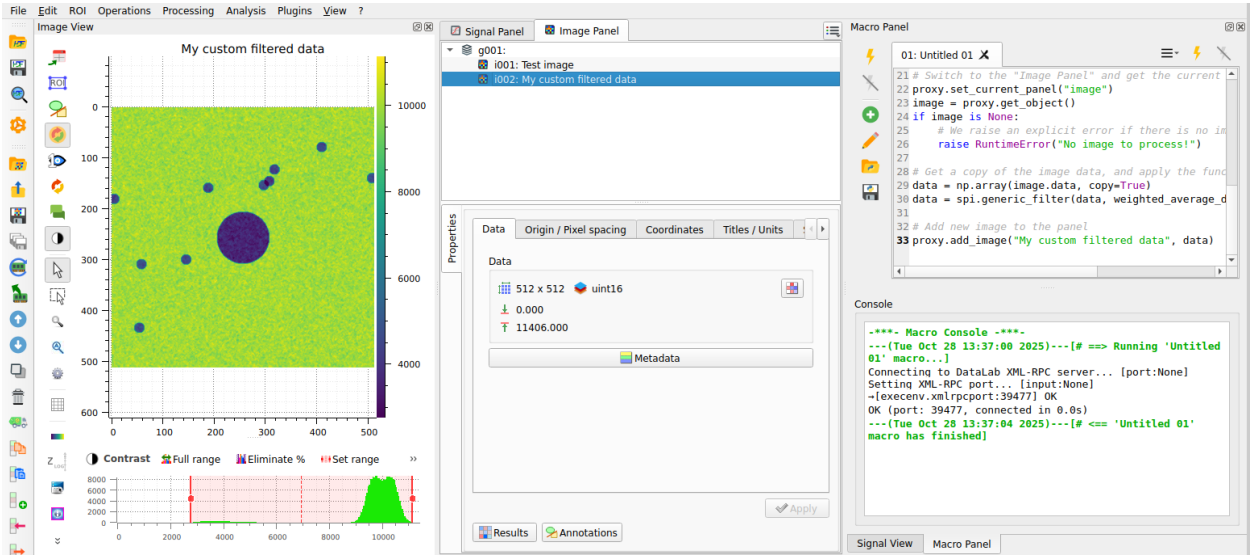


FIG. 99 – Lorsque la macro-commande est terminée, nous pouvons voir la nouvelle image dans le « Panneau Image ». Notre filtre a été appliqué à l'image, et nous pouvons voir que le bruit a été réduit.

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 souhaitez ou devez garder DataLab et Spyder dans des environnements Python séparés, vous pouvez installer le package Sigima (`sigima`) qui fournit un client distant simple pour DataLab. Pour l'installer, il suffit d'exécuter :

  pip install sigima

  Ou vous pouvez également installer le paquet Python DataLab (`datalab`) 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 datalab-platform
```

— 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 `datalab`).

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

```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

@author: p.raybaut
"""

# %% Importing necessary modules

import numpy as np
import scipy.ndimage as spi
from sigima.client 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
```

(suite sur la page suivante)

(suite de la page précédente)

```
    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)
```

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

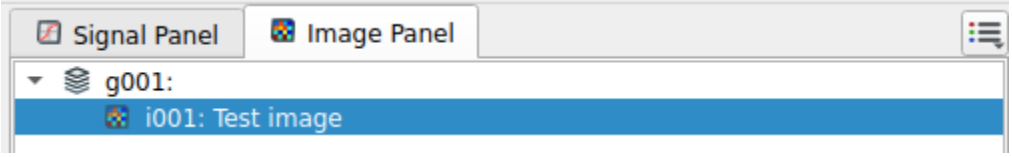


Fig. 100 – La première image dans le « Panneau Image » sélectionnée.

Ensuite, nous exécutons le script dans Spyder (ou votre éditeur préféré), étape par étape (en utilisant les cellules définies), et nous pouvons voir le résultat dans DataLab.

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.

Il existe une autre application pour laquelle l'usage d'un script externe interfacé avec DataLab est utile : vous pouvez imaginer laisser votre script externe acquérir les données et les ajouter automatiquement à DataLab, par exemple dans une boucle d'acquisition. De cette façon, vous pouvez améliorer votre flux de travail de mesure avec la possibilité de visualiser et d'analyser les données directement après leur acquisition.

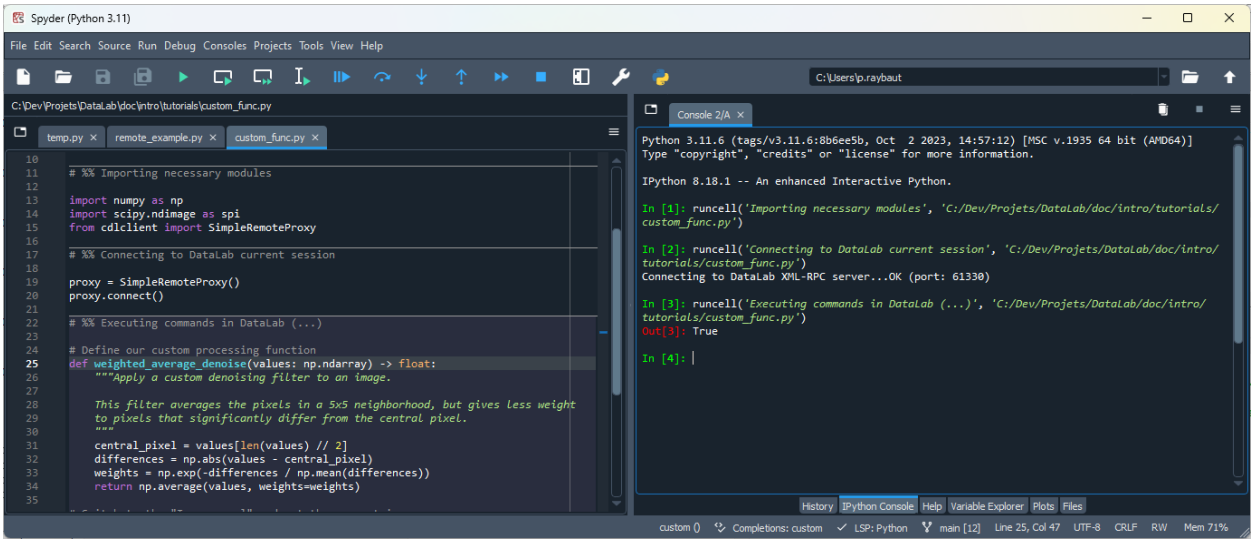


FIG. 101 – L'environnement de l'éditeur Spyder avec le script de traitement personnalisé.

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).

La procédure est la même que pour Spyder ou tout autre IDE comme Visual Studio Code, par exemple.

Création d'un plugin

Maintenant que nous avons un prototype fonctionnel de notre chaîne de traitement, nous pouvons choisir de 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*.

Outre l'intégration de la fonctionnalité dans l'interface graphique de DataLab, ce qui est plus pratique pour l'utilisateur, l'avantage de créer un plugin est que nous pouvons bénéficier 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 que nous devons apporter à notre code est que nous devons définir une fonction qui opérera sur les objets d'image natifs de DataLab (`sigima.objects.ImageObj`), au lieu d'opérer sur des tableaux NumPy. Nous devons donc trouver un moyen d'appeler notre fonction personnalisée `weighted_average_denoise` avec un `sigima.objects.ImageObj` en entrée et en sortie. Pour éviter d'écrire beaucoup de code standard, nous pouvons utiliser le wrapper de fonction fourni par DataLab : `sigima.proc.image.Wrap1to1Func`.

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

DataLab custom function example

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

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

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

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

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

Connecting to DataLab current session:

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

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

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

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

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

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

FIG. 102 – Un code interfacé avec DataLab écrit dans un notebook Jupyter.

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
==============================

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/advanced/plugins.html>`_).
"""

import numpy as np
import scipy.ndimage as spi
import sigima.proc.image as sipi

import datalab.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(datalab.plugins.PluginBase):
    """DataLab Custom Filters Plugin"""

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

(suite sur la page suivante)

(suite de la page précédente)

```
    def create_actions(self) -> None:
        """Create actions"""
        acth = self.imagepanel.acthandler
        proc = self.imagepanel.processor
        with acth.new_menu(self.PLUGIN_INFO.name):
            for name, func in (("Weighted average denoise", weighted_average_denoise),):
                # Wrap function to handle ``ImageObj`` objects instead of NumPy arrays
                wrapped_func = sipi.Wrap1to1Func(func)
                acth.new_action(
                    name,
                    triggered=lambda: proc.compute_1_to_1(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 : vous pouvez le trouver sous « Fichiers > Paramètres » (voir la section *Plugins* pour plus de détails, ou le *Détection de taches sur une image* pour un exemple). Nous redémarrons ensuite DataLab et nous pouvons voir que le plugin a été chargé.

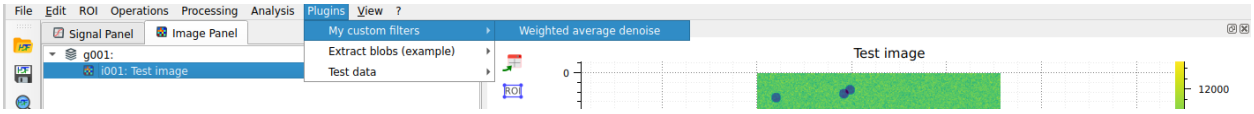


Fig. 103 – Au redémarrage, nous pouvons voir que le plugin a été chargé.

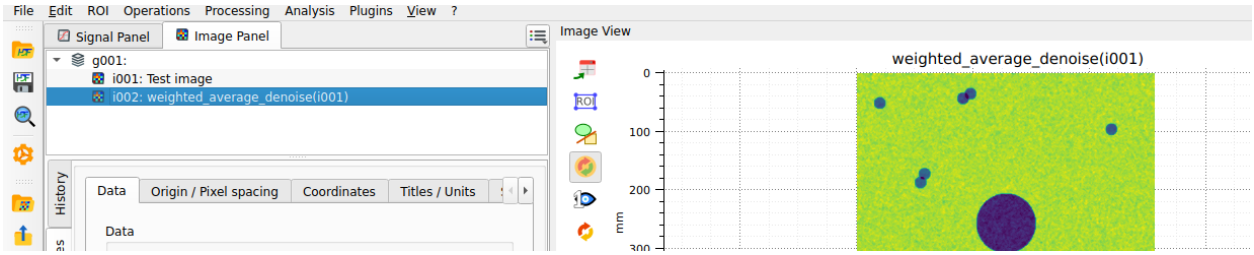
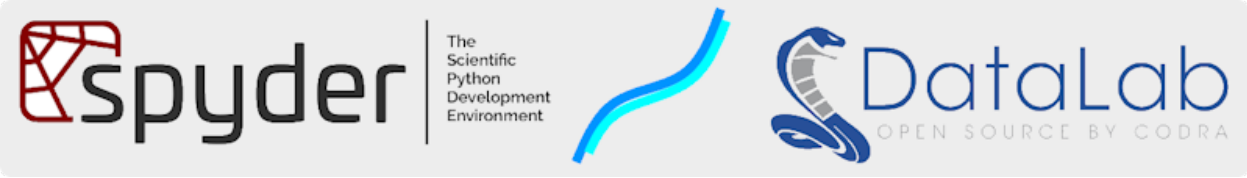


Fig. 104 – Nous générons à nouveau notre image de test en utilisant (voir les premières étapes du tutoriel), et nous la traitons en utilisant le plugin : « Plugins > My custom filters > Weighted average denoise ».

DataLab et Spyder : un duo gagnant

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

L'exemple est simple, mais il démontre les concepts essentiels 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 une plateforme d'analyse de données polyvalente qui peut effectuer une large gamme de tâches, de la simple visualisation de données à l'analyse et au traitement complexes de données. 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 des données avec ces algorithmes.

Concepts de base

Dans le contexte de vos travaux de recherche ou techniques, nous supposons que vous développez un logiciel pour traiter des données (signaux ou images). Ce logiciel peut être une application autonome, une bibliothèque à utiliser dans d'autres applications, ou même un simple script à exécuter depuis la ligne de commande. Dans tous les cas, vous devrez suivre un processus de développement qui comprend généralement 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 sections suivantes car la documentation de DataLab fournit déjà des informations détaillées 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 modifier 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 : Ce tutoriel suppose que vous avez déjà installé DataLab et 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 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 depuis n'importe quel environnement. Pour ce tutoriel, vous devez seulement installer le paquet Sigima (`pip install sigima`) dans le même environnement que Spyder.

Tester votre algorithme avec DataLab

Supposons que vous avez développé des algorithmes dans le module `my_work` de votre projet. Vous les avez déjà prototypés dans DataLab et développés dans Spyder en écrivant des fonctions qui prennent des données en entrée et retournent 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 retourne des données 1D qui vous permettront de valider votre premier algorithme, qui traite des données 1D.
- `test_my_2d_algorithm` : cette fonction retourne des données 2D qui vous permettront de valider votre second algorithme, qui traite des données 2D.

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

- Tout d'abord, démarrez DataLab et Spyder.
- Rappelez-vous que DataLab est une application autonome qui peut être utilisée depuis n'importe quel environnement, donc vous n'avez pas besoin de l'installer dans le même environnement que Spyder car la connexion entre ces deux applications est établie via un protocole de communication.

Voici comment procéder :

```
# %% Connecting to DataLab current session

from sigima.client 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_datalab.py')
Connecting to DataLab XML-RPC server...OK (port: 54577)

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

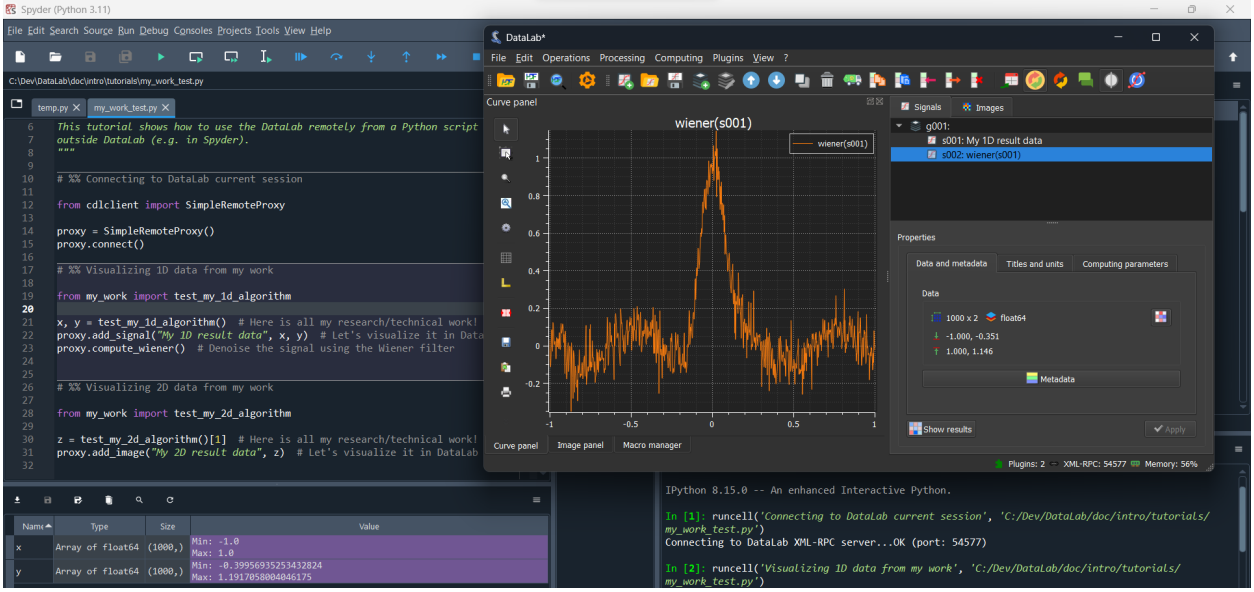


FIG. 105 – Dans cette capture d'écran, nous pouvons voir le résultat de l'évaluation des deux premières cellules : la première cellule se connecte à DataLab, et la seconde cellule visualise les données 1D retournées par la fonction `test_my_1d_algorithm`.

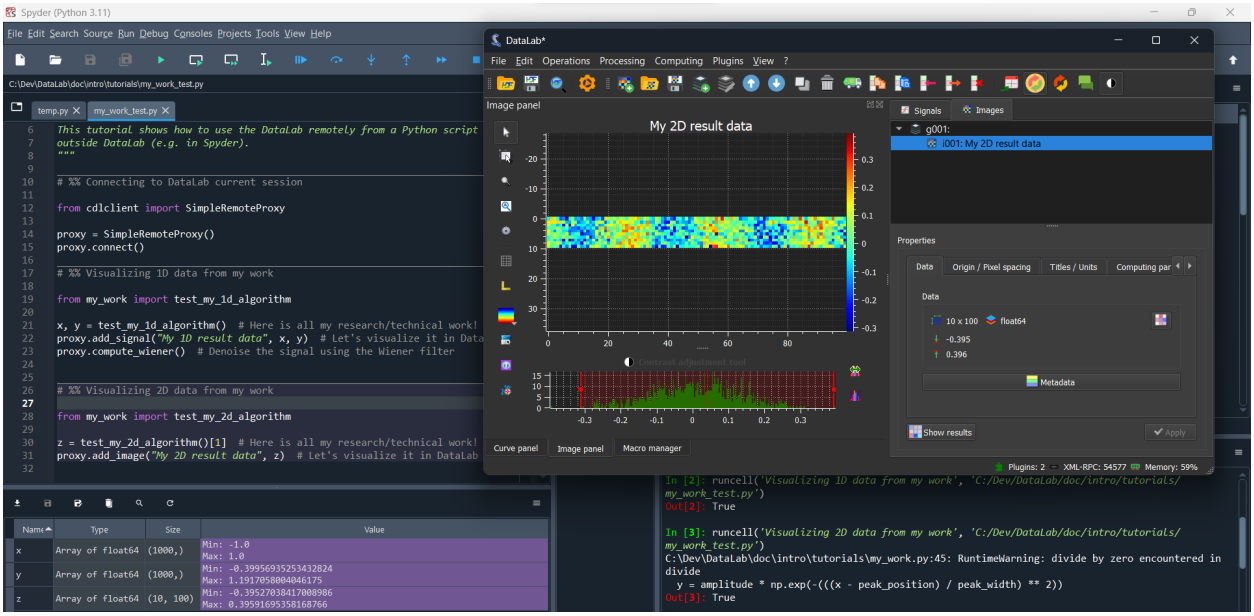


FIG. 106 – Dans cette capture d'écran, nous pouvons voir le résultat de l'évaluation de la troisième cellule : la fonction `test_my_2d_algorithm` retourne un tableau 2D, que nous pouvons visualiser directement dans DataLab.

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 d'exemple que nous voulons déboguer, dans lequel nous avons introduit un paramètre optionnel `debug_with_datalab` qui—s'il est défini à `True`—créera un objet proxy permettant la visualisation pas à pas des données dans DataLab :

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

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

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

(suite sur la page suivante)

(suite de la page précédente)

```
    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

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

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

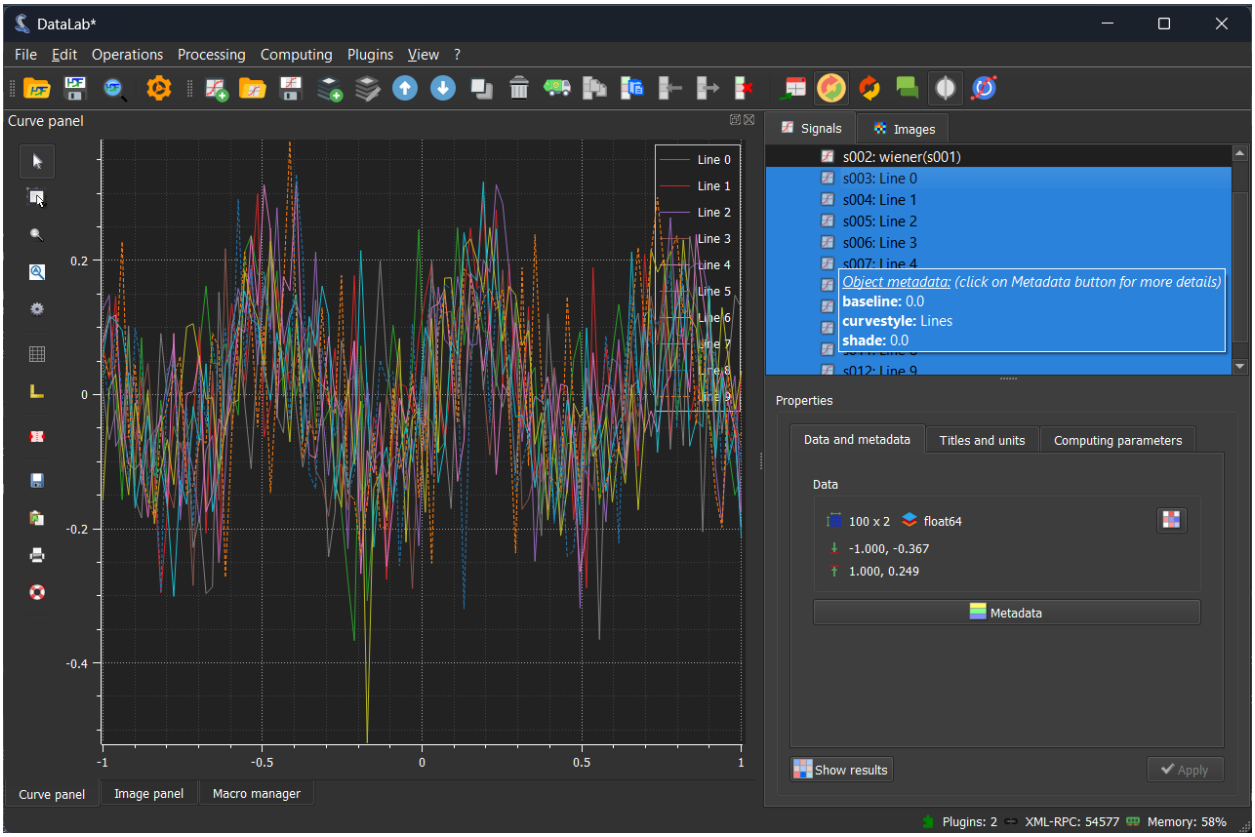


FIG. 107 – Dans cette capture d'écran, nous pouvons voir le résultat de l'évaluation de la première cellule : la fonction `test_my_2d_algorithm` est appelée avec le paramètre `debug_with_datalab` défini à `True` : 10 tableaux 1D sont générés et visualisés dans DataLab.

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. Puisque DataLab s'exécute dans un processus séparé, nous aurions pu interagir avec les données dans DataLab pendant que l'algorithme

est en pause dans Spyder.

CHAPITRE 2

Fonctionnalités

Les paragraphes suivants décrivent les fonctionnalités de DataLab, la plateforme open-source d'analyse et de visualisation de données scientifiques.

Note : Avant de plonger dans les détails des autres parties de la documentation, il est recommandé de commencer par la page *Vue d'ensemble*, 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 Vue d'ensemble et fonctionnalités communes

2.1.1 Vue d'ensemble

Concepts de base

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 gauche 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 droite 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.

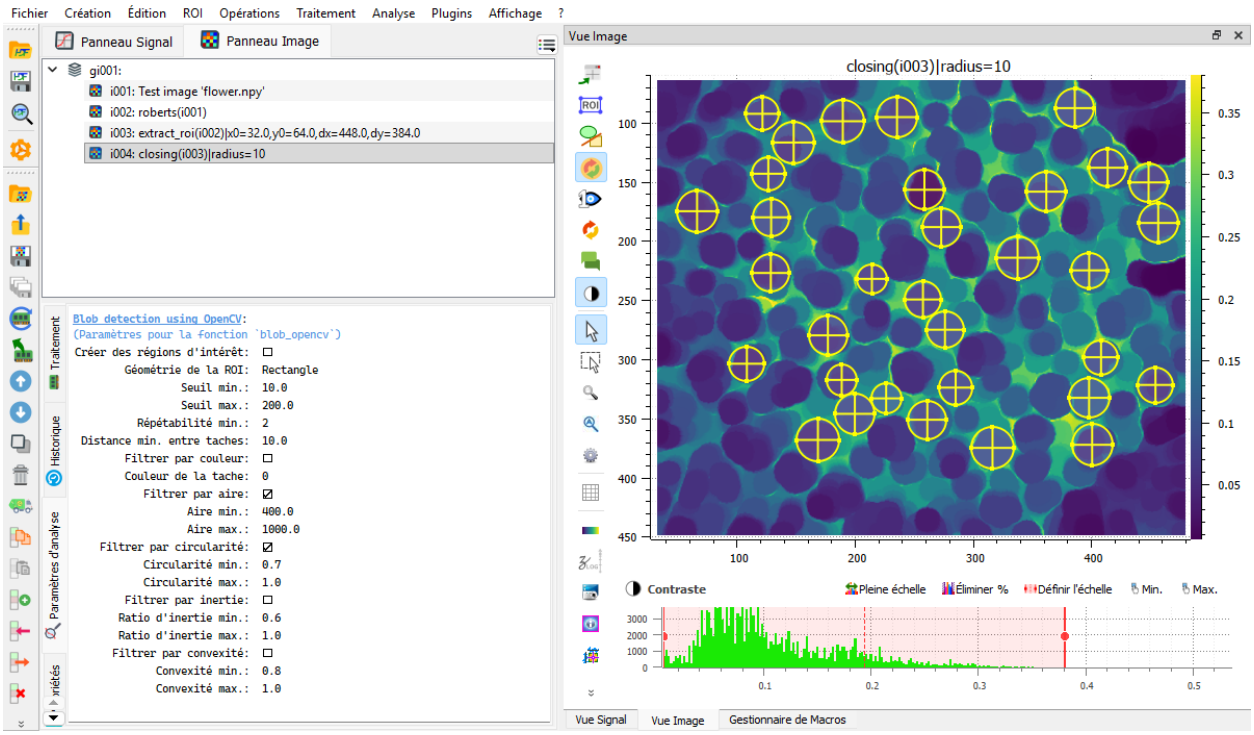


FIG. 1 – Fenêtre principale de DataLab

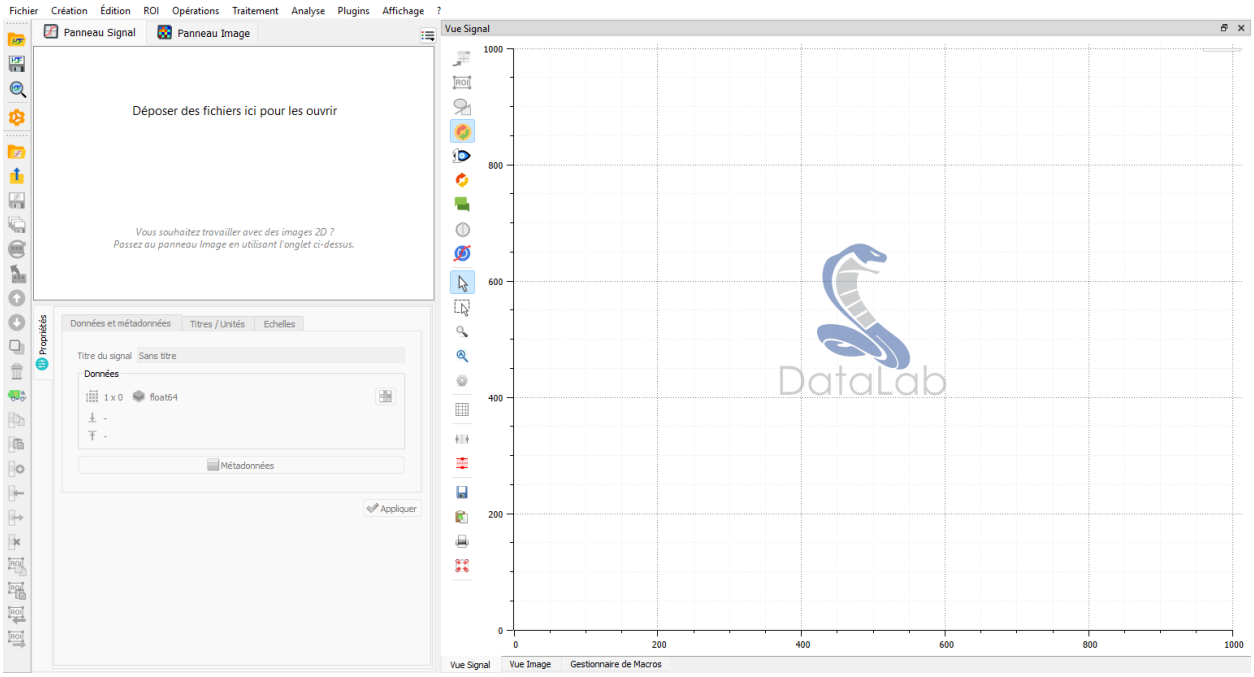


FIG. 2 – Fenêtre principale de DataLab, au démarrage.

Modèle de données interne et espace de travail

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 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 `sigima.objects`).

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 actuellement chargés dans DataLab, dans les panneaux **Signaux** et **Images**.

Chargement et enregistrement de l'espace de travail

Les actions suivantes sont disponibles pour gérer l'espace de travail depuis le menu **Fichier** :

- **Ouvrir un fichier HDF5** : charger un espace de travail à partir d'un fichier HDF5.
- **Enregistrer dans un fichier HDF5** : enregistrer l'espace de travail actuel dans un fichier HDF5.
- **Parcourir un fichier HDF5** : ouvrir le *Explorateur HDF5* pour explorer le contenu d'un fichier HDF5 et importer des jeux de données dans l'espace de travail.

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).

Création et traitement interactifs d'objets

DataLab fournit un flux de travail interactif pour créer des objets et ajuster les paramètres de traitement, vous permettant d'affiner les résultats sans créer plusieurs objets.

Création d'objets interactifs

Lors de la création d'un nouveau signal ou d'une nouvelle image à l'aide des fonctions de création (par exemple, signal gaussien, image de pic 2D, etc.), DataLab stocke les paramètres de création dans les métadonnées de l'objet. Cela permet d'ajuster les paramètres de manière interactive après la création :

1. Créer un signal ou une image en utilisant le menu **Opérations > Créer**
2. Sélectionner l'objet créé dans la liste
3. Un onglet **Création** apparaît dans le panneau des propriétés (en bas à gauche)
4. Modifier n'importe quel paramètre de création (amplitude, fréquence, taille, etc.)
5. Cliquer sur **Appliquer** pour régénérer l'objet avec les nouveaux paramètres

L'objet est mis à jour sur place, préservant tous les résultats de traitement ou d'analyse ultérieurs. Cela est particulièrement utile pour :

- Explorer différentes valeurs de paramètres sans encombrer l'espace de travail
- Affiner les caractéristiques de l'objet tout en observant les résultats
- Objectifs pédagogiques pour démontrer les effets des paramètres

Note : La création interactive n'est disponible que pour les objets créés avec des classes de paramètres (pas pour les données importées à partir de fichiers).

Traitement interactif 1-vers-1

Lors de l'application d'une opération de traitement 1-vers-1 qui a des paramètres configurables (par exemple, filtre gaussien, seuil, opérations morphologiques), DataLab stocke les métadonnées de traitement, permettant l'ajustement des paramètres et le re-traitement :

1. Appliquer une opération de traitement avec des paramètres (par exemple, **Traitement > Filtrage > Filtre gaussien**)
2. L'objet résultant contient des métadonnées de traitement (paramètres, objet source, nom de la fonction)
3. Sélectionner l'objet traité dans la liste
4. Un onglet **Traitement** apparaît dans le panneau des propriétés
5. Modifier les paramètres de traitement (par exemple, valeur sigma du filtre)
6. Cliquer sur **Appliquer** pour re-traiter avec les paramètres mis à jour

L'objet traité est mis à jour sur place avec les nouveaux résultats. Ce flux de travail est idéal pour :

- Ajustement itératif des paramètres du filtre tout en observant les résultats en temps réel
- Ajustement des valeurs de seuil sans créer plusieurs objets intermédiaires
- Expérimentation avec différentes tailles d'éléments de structure morphologique
- Démonstrations éducatives des effets des paramètres sur les résultats du traitement

Note : Exigences en matière de métadonnées de traitement :

- Ne fonctionne que pour les fonctions de traitement 1-vers-1 avec des classes de paramètres
- Les fonctions de traitement sans paramètres (par exemple, valeur absolue, inverse) fonctionnent comme auparavant
- L'objet source doit toujours exister pour le re-traitement (erreur affichée s'il est supprimé)

Non pris en charge pour :

- Traitement 2-vers-1 (opérations combinant deux objets)
- Traitement n-vers-1 (opérations agrégeant plusieurs objets)
- Ces modèles ne bénéficient pas de manière significative de l'ajustement interactif des paramètres

Exemple de flux de travail

Voici un flux de travail typique utilisant le traitement interactif :

1. **Créer un signal de test** : Opérations > Créer > Signal gaussien
   - Paramètres initiaux : amplitude=1.0, mu=50, sigma=10
2. **Ajuster les paramètres de création** : Dans l'onglet Création, changer sigma à 20, cliquer sur Appliquer
   - Le signal est régénéré avec une nouvelle largeur
3. **Appliquer le filtre gaussien** : Traitement > Filtrage > Filtre gaussien
   - Sigma initial=2.0
4. **Ajuster le filtrage** : Dans l'onglet Traitement, essayer sigma=1.0, puis sigma=5.0
   - Chaque application met à jour le résultat filtré
   - Comparer différents niveaux de lissage sans créer plusieurs objets

Cette approche interactive fournit un retour visuel immédiat et garde votre espace de travail organisé en évitant la prolifération d'objets de test avec des paramètres légèrement différents.

2.1.2 Préférences

DataLab propose une boîte de dialogue de préférences complète pour personnaliser le comportement de l'application, les paramètres de visualisation par défaut et les opérations d'entrée/sortie. Les préférences sont organisées en cinq onglets : Général, Traitement, Visualisation, Entrée/sortie et Console.

Général

L'onglet des préférences générales contient les paramètres de la fenêtre principale et les paramètres généraux :

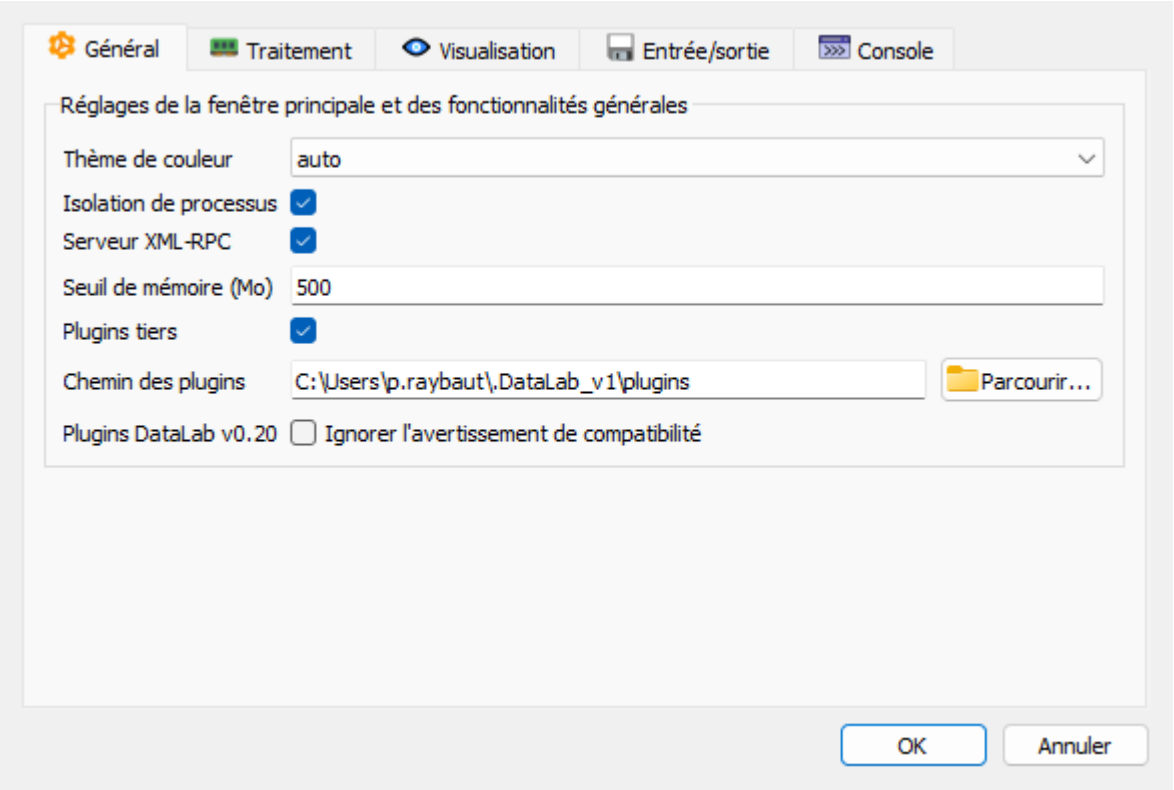


Fig. 3 – Onglet des préférences générales

Mode couleur
: Choisir le mode couleur de l'interface de l'application (par exemple, clair, sombre ou automatique).

Isolation des processus
: Si le réglage est activé, chaque calcul s'exécute dans un processus séparé, empêchant l'application de geler pendant les longs calculs. Il s'agit du paramètre recommandé pour une meilleure réactivité.

Serveur RPC
: Activer le serveur RPC (Remote Procedure Call) pour communiquer avec des applications externes, telles que vos propres scripts s'exécutant dans Spyder, Jupyter ou d'autres logiciels. Cela permet un contrôle programmatique de DataLab.

Seuil de mémoire
: Définir un seuil (en Mo) en dessous duquel un avertissement est affiché avant de charger de nouvelles données. Cela permet d'éviter les erreurs de mémoire insuffisante lors du travail avec de grands ensembles de données. Mettre à 0 pour désactiver l'avertissement.

Plugins tiers
: Activer ou désactiver les plugins tiers au démarrage.

Chemin des plugins
Spécifier le chemin du répertoire où DataLab doit rechercher les plugins tiers. DataLab découvrira également les plugins dans votre PYTHONPATH.

Traitement

L'onglet des préférences de traitement contrôle le comportement des calculs et les paramètres par défaut :

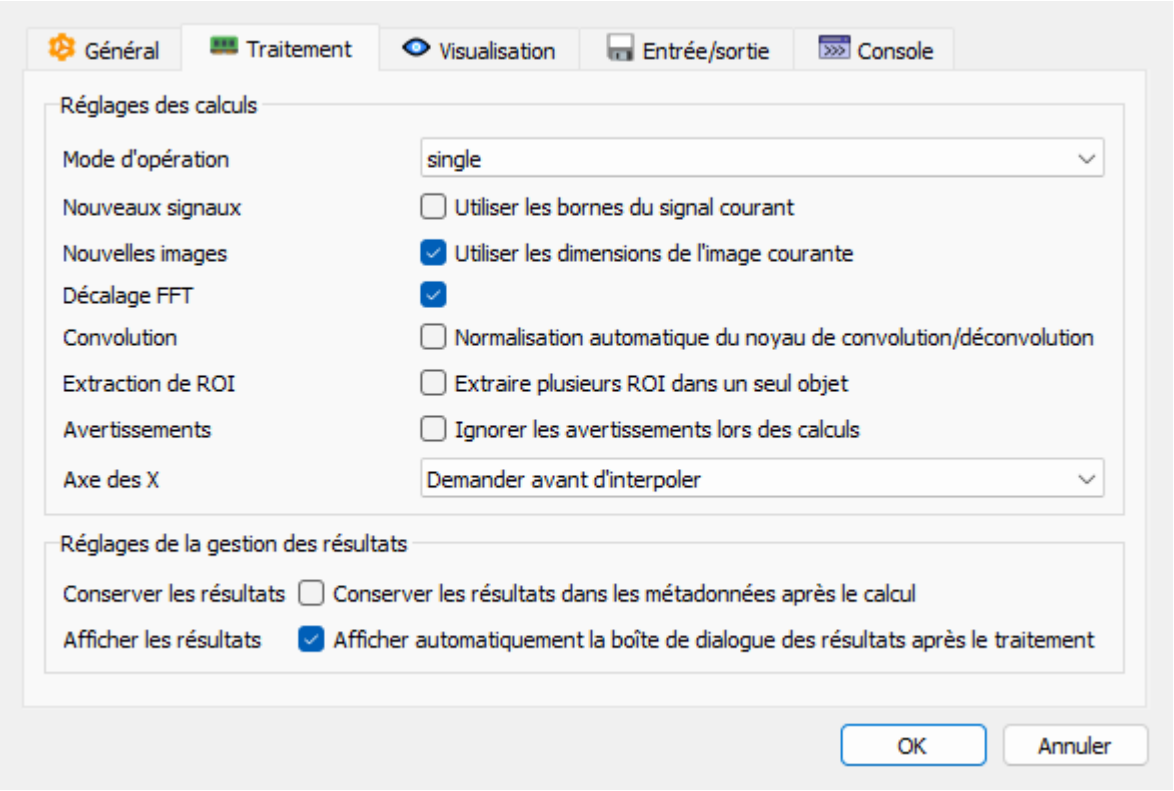


FIG. 4 – Onglet des préférences de traitement

Mode d'opération
Choisir le mode d'opération pour les calculs prenant N entrées :
— **Single** : mode opérande unique
— **Pairwise** : mode opération par paire

Note : Ces modes d'opération déterminent comment DataLab gère les calculs impliquant plusieurs objets. Ils s'appliquent à deux types d'opérations :
— **Opérations N→1** : Combiner N (2) objets en 1 sortie (par exemple somme, moyenne)
— **Opérations N+1→N** : Appliquer une opération entre N (1) objets et 1 opérande pour produire N sorties (par exemple différence, division)

Mode opérande unique (par défaut) : Les opérations sont appliquées indépendamment dans chaque groupe.
— Pour les **opérations N→1** : Tous les objets de chaque groupe sont combinés en un seul résultat par groupe. Exemple avec les groupes G1={A, B} et G2={C, D}, opération de somme :
  — Résultat : (A,B) et (C,D) (un par groupe)
— Pour les **opérations N+1→N** : Chaque objet sélectionné est combiné avec un seul opérande de référence. Exemple avec les groupes G1={A, B} et G2={C, D}, différence avec la référence R :
  — Dans G1 : A-R, B-R

  - Dans G2 : C-R, D-R

Mode opération par paire : Les objets de différents groupes sont combinés aux positions correspondantes (tous les groupes doivent avoir le même nombre d'objets).

- Pour les **opérations N→1** : Les objets à la même position dans chaque groupe sont combinés. Exemple avec les groupes G1={A, B} et G2={C, D}, opération de somme :
  - Résultat : A+C et B+D (appariement par position)
- Pour les **opérations N+1→N** : Les objets aux positions correspondantes sont combinés par paires. Exemple avec les groupes G1={A, B}, G2={C, D}, différence avec le groupe G3={E, F} :
  - Nouveau groupe 1 : A-E, B-F
  - Nouveau groupe 2 : C-E, D-F

Utiliser les bornes du signal pour les nouveaux signaux
Si le réglage est activé, les valeurs xmin et xmax pour les nouveaux signaux sont initialisées à partir des bornes du signal actuel. Lorsque désactivé, les valeurs par défaut sont utilisées.

Utiliser les dimensions d'image pour les nouvelles images
Si le réglage est activé, les valeurs de largeur et de hauteur pour les nouvelles images sont initialisées à partir des dimensions de l'image actuelle. Lorsque désactivé, les valeurs par défaut sont utilisées.

Décalage FFT
Activer le décalage FFT pour centrer la composante de fréquence nulle dans le spectre de fréquences pour une visualisation et une analyse plus faciles.

Extraire plusieurs ROI dans un seul objet
Si le réglage est activé, plusieurs ROI (Régions d'Intérêt) sont extraites dans un seul objet. Lorsque désactivé, chaque ROI est extraite dans un objet séparé.

Ignorer les avertissements
Supprimer les messages d'avertissement pendant les calculs.

Comportement de compatibilité des tableaux X
Choisir le comportement lorsque les tableaux X sont incompatibles dans les calculs multi-signaux :

- **Demander** : afficher une boîte de dialogue de confirmation (par défaut)
- **Interpoler** : interpoler automatiquement les signaux

Gestion des résultats

Conserver les résultats dans les métadonnées après le calcul
Si le réglage est activé, les résultats des analyses précédentes sont conservés dans les métadonnées de l'objet après le calcul. Lorsque désactivé, les résultats sont supprimés. Cette option est désactivée par défaut pour éviter toute confusion due à des résultats obsolètes.

Afficher automatiquement la boîte de dialogue des résultats après le traitement
Si le réglage est activé, la boîte de dialogue des résultats est affichée automatiquement après chaque opération de traitement produisant des résultats. Lorsque désactivé, la boîte de dialogue des résultats n'est pas affichée automatiquement, mais les résultats peuvent toujours être consultés à l'aide du bouton ou de l'option de menu dédiée.

Visualisation

L'onglet des préférences de visualisation contrôle comment les données sont affichées. Cet onglet est organisé en quatre sous-onglets : Commun, Signaux, Images et Résultats.

Paramètres communs

Le sous-onglet Commun contient des paramètres qui s'appliquent à toutes les visualisations :

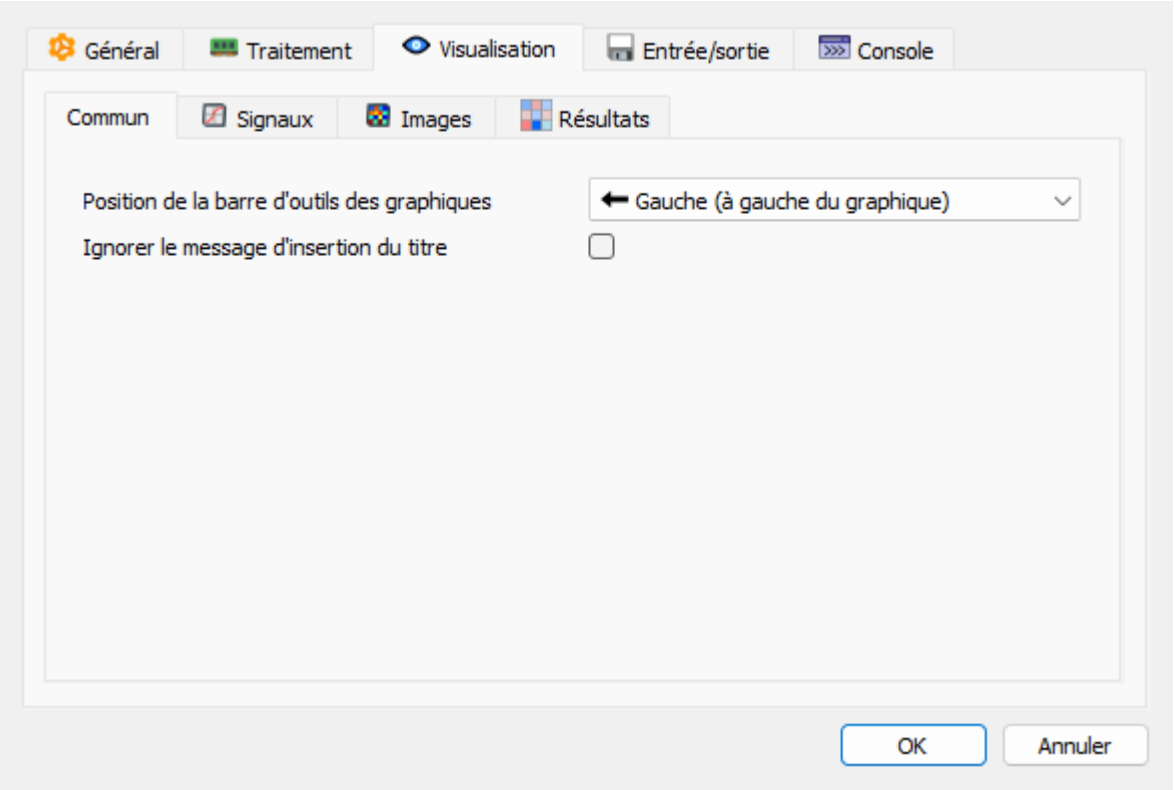


Fig. 5 – Préférences de visualisation - Sous-onglet Commun

Position de la barre d'outils de tracé
: Choisir où positionner la barre d'outils de tracé (en haut, en bas, à gauche ou à droite du tracé).

Ignorer le message d'insertion de titre
: Supprimer le message d'information lors de l'insertion d'un titre d'objet comme étiquette d'annotation.

Signaux

Le sous-onglet Signaux contient des paramètres spécifiques aux visualisations de signaux :

Épaisseur par défaut
: Épaisseur de trait par défaut pour les courbes représentant les signaux. Ce paramètre affecte toutes les visualisations de signaux, sauf si elles sont remplacées individuellement. Remarque : pour les signaux dépassant le seuil de performance de l'épaisseur de trait (voir ci-dessous), l'épaisseur de trait est automatiquement plafonnée à 1,0 pour des performances de rendu optimales.

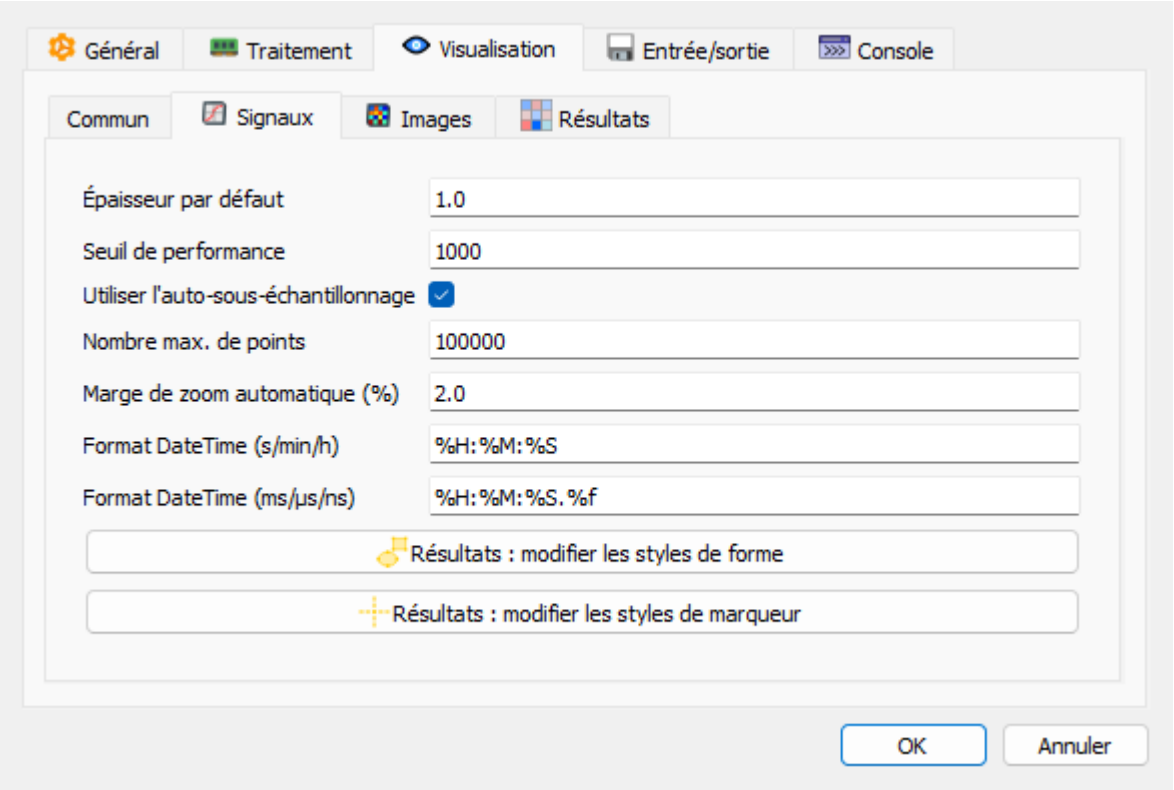


FIG. 6 – Préférences de visualisation - Sous-onglet Signaux

Seuil de performance de l'épaisseur de trait
Pour les signaux comportant plus de ce nombre de points (par défaut : 1 000), l'épaisseur de trait est automatiquement limitée à 1,0 pour des raisons de performance. Cela évite le ralentissement du rendu d'environ 10x causé par le moteur raster de Qt lors du dessin de lignes épaisses (largeur > 1,0) sur de grands ensembles de données. Pour les signaux plus petits, l'épaisseur de trait par défaut configurée s'applique normalement. Cette optimisation est transparente et ne nécessite aucune intervention de l'utilisateur.

Utiliser le sous-échantillonnage automatique
Activer le sous-échantillonnage automatique pour les grands signaux afin d'améliorer les performances et la clarté de la visualisation.

Nombre maximum de points de sous-échantillonnage
Nombre maximum de points à afficher lorsque le sous-échantillonnage est activé (par défaut : 10 000).

Marge d'échelle automatique
Pourcentage de marge à ajouter autour des données lors de la mise à l'échelle automatique des tracés de signaux. Une valeur de 0,2 % ajoute une petite marge pour une meilleure visualisation. Mettre à 0 % pour aucune marge (bornes de données exactes).

Format DateTime (s/min/h)
Chaîne de format pour les étiquettes de l'axe X datetime lors de l'utilisation d'unités de temps standard (s, min, h). Utilise les codes de format strftime de Python (par exemple, %H :%M :%S pour heures :minutes :secondes).

Format DateTime (ms/s/ns)
Chaîne de format pour les étiquettes de l'axe X datetime lors de l'utilisation d'unités de temps infra-secondes (ms, s, ns). Utilise les codes de format strftime de Python (par exemple, %H :%M :%S.%f pour heures :minutes :secondes.microsecondes).

Résultats : modifier les styles de forme
Cliquer sur ce bouton pour configurer le style visuel des formes d'annotation (rectangles, cercles, segments, etc.) affichées sur les tracés de signaux. Cela inclut :

— Style de ligne, couleur et largeur
— Motif de remplissage, couleur et transparence
— Forme du symbole, taille et couleurs

Ces paramètres s'appliquent à toutes les formes de résultats dessinées sur les tracés de signaux (par exemple, les marqueurs de pics, les indicateurs FWHM, les résultats de détection de caractéristiques).

Résultats : modifier les styles de marqueur

Cliquer sur ce bouton pour configurer le style visuel des marqueurs de curseur sur les tracés de signaux. Cela inclut :

— Style de ligne, couleur et largeur
— Apparence du symbole
— Formatage et positionnement des étiquettes de texte
— Transparence de l'arrière-plan

Ces paramètres s'appliquent aux marqueurs de type curseur utilisés dans les résultats d'analyse de signaux.

Images

Le sous-onglet Images contient des paramètres spécifiques aux visualisations d'images :

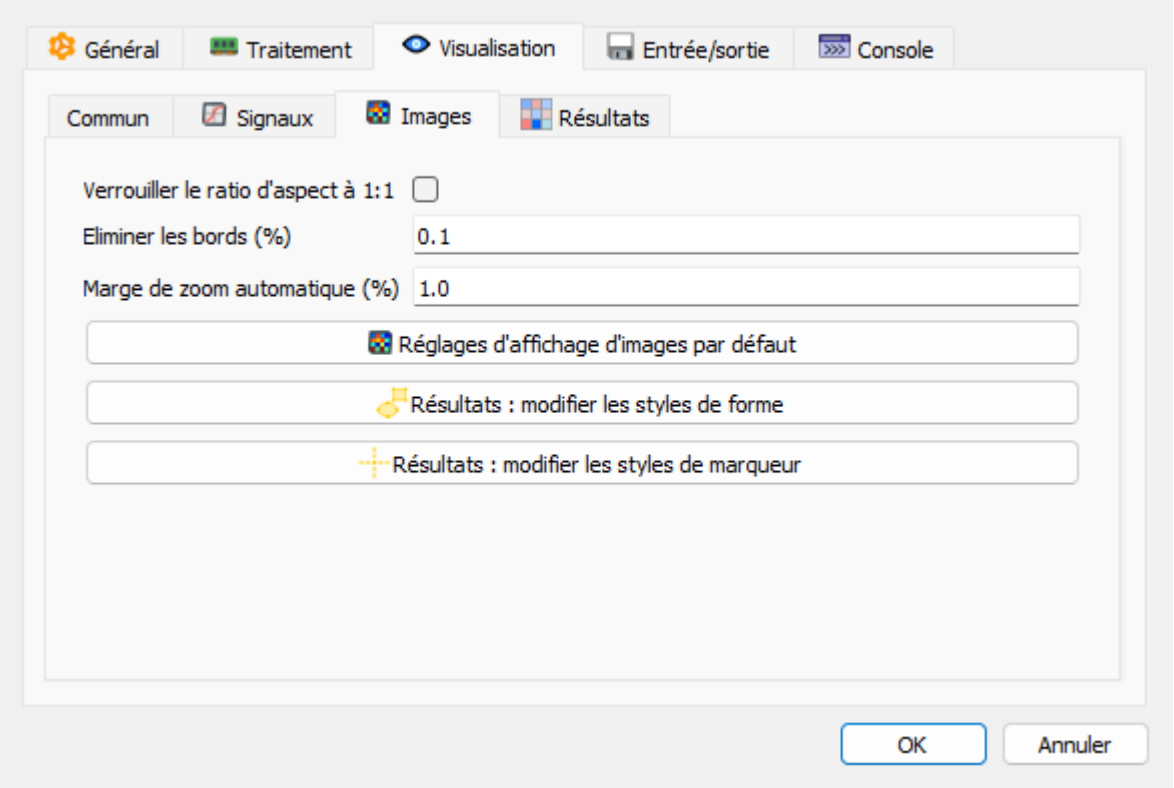


Fig. 7 – Préférences de visualisation - Sous-onglet Images

Verrouiller le rapport d'aspect à 1 :1

Si le réglage est activé, le rapport d'aspect des images est verrouillé à 1 :1. Lorsque désactivé, le rapport d'aspect est déterminé par la taille physique du pixel (paramètre par défaut et recommandé).

Éliminer les valeurs aberrantes

Pourcentage des valeurs les plus élevées et les plus basses à éliminer de l'histogramme de l'image. Les valeurs recommandées sont inférieures à 1 %.

Marge d'échelle automatique
Pourcentage de marge à ajouter autour des données lors de la mise à l'échelle automatique des tracés d'images. Une valeur de 0,2 % ajoute une petite marge pour une meilleure visualisation. Mettre à 0 % pour aucune marge (bornes de données exactes).

Paramètres de visualisation d'image par défaut
Cliquer sur ce bouton pour configurer les paramètres de visualisation par défaut pour les images (carte de couleurs, interpolation, contraste, etc.).

Résultats : modifier les styles de forme
Cliquer sur ce bouton pour configurer le style visuel des formes d'annotation affichées sur les tracés d'images. Les paramètres sont similaires à ceux des formes de signaux mais optimisés pour la visualisation d'images (par exemple, différentes couleurs pour une meilleure visibilité sur les images).

Résultats : modifier les styles de marqueur
Cliquer sur ce bouton pour configurer le style visuel des marqueurs de curseur sur les tracés d'images.

Résultats

L'onglet des préférences de résultats contient des paramètres pour l'affichage des résultats d'analyse sur les graphiques :

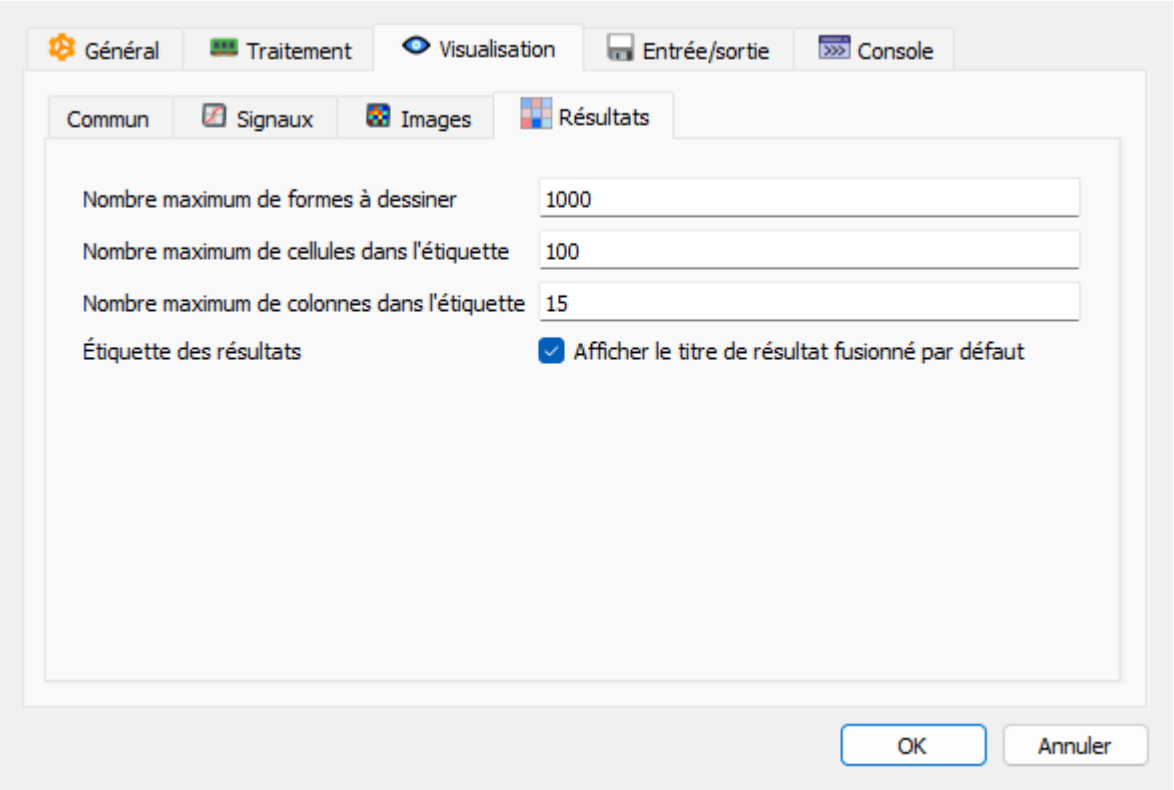


Fig. 8 – Préférences de visualisation - Sous-onglet Résultats

Ces paramètres contrôlent la façon dont les résultats d'analyse sont affichés sur les graphiques afin de limiter les problèmes de performance avec de grands ensembles de données :

Nombre maximum de formes à dessiner
Nombre maximum de formes géométriques à dessiner sur le graphique (par défaut : 1 000). Lorsque le nombre de formes dépasse cette limite, seules les N premières formes sont dessinées et une étiquette d'avertissement est affichée.

Nombre maximum de cellules dans l'étiquette
Nombre maximum de cellules de tableau (lignes × colonnes) à afficher dans les étiquettes de résultats fusionnés sur les graphiques (par défaut : 100). Lorsque le nombre de cellules dépasse cette limite, le tableau est tronqué.

Nombre maximum de colonnes dans l'étiquette
Nombre maximum de colonnes à afficher dans les étiquettes de résultats fusionnés (par défaut : 15). Lorsque le nombre de colonnes dépasse cette limite, seules les N premières colonnes sont affichées.

Afficher l'étiquette de résultat fusionné par défaut
Si le réglage est activé, l'étiquette de résultat fusionné est affichée sur le graphique par défaut pour les nouveaux objets. Ce paramètre peut être basculé par objet à l'aide de la case à cocher dans le panneau Propriétés.

Note : Ces paramètres n'affectent que la visualisation des résultats sur les graphiques. Ils n'affectent pas le calcul réel ou le stockage des résultats dans les métadonnées.

Entrée/sortie

L'onglet des préférences d'entrée/sortie contrôle les opérations d'entrée/sortie :

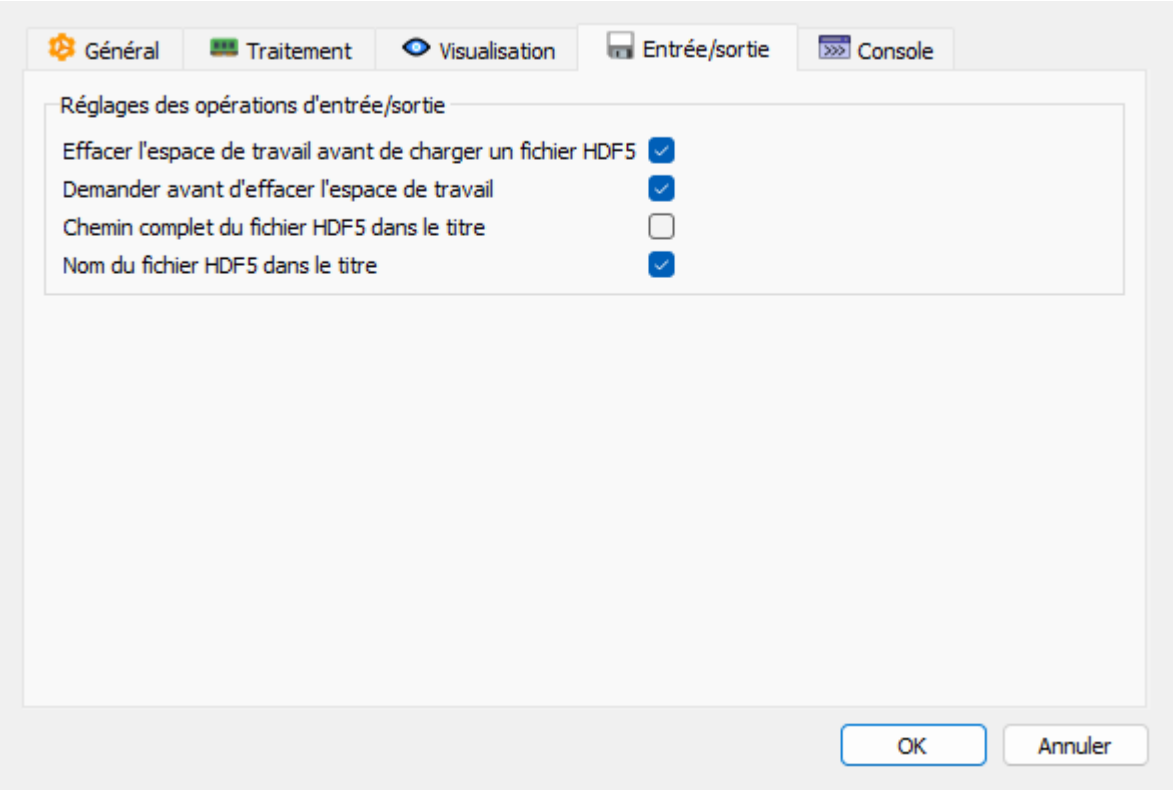


Fig. 9 – Onglet des préférences d'entrée/sortie

Effacer l'espace de travail avant de charger le fichier HDF5
Si le réglage est activé, l'espace de travail est effacé avant de charger un fichier HDF5.

Demander avant d'effacer l'espace de travail
Si le réglage est activé, une boîte de dialogue de confirmation est affichée avant d'effacer l'espace de travail.

Chemin complet HDF5 dans le titre
Si le réglage est activé, le chemin complet du jeu de données HDF5 est utilisé comme titre pour l'objet signal/image. Lorsque désactivé, seul le nom du jeu de données est utilisé.

Nom de fichier HDF5 dans le titre
Si le réglage est activé, le nom du fichier HDF5 est ajouté comme suffixe au titre de l'objet signal/image.

Console

L'onglet des préférences de console configure la console interne pour le débogage et les utilisateurs avancés :

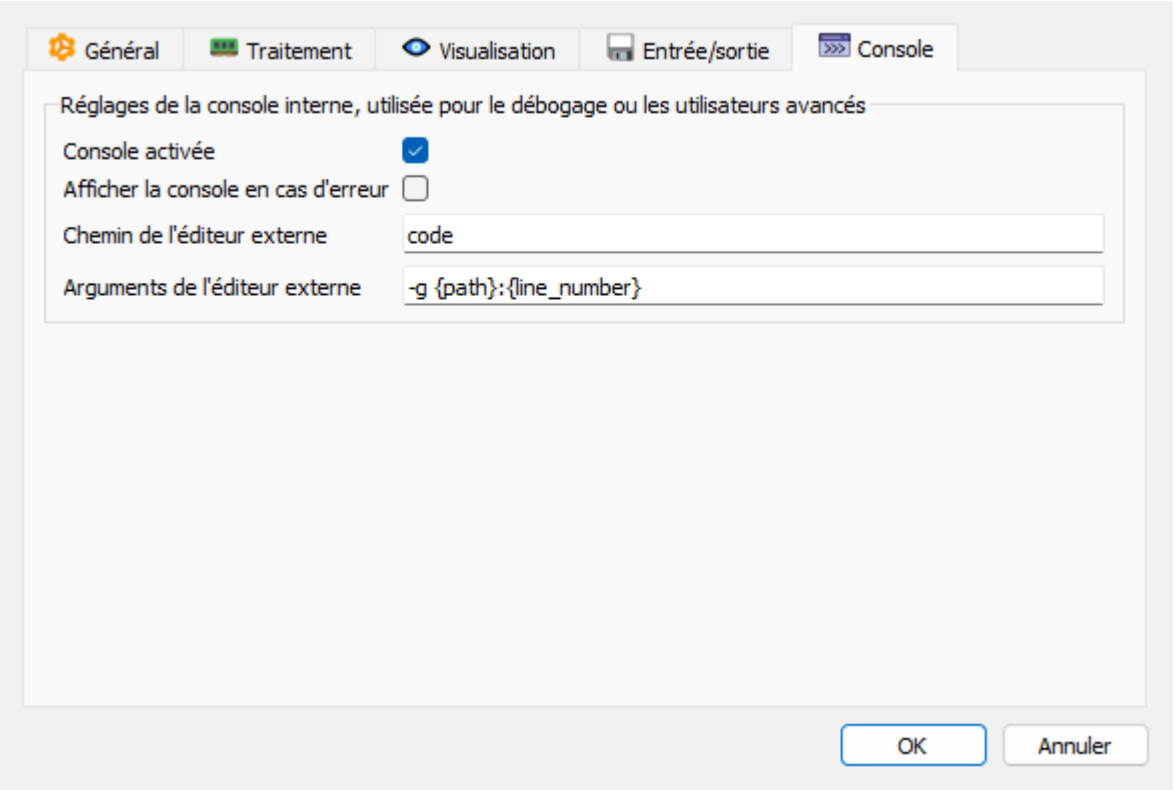


Fig. 10 – Onglet des préférences de console

Console activée
Activer la console Python interne pour le débogage et les scripts avancés.

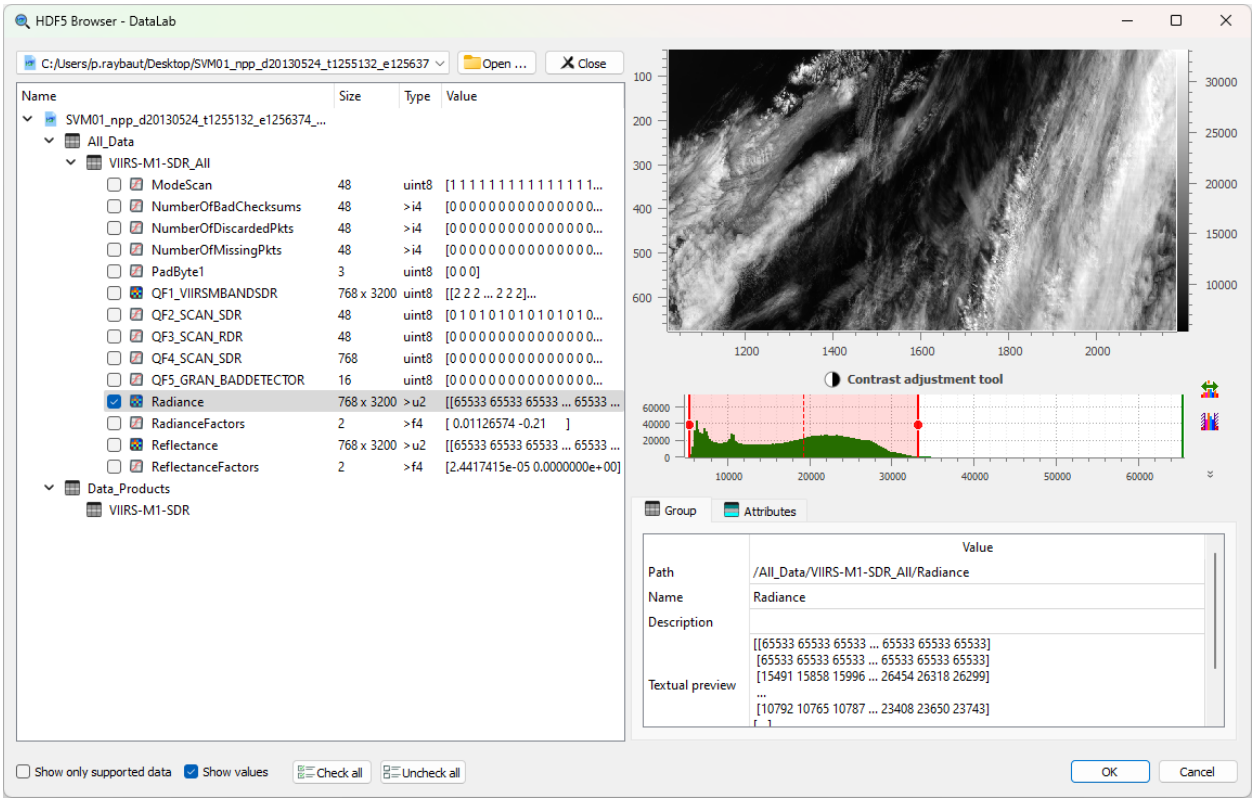
Afficher la console en cas d'erreur
Si le réglage est activé, la console est automatiquement affichée lorsqu'une erreur se produit dans l'application. Ceci est utile pour le débogage car cela permet de voir la trace d'erreur.

Chemin de l'éditeur externe
Chemin vers un éditeur de texte externe à utiliser pour éditer du code Python depuis la console.

Arguments de l'éditeur externe
Arguments de ligne de commande à passer à l'éditeur externe.

2.1.3 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.

2.2 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.

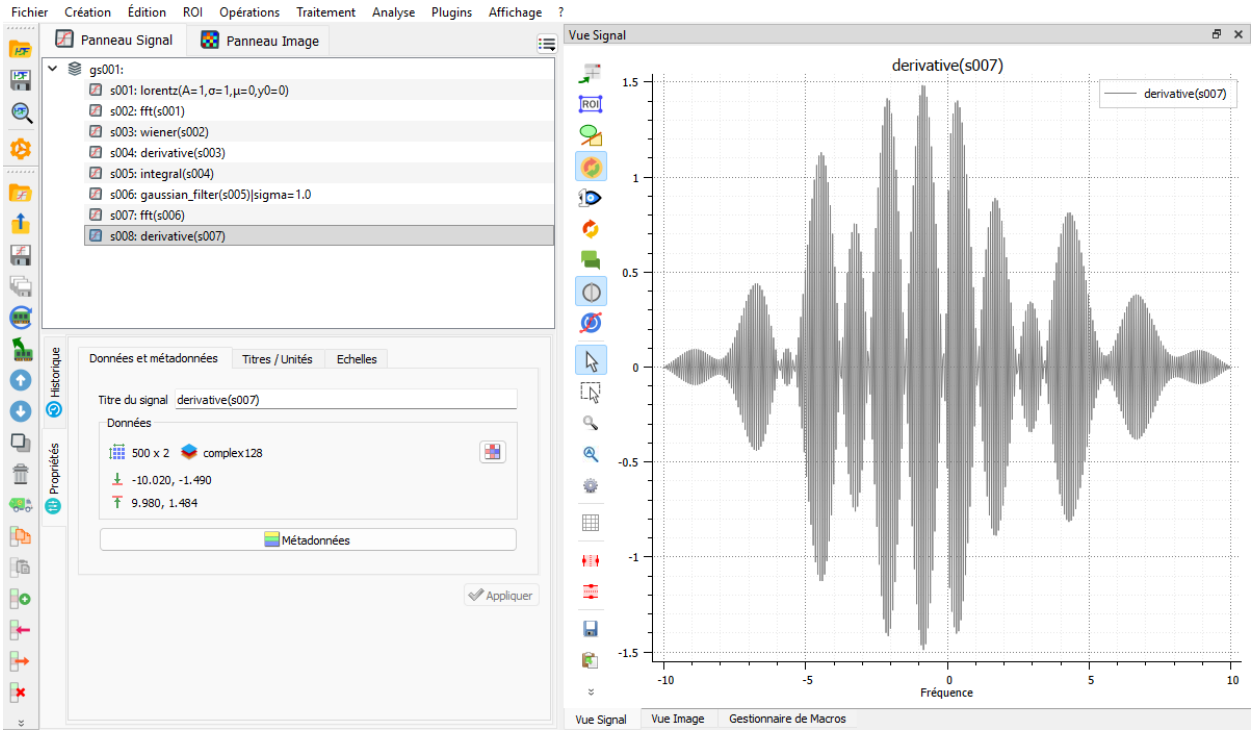


FIG. 11 – Fenêtre principale de DataLab : vue du traitement du signal

2.2.1 Ouvrir et enregistrer des signaux

Cette section décrit comment ouvrir et enregistrer des signaux (et des espaces de travail).

Note : Pour créer de nouveaux signaux à partir de modèles mathématiques, voir *Créer des signaux*.

Lorsque le « Panneau Signal » est sélectionné, les menus et barres d'outils sont mis à jour pour fournir les actions liées aux signaux.

Le menu « Fichier » vous permet de :

- Ouvrir, enregistrer et fermer des signaux (voir ci-dessous).
- Sauvegarder et restaurer l'espace de travail actuel ou parcourir les fichiers HDF5 (voir *Vue d'ensemble*).
- Modifier les préférences de DataLab (voir *Préférences*).

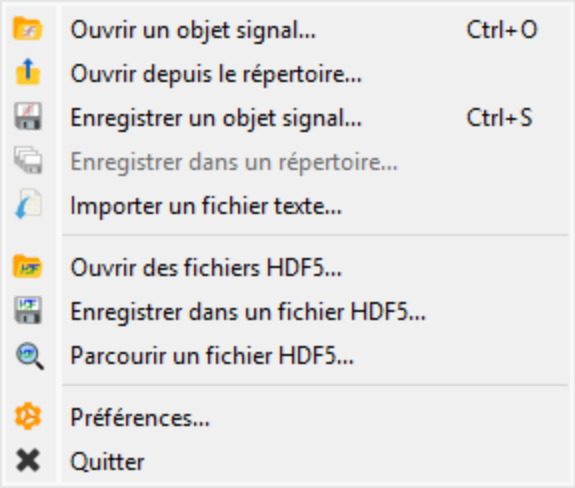


FIG. 12 – Capture d'écran du menu « Fichier ».

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
Fichiers MAT	.mat
Fichiers FT-Lab	.sig

Ouvrir depuis un répertoire

Ouvrir plusieurs signaux depuis un répertoire spécifié.

Enregistrer un signal

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

Type de fichier	Extensions
Fichiers texte	.csv

Enregistrer les signaux dans un répertoire

Enregistrer tous les signaux sélectionnés dans un répertoire spécifié, avec un patron de nom de fichier et un format de signal configurables.

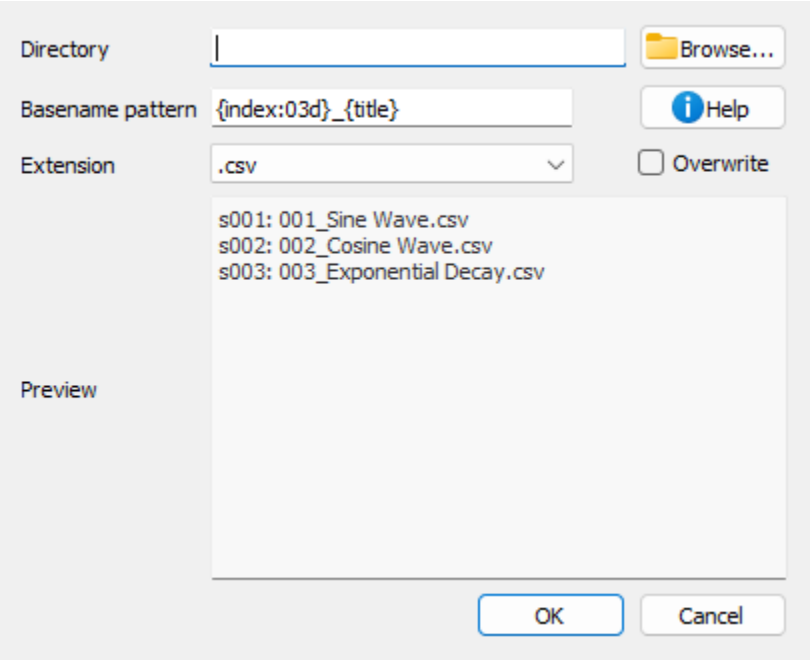


Fig. 13 – Boîte de dialogue Enregistrer les signaux dans un répertoire.

Lorsque vous sélectionnez « Enregistrer dans un répertoire… » dans le menu Fichier, une boîte de dialogue apparaît où vous pouvez :

- **Répertoire** : Choisissez le répertoire cible où les signaux seront enregistrés.
- **Nom de fichier** : Définissez un modèle pour les noms de fichiers en utilisant des chaînes de format Python.
- **Extension de fichier** : Sélectionnez le format de sortie (.csv, .txt, etc.).
- **Écraser** : Choisissez si vous souhaitez écraser les fichiers existants.
- **Aperçu** : Voir la liste des fichiers qui seront créés (avec les ID d'objet)

Le modèle de nom de fichier prend en charge les espaces réservés suivants :

- `{title}` : Signal title
- `{index}` : 1-based index of the signal in the selection (with zero-padding)
- `{count}` : Total number of selected signals
- `{xlabel}`, `{xunit}`, `{ylabel}`, `{yunit}` : Axis labels and units
- `{metadata[key]}` : Access metadata values

You can also use format modifiers, for example `{index:03d}` will format the index with 3 digits zero-padding (001, 002, 003, etc.).

Importer un fichier texte

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.

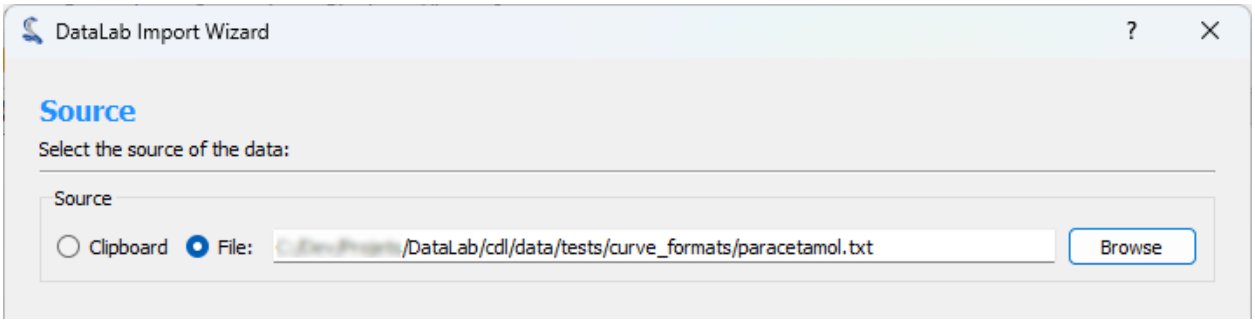


Fig. 14 – Etape 1 : Sélectionner la source

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.

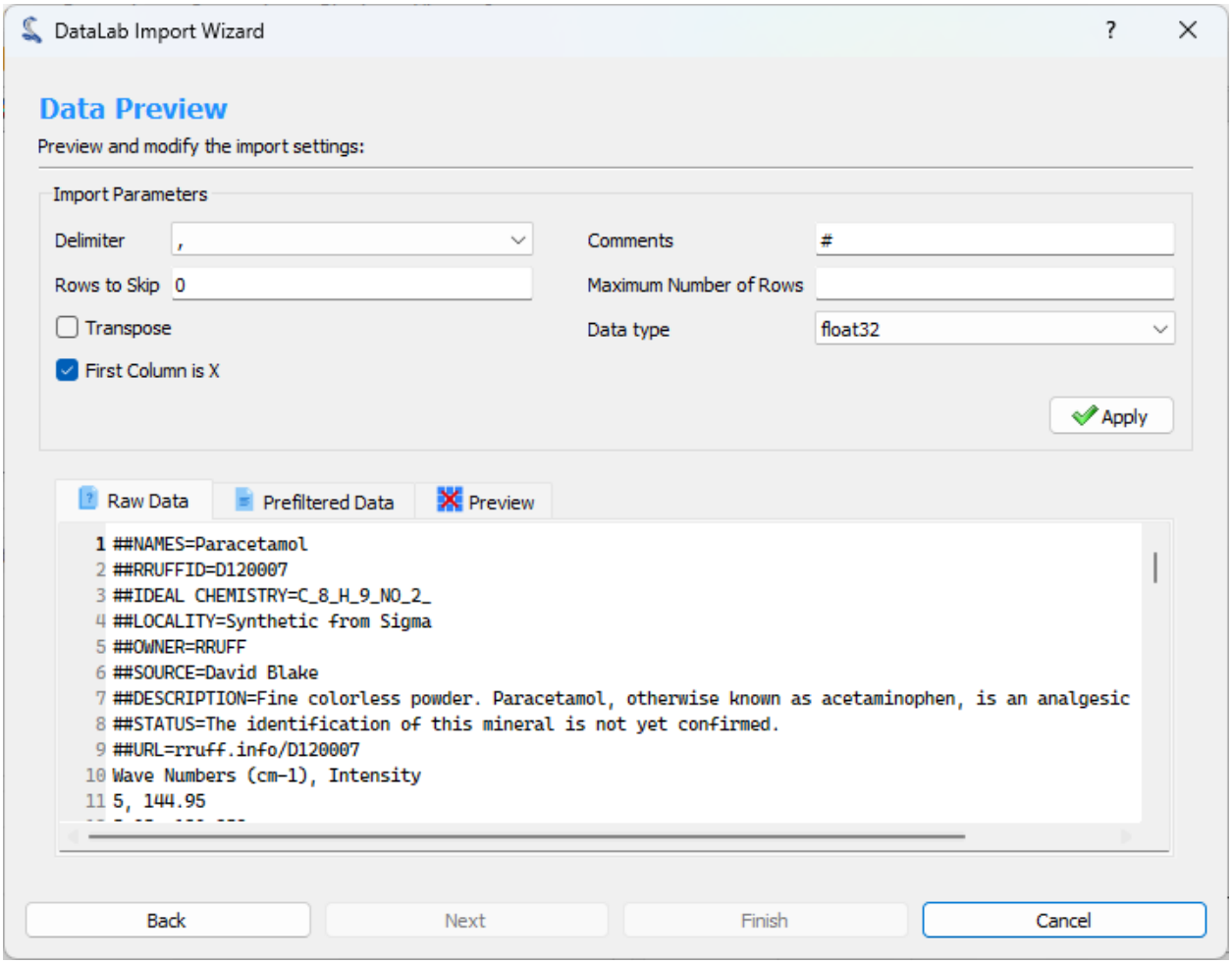


FIG. 15 – Etape 2 : Configurer l'importation

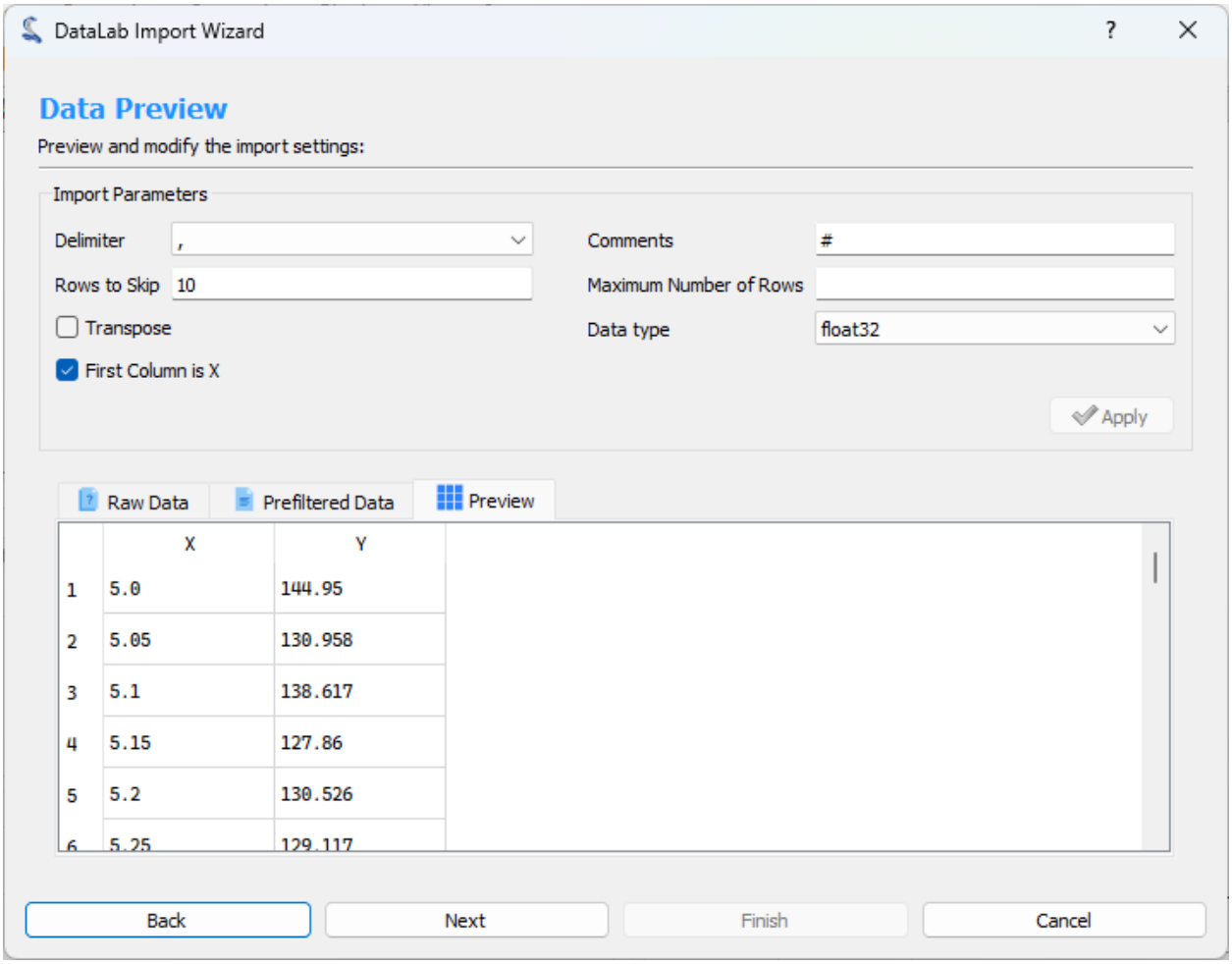


Fig. 16 – Etape 2 : Prévisualiser le résultat

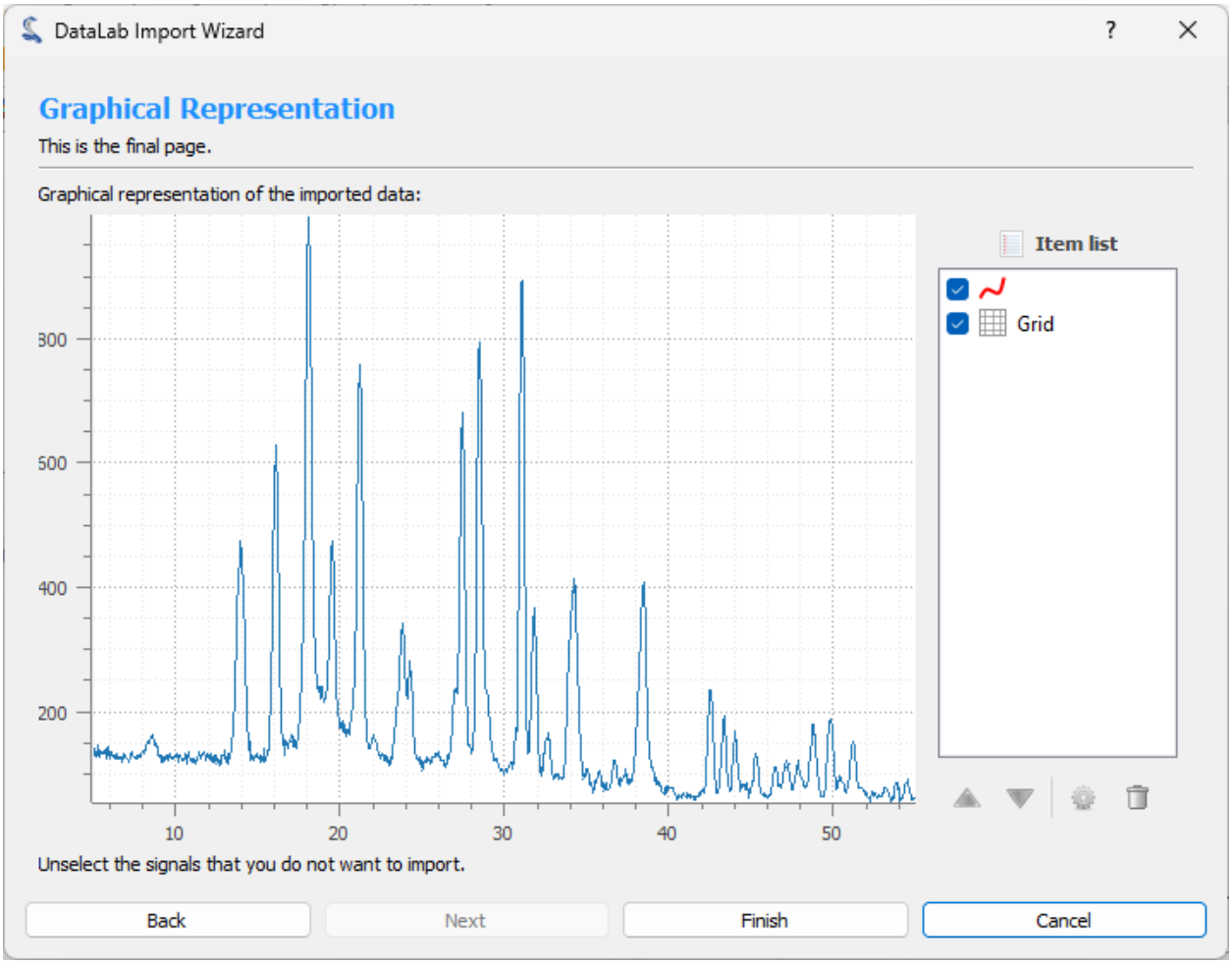


Fig. 17 – Etape 3 : Afficher la représentation graphique

2.2.2 Créer des signaux

Cette section décrit comment créer de nouveaux signaux à partir de divers modèles mathématiques.

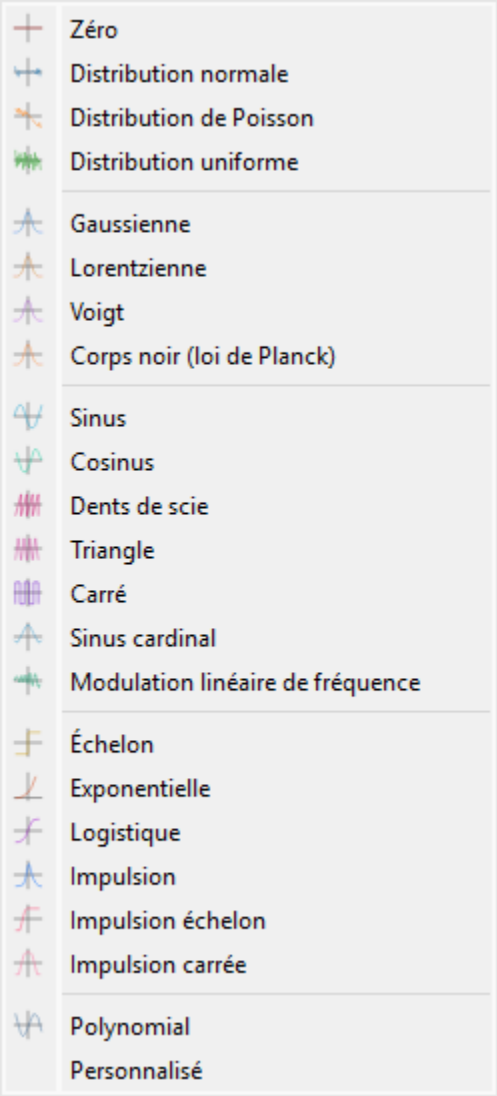


FIG. 18 – Capture d'écran du menu « Création ».

Lorsque le « Panneau Signal » est sélectionné, les menus et barres d'outils sont mis à jour pour fournir les actions relatives aux signaux.

Le menu « Création » permet de créer de nouveaux signaux à partir de divers modèles (voir ci-dessous).

Nouveau signal

Créer un nouveau signal à partir de divers modèles :

Icône	Modèle	Équation
	Zéro	$y[i] = 0$
	Distribution normale	$y[i]$ suit une distribution normale avec moyenne et écart-type configurables
	Distribution de Poisson	$y[i]$ suit une distribution de Poisson avec moyenne configurable
	Distribution uniforme	$y[i]$ suit une distribution uniforme entre deux bornes configurables
	Gaussienne	$y = y_0 + \frac{A}{\sqrt{2\pi} \cdot \sigma} \cdot \exp\left(-\frac{1}{2} \cdot \left(\frac{x - x_0}{\sigma}\right)^2\right)$
	Lorentzienne	$y = y_0 + \frac{A}{\sigma \cdot \pi} \cdot \frac{1}{1 + \left(\frac{x - x_0}{\sigma}\right)^2}$
	Voigt	$y = y_0 + A \cdot \frac{\Re\left(\exp\left(-z^2\right) \cdot \mathrm{erfc}(-j \cdot z)\right)}{\sqrt{2\pi} \cdot \sigma}$ avec $z = \frac{x - x_0 - j \cdot \sigma}{\sqrt{2} \cdot \sigma}$
	Corps noir (loi de Planck)	$y = \frac{2hc^2}{\lambda^5 \left(\exp\left(\frac{hc}{\lambda kT}\right) - 1\right)}$
	Sinus	$y = y_0 + A \sin\left(2\pi \cdot f \cdot x + \phi\right)$
	Cosinus	$y = y_0 + A \cos\left(2\pi \cdot f \cdot x + \phi\right)$
	Dent de scie	$y = y_0 + A\left(2\left(fx + \frac{\phi}{2\pi} - \left\lfloor fx + \frac{\phi}{2\pi} + \frac{1}{2}\right\rfloor\right)\right)$
	Triangle	$y = y_0 + A\,\mathrm{sawtooth}\left(2\pi f x + \phi, \mathrm{width} = 0.5\right)$
	Carré	$y = y_0 + A\,\mathrm{sgn}\left(\sin\left(2\pi f x + \phi\right)\right)$
	Sinus cardinal	$y = y_0 + A\,\mathrm{sinc}\left(2\pi f x + \phi\right)$
	Balayage linéaire	$y = y_0 + A \sin\left(\phi_0 + 2\pi\left(f_0\, x + \frac{1}{2} c\, x^2\right)\right)$
	Échelon	$y = y_0 + A \begin{cases} 1 & \text{si } x > x_0 \\ 0 & \text{sinon} \end{cases}$
	Exponentielle	$y = y_0 + A \exp\left(B \cdot x\right)$
	Logistique	$y = y_0 + \frac{A}{1 + \exp\left(-k\left(x - x_0\right)\right)}$
	Impulsion	$y = y_0 + A \begin{cases} 1 & \text{si } x_0 < x < x_1 \\ 0 & \text{sinon} \end{cases}$
	Impulsion à front monté	$y = \left(\begin{cases} y_0 & \text{si } x < t_0 \\ y_0 + A \cdot \frac{x - t_0}{t_r} & \text{si } t_0 \leq x < t_0 + t_r \\ y_0 + A & \text{si } x \geq t_0 + t_r \end{cases}\right) + \mathcal{N}\left(0, \sigma_n\right)$ où : — t_0 est le temps de début de l'impulsion, — t_r est le temps de montée, — σ_n est l'amplitude du bruit

suite sur la page suivante

Tableau 1 – suite de la page précédente

Icône	Modèle	Équation
	Impulsion carrée	$y(x) = \left(\begin{cases} y_0 & \text{si } x < t_0 \\ y_0 + A \cdot \dfrac{x - t_0}{t_r} & \text{si } t_0 \leq x < t_0 + t_r \\ y_0 + A & \text{si } t_0 + t_r \leq x < t_1 \\ y_0 + A - A \cdot \dfrac{x - t_1}{t_f} & \text{si } t_1 \leq x < t_1 + t_f \\ y_0 & \text{si } x \geq t_1 + t_f \end{cases} \right) + \mathcal{N}(0, \sigma_n)$ où : — t_0 est le temps de début de l'impulsion, — t_r est le temps de montée, — t_f est le temps de descente, — $t_1 = t_0 + t_r + d$ est le temps auquel la décroissance commence, — σ_n est l'amplitude du bruit — la durée du plateau d est calculée par $d = t_{\mathrm{FWHM}} - \dfrac{t_r + t_f}{2}$ à partir de la largeur à mi-hauteur t_{FWHM} **Avertissement :** La durée du plateau d ne doit pas être négative.
	Polynôme	$y = y_0 + A_0 + A_1 \cdot x + A_2 \cdot x^2 + \ldots + A_n \cdot x^n$
	Personnalisé	Saisie manuelle des valeurs X et Y

2.2.3 Manipuler les métadonnées et les annotations

Cette section décrit comment manipuler les métadonnées et les annotations dans DataLab.

Les métadonnées du signal contiennent diverses informations sur le signal ou sa représentation, telles que les paramètres d'affichage, les régions d'intérêt (ROIs), l'historique de la chaîne de traitement, les résultats d'analyse et toute autre information que vous avez pu ajouter aux métadonnées d'un signal (ou qui provient du fichier du signal lui-même).

Le menu « Édition » vous permet d'effectuer des opérations d'édition classiques sur le signal ou le groupe de signaux actuel (créer/renommer un groupe, déplacer vers le haut/vers le bas, supprimer le signal/le groupe de signaux, etc.).

Comme détaillé ci-dessous, il vous permet également de :

— Naviguer et utiliser l'historique de la chaîne de traitement grâce à des actions telles que « Recalculer » et « Sélectionner les objets sources ».
— Manipuler les métadonnées et les annotations associées au signal actuel, grâce aux sous-menus « Métadonnées » et « Annotations » qui offrent les fonctionnalités suivantes.

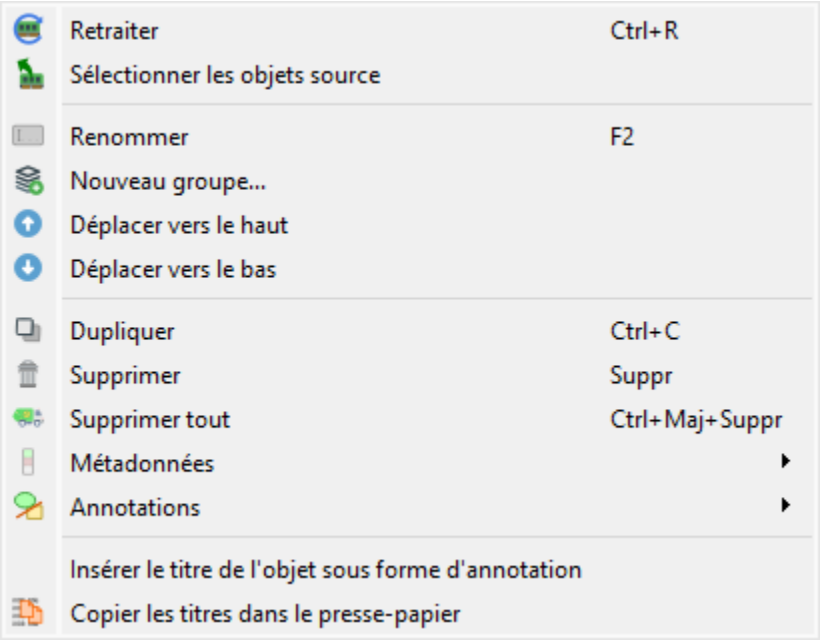


FIG. 19 – Capture d'écran du menu « Édition ».

Recalculer

L'action « Recalculer » vous permet de recalculer le signal (ou les signaux) sélectionné(s) en utilisant leurs paramètres de traitement d'origine. Cela est utile lorsque vous souhaitez réexécuter la chaîne de traitement qui a été utilisée pour créer un signal, par exemple après avoir modifié les paramètres globaux ou les dépendances.

Note : Cette action n'est disponible que pour les signaux qui ont été créés par des opérations de traitement et qui ont des paramètres de traitement stockés.

Sélectionner les objets sources

L'action « Sélectionner les objets sources » vous permet de sélectionner l'objet (ou les objets) sources qui ont été utilisés pour créer le signal actuellement sélectionné. Cela aide à retracer l'historique de traitement et à comprendre quels signaux d'origine ont été utilisés comme entrée pour le résultat actuel.

Note : Cette action n'est disponible que lorsqu'un seul signal est sélectionné et que ce signal possède des références d'objets sources.

Métadonnées

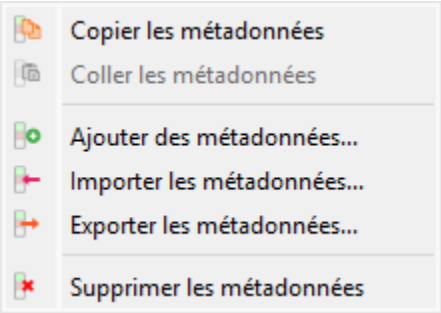


Fig. 20 – Capture d'écran du sous-menu « Métadonnées ».

Copier/coller les métadonnées

Compte tenu du fait que les métadonnées contiennent des informations utiles sur le signal, elles peuvent être copiées et collées d'un signal à un autre en sélectionnant les actions « Copier les métadonnées » et « Coller les métadonnées » dans le menu « Édition ».

Cette fonctionnalité vous permet de transférer ces informations d'un signal à un autre :

- *Régions d'intérêt (ROIs)* : c'est un moyen très efficace de réutiliser la même ROI sur différents signaux et de comparer facilement les résultats de l'analyse sur ces signaux
- Résultats de calcul, tels que les positions de pics ou les intervalles FHWM (la pertinence du transfert de ces informations dépend du contexte et revient à l'utilisateur de décider)
- Toute autre information que vous avez pu ajouter aux métadonnées d'un signal

---

Note : Copier les métadonnées d'un signal à un autre écrasera les métadonnées du signal de destination (pour les clés de métadonnées communes aux deux signaux) ou ajoutera simplement les clés de métadonnées qui ne sont pas présentes dans le signal de destination.

Importer/exporter les métadonnées

Les métadonnées peuvent également être importées et exportées depuis/vers un fichier JSON en utilisant les actions « Importer les métadonnées » et « Exporter les métadonnées » dans le menu « Édition ». C'est exactement la même chose que la fonctionnalité de copier/coller des métadonnées (voir ci-dessus pour plus de détails sur les cas d'utilisation de cette fonctionnalité), mais cela vous permet de sauvegarder les métadonnées dans un fichier et de les importer ultérieurement.

Supprimer les métadonnées

Lors de la suppression des métadonnées en utilisant l'action « Supprimer les métadonnées » dans le menu « Édition », vous serez invité à confirmer la suppression des régions d'intérêt (ROIs) si elles sont présentes dans les métadonnées. Après cette confirmation éventuelle, les métadonnées seront supprimées, ce qui signifie que les résultats d'analyse, les ROIs et toute autre information associée au signal seront perdus.

Ajouter des métadonnées

L'action « Ajouter des métadonnées » vous permet d'ajouter des éléments de métadonnées personnalisés à un ou plusieurs signaux sélectionnés. Cela est utile pour étiqueter les signaux avec des identifiants d'expérience, des noms d'échantillons, des étapes de traitement ou toute autre information personnalisée.

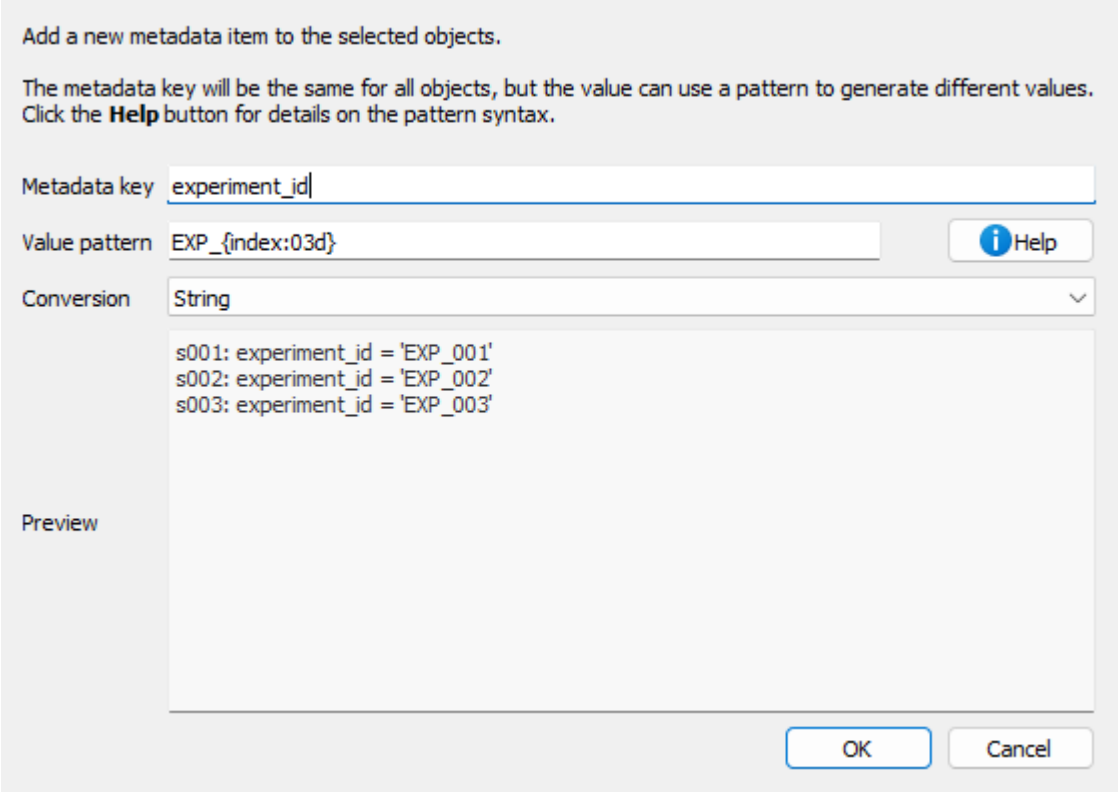


Fig. 21 – Boîte de dialogue Ajouter des métadonnées.

Lorsque vous sélectionnez « Ajouter des métadonnées… » dans le menu Édition, une boîte de dialogue apparaît où vous pouvez :

- **Clé de métadonnées** : Entrez le nom du champ de métadonnées à ajouter
- **Modèle de valeur** : Définissez un modèle pour la valeur des métadonnées en utilisant des chaînes de format Python
- **Conversion** : Choisissez comment stocker la valeur (chaîne, flottant, entier ou booléen)
- **Aperçu** : Voir comment les métadonnées seront ajoutées à chaque signal sélectionné.

Le modèle de valeur prend en charge les espaces réservés suivants :

- `{title}` : Titre du signal
- `{index}` : Index basé sur 1 du signal dans la sélection
- `{count}` : Nombre total de signaux sélectionnés
- `{xlabel}`, `{xunit}`, `{ylabel}`, `{yunit}` : Étiquettes et unités des axes
- `{metadata[key]}` : Accéder aux valeurs de métadonnées existantes

Vous pouvez également utiliser des modificateurs de format :

- `{title:upper}` : Convertir en majuscules
- `{title:lower}` : Convertir en minuscules
- `{index:03d}` : Formater en nombre à 3 chiffres avec des zéros devant

Exemples :

- Ajouter l'ID d'expérience : clé=``experiment_id``, modèle=``EXP_{index :03d}``, conversion=chaîne → Crée des métadonnées comme `experiment_id="EXP_001"`

— Ajouter la température de l'échantillon : clé=``temperature``, modèle=``{metadata[temp]}``, conversion=float → Copie la température des métadonnées existantes et la convertit en flottant
— Marquer les signaux traités : clé=``is_processed``, modèle=``true``, conversion=bool → Définit `is_processed=True` pour tous les signaux sélectionnés

Annotations

Les annotations sont des éléments visuels qui peuvent être ajoutés aux signaux pour mettre en évidence des fonctionnalités spécifiques, marquer des régions d'intérêt ou ajouter des notes explicatives. DataLab propose un sous-menu dédié dans le menu « Édition » pour gérer les annotations.

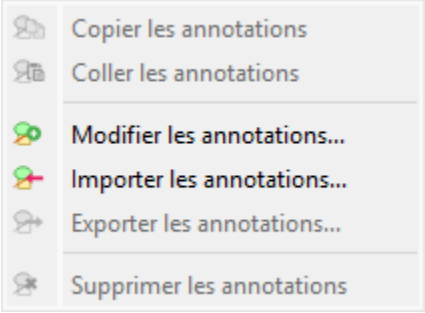


Fig. 22 – Capture d'écran du sous-menu « Annotations ».

Copier/coller les annotations

Les annotations peuvent être copiées d'un signal et collées dans un ou plusieurs autres signaux à l'aide des actions « Copier les annotations » et « Coller les annotations » . Cela est utile lorsque vous souhaitez appliquer les mêmes marqueurs visuels à plusieurs signaux.

L'action « Coller les annotations » n'est activée que lorsqu'il y a des annotations dans le presse-papiers (c'est-à-dire après avoir utilisé « Copier les annotations »).

Modifier les annotations

L'action « Modifier les annotations » ouvre une boîte de dialogue où vous pouvez afficher, ajouter, modifier ou supprimer des annotations du signal actuel. Cela fournit un moyen visuel de gérer toutes les annotations sur un signal.

Importer/exporter les annotations

Les annotations peuvent être enregistrées et chargées à partir de fichiers JSON (extension .dlabann) à l'aide des actions « Importer les annotations » et « Exporter les annotations » . Cela vous permet de :
— Enregistrer les ensembles d'annotations pour une réutilisation ultérieure
— Partager les annotations avec des collègues
— Archiver les annotations séparément des données du signal
— Appliquer les mêmes annotations à différents signaux au cours des sessions
L'action « Exporter les annotations » n'est disponible que lorsque le signal sélectionné a des annotations.

Supprimer les annotations

L'action « Supprimer les annotations » supprime toutes les annotations des signaux sélectionnés. Cette action n'est activée que lorsque les signaux sélectionnés ont des annotations.

Note : Les annotations sont stockées séparément des métadonnées et des résultats d'analyse. La suppression des annotations n'affecte pas les ROIs ou d'autres éléments de métadonnées.

Titres des signaux

Les titres des signaux peuvent être considérés comme des métadonnées du point de vue de l'utilisateur, même s'ils ne sont pas stockés dans les métadonnées du signal (mais dans un attribut de l'objet signal).

Le menu « Édition » vous permet de :

- « Ajouter le titre de l'objet au graphique » : cette action ajoutera une étiquette en haut du signal avec son titre.
- « Copier les titres dans le presse-papiers » : cette action copiera les titres des signaux sélectionnés dans le presse-papiers, ce qui peut être utile pour les coller dans un éditeur de texte ou dans un tableur.

  Exemple du contenu du presse-papiers :

```
g001:
    s001: lorentz(a=1,sigma=1,mu=0,ymin=0)
    s002: derivative(s001)
    s003: wiener(s002)
g002: derivative(g001)
    s004: derivative(s001)
    s005: derivative(s002)
    s006: derivative(s003)
g003: fft(g002)
    s007: fft(s004)
    s008: fft(s005)
    s009: fft(s006)
```

2.2.4 Régions d'intérêt (ROI)

Cette section décrit comment manipuler les régions d'intérêt (ROIs) pour les signaux dans DataLab.

Le menu « ROI » vous permet de gérer les régions d'intérêt (ROIs) associées au signal actuel.

Voir aussi :

Pour plus d'informations sur les métadonnées de signal, voir la section *Manipuler les métadonnées et les annotations*.

Les régions d'intérêt (ROI) sont des zones de signal définies par l'utilisateur pour effectuer des opérations, des traitements ou des analyses spécifiques sur elles.

Les ROI sont prises en compte presque partout dans les fonctionnalités de calcul de DataLab :

- Les fonctionnalités du menu « Opérations » sont effectuées uniquement sur la ROI si elle est définie (sauf si l'opération modifie le nombre de points, comme l'interpolation ou le rééchantillonnage).
- Les actions du menu « Traitement » sont effectuées uniquement sur la ROI si elle est définie (sauf si le type de données du signal de destination est différent de celui de la source, comme dans les fonctionnalités d'analyse de Fourier).
- Les actions du menu « Analyse » sont effectuées uniquement sur la ROI si elle est définie.

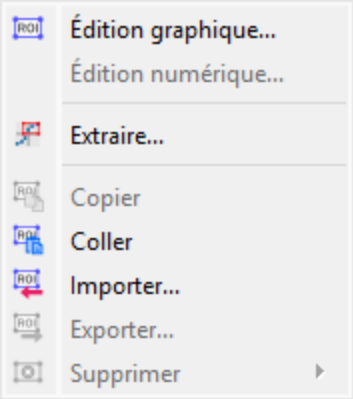


Fig. 23 – Capture d'écran du menu « ROI ».

Note : Les ROI sont stockées en tant que métadonnées et sont donc attachées au signal.

Le menu « ROI » vous permet de :

- « Édition graphique » : ouvrir une boîte de dialogue pour gérer les ROI associées au signal sélectionné (ajouter, supprimer, déplacer, redimensionner, etc.). La boîte de dialogue de définition des ROI est exactement la même que celle de l'extraction des ROI (voir ci-dessous) : la ROI est définie en déplaçant la position et en ajustant la largeur d'une plage horizontale.
- « Édition numérique » : ouvrir une boîte de dialogue pour modifier les paramètres des ROI sélectionnées numériquement (c'est-à-dire à l'aide d'un formulaire simple). Cela vous permet de définir ou de modifier des ROI en fonction de valeurs numériques.
- « Extraire » : extraire la ROI définie des signaux sélectionnés. Cela créera un nouveau signal pour chaque ROI (ou un seul signal, si l'option « Extraire toutes les ROIs dans un seul signal » est sélectionnée dans la boîte de dialogue), avec les mêmes métadonnées que le signal d'origine, mais avec les données correspondant uniquement à la ROI. Les nouveaux signaux seront ajoutés à l'espace de travail actuel.
- « Copier » : copier la ROI définie des signaux sélectionnés dans le presse-papiers. Cela vous permet de coller la ROI dans un autre signal ou de l'utiliser dans d'autres opérations.
- « Coller » : coller la ROI copiée du presse-papiers dans les signaux sélectionnés. Cela ajoutera la ROI aux signaux, vous permettant de définir ou de modifier des ROI en fonction de celles précédemment copiées.
- « Importer » : importer des ROI à partir d'un fichier. Cela vous permet de charger des ROI précédemment enregistrées dans le signal actuel.
- « Exporter » : exporter les ROI définies vers un fichier. Cela vous permet de sauvegarder les ROI pour une utilisation ultérieure ou de les partager avec d'autres.
- « Supprimer tout » : supprimer toutes les ROI définies pour les signaux sélectionnés.

2.2.5 Opérations sur les signaux

Cette section décrit les opérations qui peuvent être effectuées sur les signaux.

Voir aussi :

Traitement des signaux pour plus d'informations sur les fonctionnalités de traitement des signaux, ou *Analyse sur les signaux* pour des informations sur les fonctionnalités d'analyse des signaux.

Lorsque le « Panneau Signal » est sélectionné, les menus et barres d'outils sont mis à jour pour fournir les actions liées aux signaux.

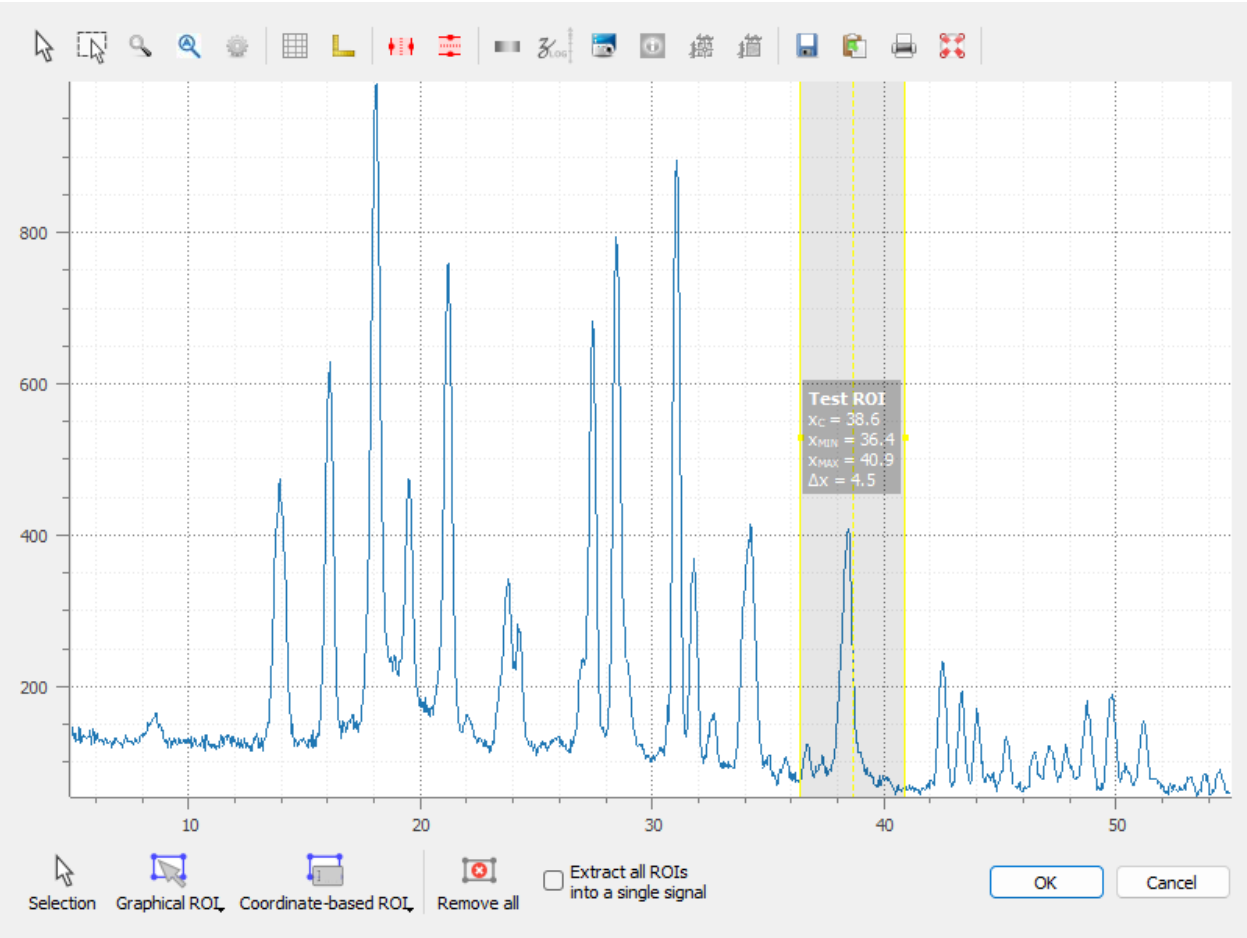


FIG. 24 – Un signal avec une ROI.

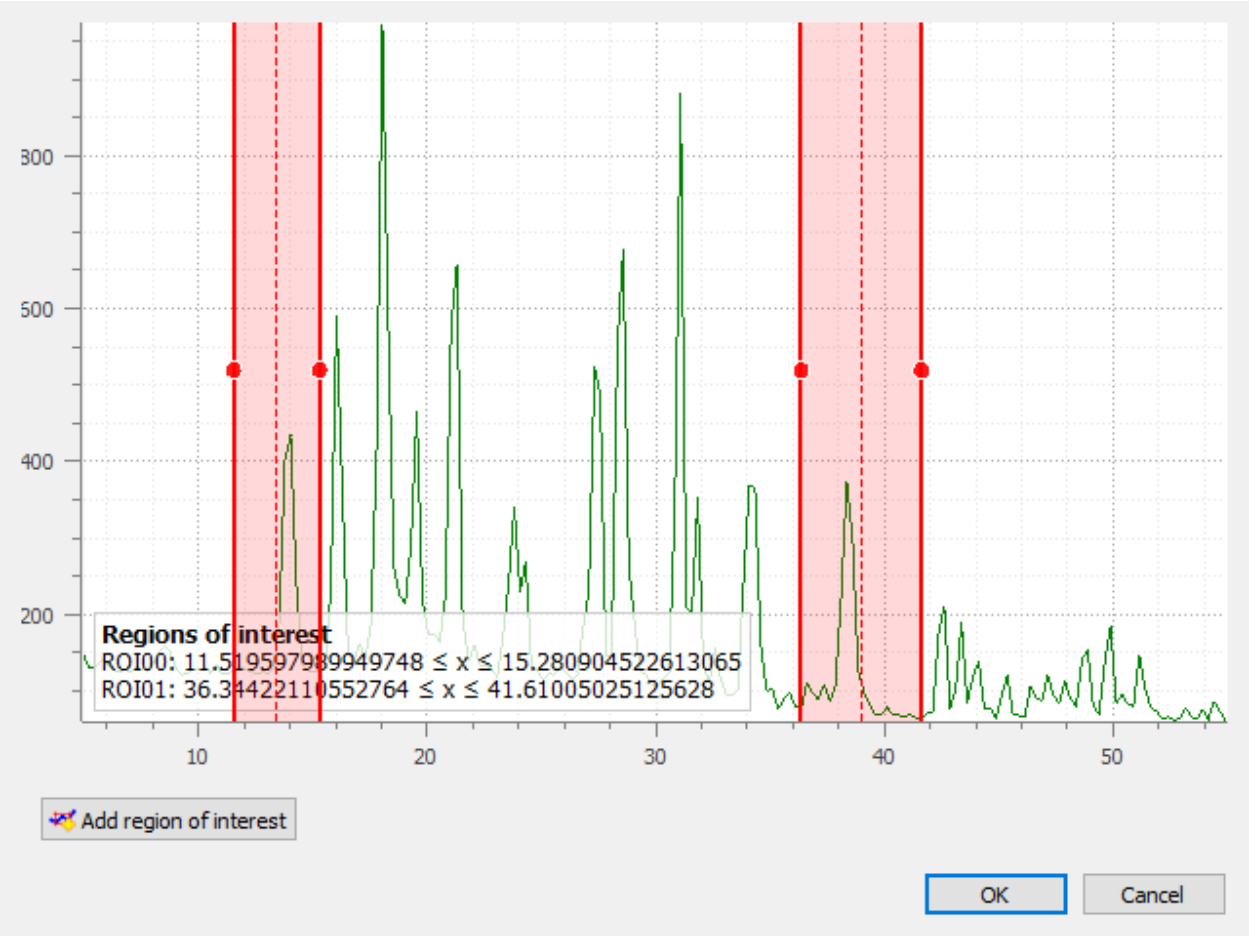


FIG. 25 – Boîte de dialogue d'extraction des ROI : la ROI est définie en déplaçant la position et en ajustant la largeur d'une plage horizontale.

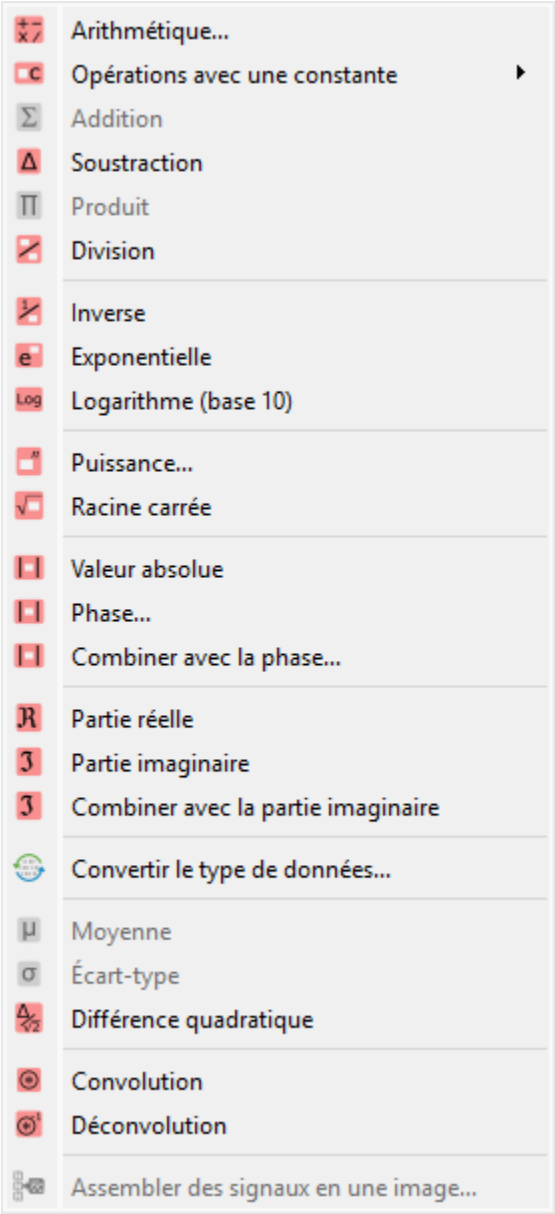


FIG. 26 – Capture d'écran du menu « Opérations ».

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.

Opérations avec une constante

Crée un signal à partir d'une opération avec une constante sur chaque signal sélectionné :

Opération	Description
Addition	$y_k = y_{k-1} + c$
Soustraction	$y_k = y_{k-1} - c$
Multiplication	$y_k = y_{k-1} \times c$
Division	$y_k = \frac{y_{k-1}}{c}$

Opérations arithmétiques de base

Opération	Description
Somme	$y_M = \sum_{k=0}^{M-1} y_k$
Différence	$y_2 = y_1 - y_0$
Produit	$y_M = \prod_{k=0}^{M-1} y_k$
Division	$y_2 = \frac{y_1}{y_0}$
Inverse	$y_2 = \frac{1}{y_1}$
Exponentielle	$y_2 = \exp(y_1)$
Logarithme (base 10)	$y_2 = \log_{10}(y_1)$

Convolution et Déconvolution

Opération	Implémentation
Convolution	Basée sur scipy.signal.convolve
Déconvolution	Déconvolution dans le domaine fréquentiel

Valeur absolue et opérations sur les signaux complexes

Opération	Description
Valeur absolue	$y_k = \lvert y_{k-1} \rvert$
Phase (argument)	`sigima.proc.signal.phase()`
Combiner avec la phase	Considère le signal courant comme le module et permet de sélectionner un signal représentant la phase pour les fusionner en un signal complexe `sigima.proc.signal.complex_from_magnitude_phase()`
Partie réelle	$y_k = \Re(y_{k-1})$
Partie imaginaire	$y_k = \Im(y_{k-1})$
Combiner avec la partie imaginaire	Considère le signal courant comme la partie réelle et permet de sélectionner un signal représentant la partie imaginaire pour les fusionner en un signal complexe `sigima.proc.signal.complex_from_real_imag()`

Conversion du type de données

L'action « Convertir le type de données » permet de convertir le type de données des signaux sélectionnés.

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

Statistiques entre les signaux

Crée un nouveau signal à partir d'une opération statistique sur chaque point des signaux sélectionnés :

Opération	Description
Moyenne	$y_M = \frac{1}{M} \sum_{k=0}^{M-1} y_k$
Écart-type	$y_M = \sqrt{\frac{1}{M} \sum_{k=0}^{M-1} (y_k - \bar{y})^2}$

Autres fonctions mathématiques

Fonction	Description
Puissance	$y_k = y_k^n$
Racine carrée	$y_k = \sqrt{y_k}$
Dérivée	Basée sur numpy.gradient
Intégrale	Basée sur scipy.integrate.cumulative_trapezoid
Assembler les signaux en une image	Créer une image 2D en assemblant les signaux 1D sélectionnés en lignes ou colonnes, avec une normalisation optionnelle.

2.2.6 Traitement des signaux

Cette section décrit les fonctionnalités de traitement de signal disponibles dans DataLab.

Voir aussi :

Opérations sur les signaux pour plus d'informations sur les opérations qui peuvent être effectuées sur les signaux, ou *Analyse sur les signaux* pour des informations sur les fonctionnalités d'analyse des signaux.

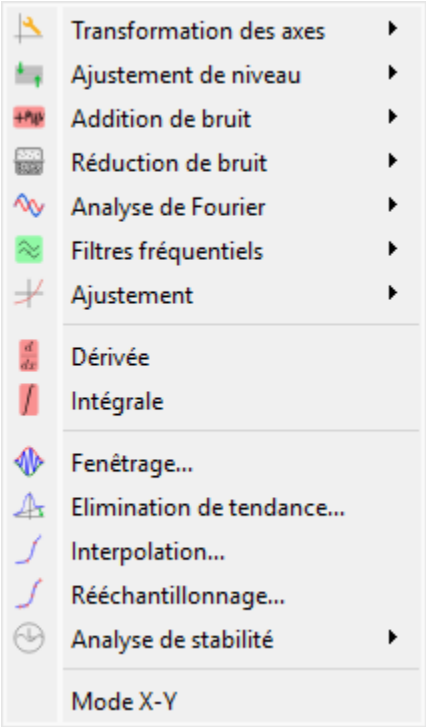


FIG. 27 – Capture d'écran du menu « Traitement ».

Lorsque le « Panneau Signal » est sélectionné, les menus et barres d'outils sont mis à jour pour fournir les actions liées aux signaux.

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

Transformation des axes

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	$y_1 = a.y_0 + b$

Permuter les axes X/Y

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

Inverser l'axe des X

Crée un signal à partir des données inversées de l'axe des X du signal sélectionné.

Convertir en coordonnées cartésiennes

Crée un signal en convertissant les coordonnées cartésiennes en coordonnées polaires.

Cette fonction suppose que l'axe des x représente le rayon et l'axe des y l'angle.

Avertissement : Un rayon ne peut pas être négatif. Toute valeur négative est écrêtée à 0.

Convertir en coordonnées polaires

Crée un signal en convertissant les coordonnées cartésiennes en coordonnées polaires.

Ajustement des niveaux

Normaliser

Crée un signal à partir de la normalisation de chaque signal sélectionné par maximum, amplitude, somme, énergie ou RMS :

Paramètre	Normalisation
Maximum	$y_1 = \frac{y_0}{\max(y_0)}$
Amplitude	$y_1 = \frac{y'_0}{\max(y'_0)}$ with $y'_0 = y_0 - \min(y_0)$
Aire	$y_1 = \frac{y_0}{\sum_{n=0}^{N} y_0[n]}$
Energie	$y_1 = \frac{y_0}{\sqrt{\sum_{n=0}^{N} \lvert y_0[n]\rvert^2}}$
RMS	$y_1 = \frac{y_0}{\sqrt{\frac{1}{N}\sum_{n=0}^{N} \lvert y_0[n]\rvert^2}}$

Ecrêtage

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

Soustraction d'offset

Crée un signal à partir de la soustraction d'offset de chaque signal sélectionné. Cette opération est réalisée en soustrayant la valeur moyenne d'une plage définie par l'utilisateur.

Ajout de bruit

Génère de nouveaux signaux en ajoutant le même bruit à chaque signal sélectionné. Les types de bruit disponibles sont :

Bruit	Description
Gaussien	Distribution normale
Uniforme	Distribution uniforme
Poisson	Distribution de Poisson

Réduction de bruit

Crée un signal à partir du débruitage de chaque signal sélectionné.

Les filtres suivants sont disponibles :

Filtre	Formule/implémentation
Filtre gaussien	scipy.ndimage.gaussian_filter
Moyenne mobile	scipy.ndimage.uniform_filter
Médiane mobile	scipy.ndimage.median_filter
Filtre de Wiener	scipy.signal.wiener

Analyse de Fourier

Complément de zéro

Crée un signal à partir du complément de zéro de chaque signal sélectionné.

Les paramètres suivants sont disponibles :

Paramètre	Description
Stratégie	Stratégie de complément de zéro (voir ci-dessous)
Nombre de points	Longueur personnalisée (si *strategy* est « custom »)

La stratégie de complément de zéro fait référence à la méthode utilisée pour augmenter la longueur du signal, et elle peut être l'une des suivantes :

Stratégie	Description
next_pow2	Puissance de 2 supérieure (e.g. 1024 → 2048)
double	Double la longueur (e.g. 1024 → 2048)
triple	Triple la longueur (e.g. 1024 → 3072)
custom	Longueur personnalisée (définie par l'utilisateur)

Fonctions liées à la FFT

Crée un signal à partir de l'analyse de Fourier de chaque signal sélectionné.

Les fonctions suivantes sont disponibles :

Fonction	Description	Formule/implémentation
FFT	Transformée de Fourier rapide	numpy.fft.fft
FFT inverse	Transformée de Fourier rapide inverse	numpy.fft.ifft
Spectre d'amplitude	Optionnel : sortie en décibels (dB)	$y_1 = \lvert\mathrm{FFT}\,(y_0)\rvert$ ou $y_1 = 20\log_{10}(\lvert\mathrm{FFT}\,(y_0)\rvert)$ (dB)
Spectre de phase	Phase de la FFT exprimée en degrés, utilisant la fonction numpy.angle	$y_1 = \angle\,\mathrm{FFT}\,(y_0)$
Densité spectrale de puissance (PSD)	Optionnel : sortie en décibels (dB). La PSD est estimée en utilisant la méthode de Welch (voir scipy.signal.welch)	$y_1 = \mathrm{PSD}\,(y_0)$ ou $y_1 = 10\log_{10}(\mathrm{PSD}\,(y_0))$ (dB)

Note : La FFT et la FFT inverse sont effectuées en utilisant un décalage de fréquence si l'option est activée dans les paramètres de DataLab (voir *Préférences*).

Filtres fréquentiels

Crée un signal à partir de l'application d'un filtre fréquentiel à chaque signal sélectionné.

Les filtres suivants sont disponibles :

Filtre	Description
Low-pass	Filtre les hautes fréquences, au-dessus d'une fréquence de coupure
High-pass	Filtre les basses fréquences, en-dessous d'une fréquence de coupure
Band-pass	Filtre les fréquences en dehors d'une plage
Band-stop	Filtre les fréquences à l'intérieur d'une plage

Pour chaque filtre, les méthodes suivantes sont disponibles :

Méthode	Description
Bessel	Filtre de Bessel, utilisant la fonction scipy.signal.bessel
Filtre idéal	Filtre idéal (rectangulaire) dans le domaine fréquentiel
Butterworth	Filtre de Butterworth, utilisant la fonction scipy.signal.butter
Chebyshev I	Filtre de Chebyshev de type I, utilisant la fonction scipy.signal.cheby1
Chebyshev II	Filtre de Chebyshev de type II, utilisant la fonction scipy.signal.cheby2
Elliptic	Filtre elliptique, utilisant la fonction scipy.signal.ellip

Ajustement

Ajuste un modèle à chaque signal sélectionné. Cela peut être fait automatiquement ou via une boîte de dialogue d'ajustement de courbe interactive.

Modèle	Equation
Linéaire	$y = c_0 + c_1 \cdot x$
Polynomial	$y = c_0 + c_1 \cdot x + c_2 \cdot x^2 + ... + c_n \cdot x^n$
Gaussien	$y = y_0 + \frac{A}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2}\left(\frac{x - x_0}{\sigma}\right)^2\right)$
Lorentzien	$y = y_0 + \frac{A}{\pi\sigma} \cdot \frac{1}{1 + \left(\frac{x - x_0}{\sigma}\right)^2}$
Voigt	$y = y_0 + A \cdot \frac{\Re\left(\exp(-z^2) \cdot \mathrm{erfc}(-j \cdot z)\right)}{\sqrt{2\pi}\sigma}$ avec $z = \frac{x - x_0 - j \cdot \sigma}{\sqrt{2}\sigma}$
Multi-gaussien	$y = y_0 + \sum_{i=0}^{N} \frac{A_i}{\sqrt{2\pi}\sigma_i} \exp\left(-\frac{1}{2}\left(\frac{x - x_{0,i}}{\sigma_i}\right)^2\right)$
Multi-lorentzien	$y = y_0 + \sum_{i=0}^{N} \frac{A_i}{\pi\sigma_i} \cdot \frac{1}{1 + \left(\frac{x - x_{0,i}}{\sigma_i}\right)^2}$
Planck	$y = y_0 + A \cdot \frac{2h \cdot c^2}{\lambda^5} \cdot \left(\exp\left(\frac{h \cdot c}{\lambda \cdot k \cdot T}\right) - 1\right)^{-1}$
Deux demi-gaussiennes	$y = y_0 + A \cdot \frac{1}{\sqrt{2\pi}\sigma_0} \cdot \exp\left(-\frac{1}{2}\left(\frac{x - x_0}{\sigma_0}\right)^2\right)$ si $x < x_0$ $y = y_0 + A \cdot \frac{1}{\sqrt{2\pi}\sigma_1} \cdot \exp\left(-\frac{1}{2}\left(\frac{x - x_1}{\sigma_1}\right)^2\right)$ sinon
Deux exponentielles dos à dos	$y = y_0 + A_1 \cdot \exp\left(-x/\tau_1\right)$ si $x < 0$ $y = y_0 + A_2 \cdot \exp(-x/\tau_2)$ sinon
Exponentielle	$y = y_0 + A \exp\left(B \cdot x\right)$
Sinusoïdal	$y = y_0 + A \sin\left(2\pi f \cdot x + \phi\right)$
Fonction de distribution cumulative (CDF)	$y = y_0 + A\,\mathrm{erf}\left(\frac{x - x_0}{\sqrt{2}\sigma}\right)$

Fenêtrage

Crée un signal à partir de l'application d'une fonction de fenêtrage à chaque signal sélectionné.

Les fonctions de fenêtrage suivantes sont disponibles :

Fonction de fenêtrage	Référence
Barthann	`scipy.signal.windows.barthann()`
Bartlett	`numpy.bartlett()`
Blackman	`scipy.signal.windows.blackman()`
Blackman-Harris	`scipy.signal.windows.blackmanharris()`
Bohman	`scipy.signal.windows.bohman()`
Boîte (rectangulaire)	`scipy.signal.windows.boxcar()`
Cosinus	`scipy.signal.windows.cosine()`
Exponentielle	`scipy.signal.windows.exponential()`
Flat top	`scipy.signal.windows.flattop()`
Gaussien	`scipy.signal.windows.gaussian()`
Hamming	`numpy.hamming()`
Hann	`numpy.hanning()`
Kaiser	`scipy.signal.windows.kaiser()`
Lanczos	`scipy.signal.windows.lanczos()`
Nuttall	`scipy.signal.windows.nuttall()`
Parzen	`scipy.signal.windows.parzen()`
Taylor	`scipy.signal.windows.taylor()`
Tukey	`scipy.signal.windows.tukey()`

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.

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

Analyse de stabilité

Crée un signal à partir de l'analyse de stabilité de chaque signal

Les méthodes d'analyse de stabilité suivantes sont disponibles :

Fonction	Description
Variance d'Allan	Mesure de la stabilité d'un signal : définie comme la variance de la différence entre deux mesures successives en fonction de l'intervalle de temps entre elles.
Ecart-type d'Allan	Racine carrée de la variance d'Allan.
Ecart-type d'Allan avec recouvrement	Une version plus robuste de la variance d'Allan qui recoupe les segments successifs pour améliorer la confiance statistique.
Variance d'Allan modifiée	Une variation de la variance d'Allan qui tient compte du bruit de phase en introduisant une opération de filtrage.
Variance de Hadamard	Une alternative à la variance d'Allan, plus robuste aux dérives de fréquence linéaires dans le signal.
Variance totale	Etend le concept de variance d'Allan pour couvrir tous les intervalles de moyenne possibles.
Déviation temporelle	Dérivée de la déviation d'Allan, quantifie la stabilité en termes de temps plutôt que de fréquence.

Note : L'option « Toutes les analyses de stabilité » permet de calculer toutes les méthodes d'analyse de stabilité en une seule fois.

Mode XY

Simuler le mode XY d'un oscilloscope.

2.2.7 Analyse sur les signaux

Cette section décrit les fonctionnalités d'analyse de signaux disponibles dans DataLab.

Voir aussi :

Opérations sur les signaux pour plus d'informations sur les opérations qui peuvent être effectuées sur les signaux, ou *Traitement des signaux* pour des informations sur les fonctionnalités de traitement des signaux.

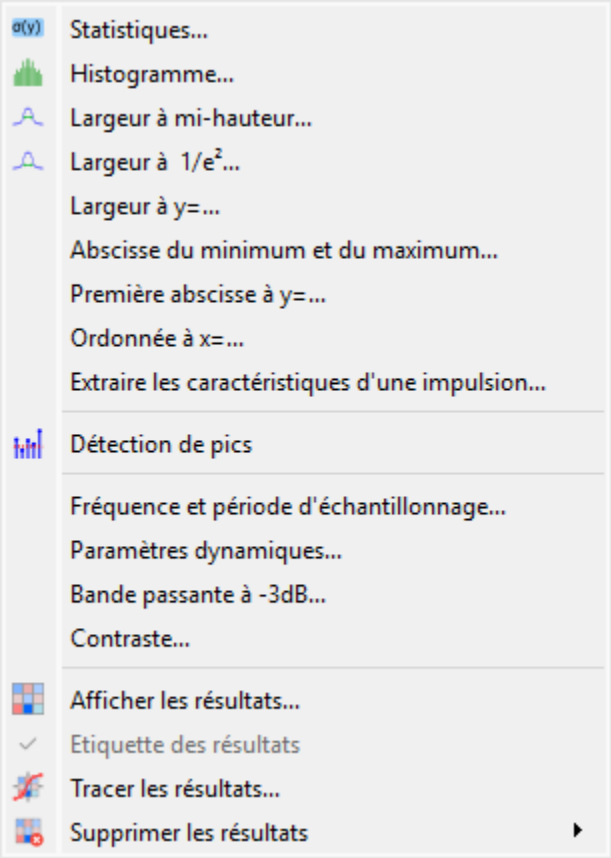


Fig. 28 – Capture d'écran du menu Analyse ».

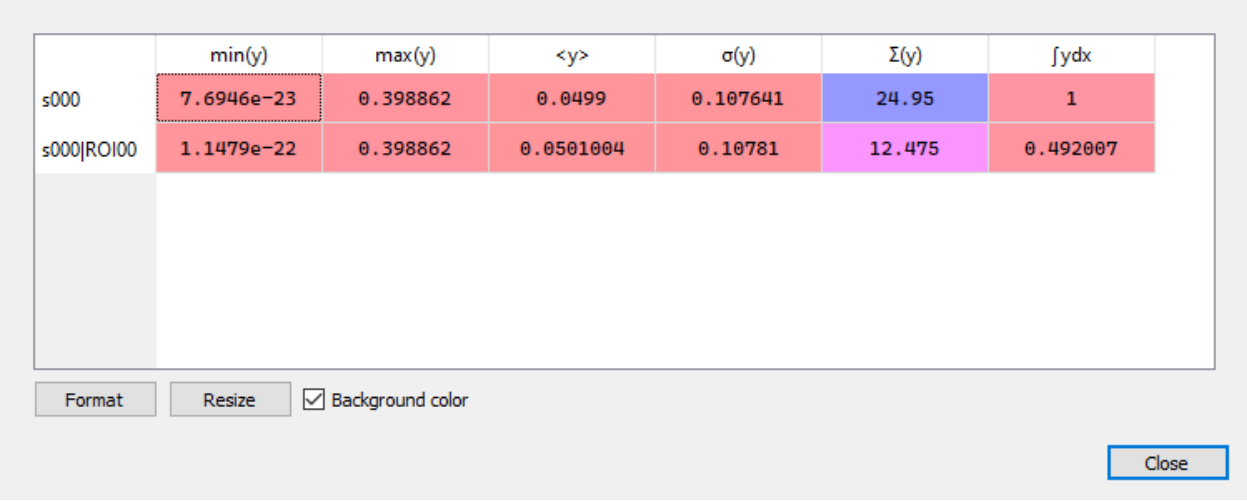
Lorsque le « Panneau Signal » est sélectionné, les menus et barres d'outils sont mis à jour pour fournir les actions liées aux signaux.

Le menu Analyse » 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 « analyse » 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.

Statistiques

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



	min(y)	max(y)	<y>	σ(y)	Σ(y)	∫ydx
s000	7.6946e-23	0.398862	0.0499	0.107641	24.95	1
s000|ROI00	1.1479e-22	0.398862	0.0501004	0.10781	12.475	0.492007

Fig. 29 – 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

Calcule la largeur à mi-hauteur (LMH) du signal sélectionné, en utilisant l'une des méthodes suivantes :

Méthode	Description
Passage par zéro	Recherche les passages par zéro du signal après avoir centré son amplitude autour de zéro
Gauss	Ajuste les données à un modèle gaussien en utilisant un algorithme de moindres carrés
Lorentz	Ajuste les données à un modèle lorentzien en utilisant un algorithme de moindres carrés
Voigt	Ajuste les données à un modèle de Voigt en utilisant un algorithme de moindres carrés

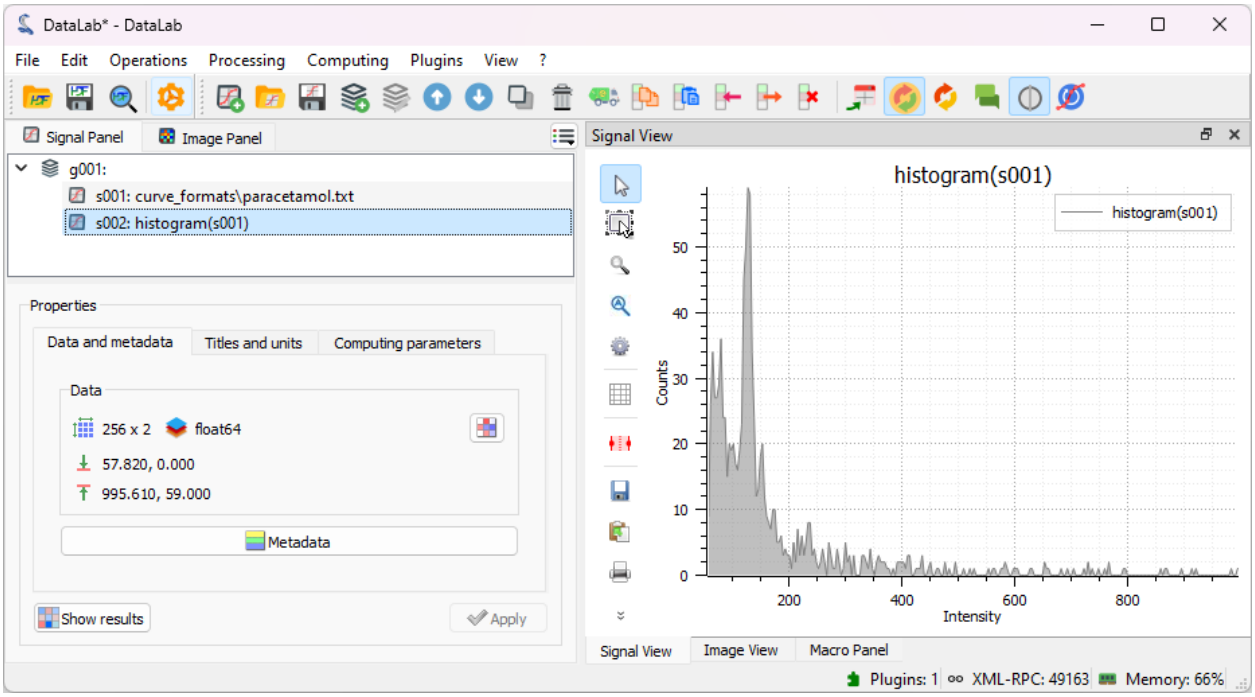


FIG. 30 – Exemple d'histogramme.

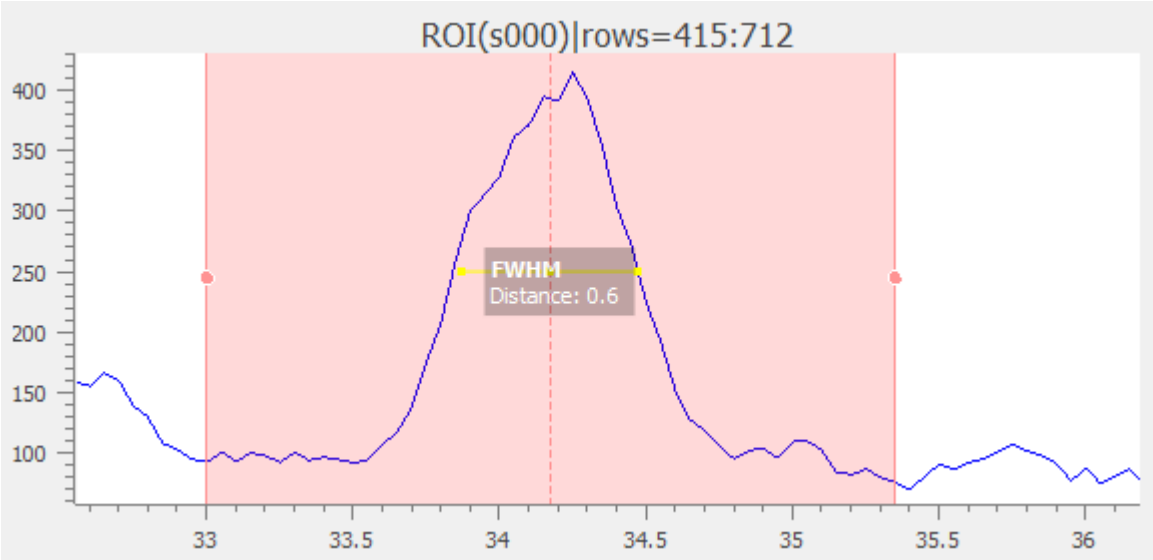


FIG. 31 – Le résultat du calcul est affiché sous la forme d'un segment annoté.

Largeur à 1/e^2

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

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

Abscisse du minimum et du maximum

Calcule le plus petit argument du minimum et le plus petit argument du maximum du signal sélectionné.

Abscisse pour y=...

Calcule l'abscisse pour une ordonnée donnée du signal sélectionné. Si aucune solution n'est trouvée, le résultat affiché est NaN. Si plusieurs solutions existent, le résultat affiché est la plus petite valeur.

Détection de pics

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

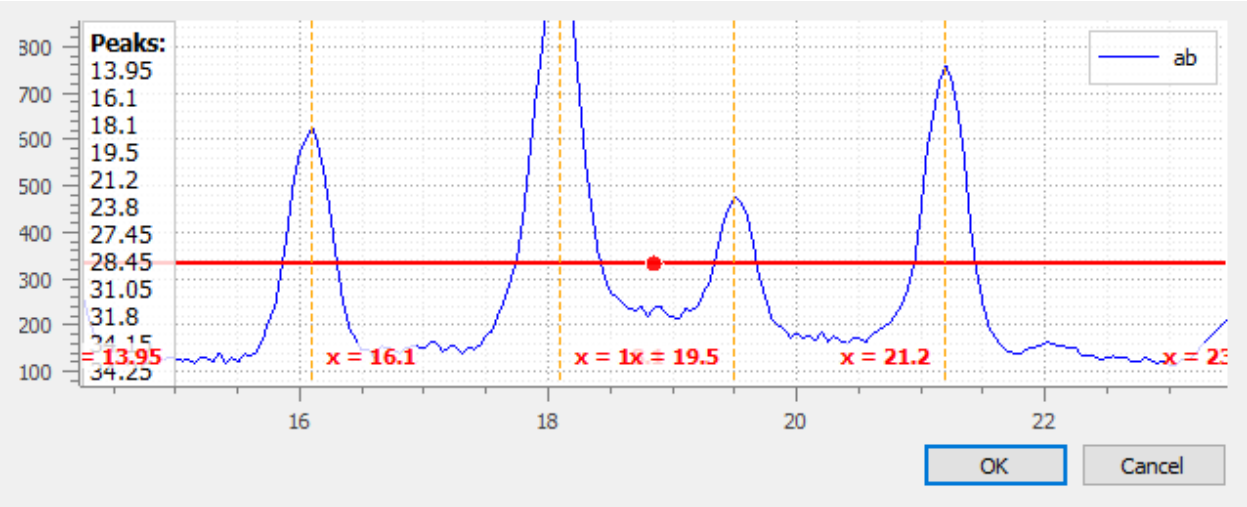


Fig. 32 – Boîte de dialogue de détection de pics : le seuil de détection est ajustable en déplaçant le curseur horizontal, les pics sont détectés automatiquement (des marqueurs verticaux indiquent les pics détectés avec leur position)

Fréquence et période d'échantillonnage

Calcule la fréquence et la période d'échantillonnage du signal sélectionné.

Avertissement : Cette fonctionnalité suppose que les valeurs de X sont régulièrement espacées.

Paramètres dynamiques

Calcule les paramètres dynamiques suivants sur le signal sélectionné :

Paramètre	Description
f	Fréquence (ajustement sinusoïdal)
ENOB	Bits effectifs
SNR	Rapport signal/bruit
SINAD	Rapport signal/bruit et distorsion
THD	Distorsion harmonique totale
SFDR	Dynamique sans distorsion

Bande passante à -3 dB

Détermine la bande passante à -3 dB en identifiant l'intervalle des valeurs d'abscisse où le signal reste supérieur à sa valeur maximale moins 3 dB.

Avertissement : Cette fonctionnalité nécessite que le signal soit exprimé en décibels (dB).

Contraste

Calcule le contraste du signal sélectionné.

Le contraste est défini comme le rapport de la différence et de la somme des valeurs maximale et minimale :

$$\text{Contrast} = \frac{\max(y) - \min(y)}{\max(y) + \min(y)}$$

Note : Cette fonctionnalité suppose que le signal est un profil extrait d'une image, pour laquelle le contraste a un sens. Cela justifie la définition optique du contraste.

Extraction des caractéristiques d'impulsions

Effectue une analyse complète des impulsions sur les signaux sélectionnés, en extrayant automatiquement les caractéristiques temporelles et d'amplitude pour les signaux d'échelon et d'impulsions carrées.

Cette fonctionnalité fournit une caractérisation automatisée des impulsions avec reconnaissance intelligente du type de signal et extraction robuste de paramètres pour l'analyse de signaux numériques, les mesures d'oscilloscope et la validation temporelle des impulsions.

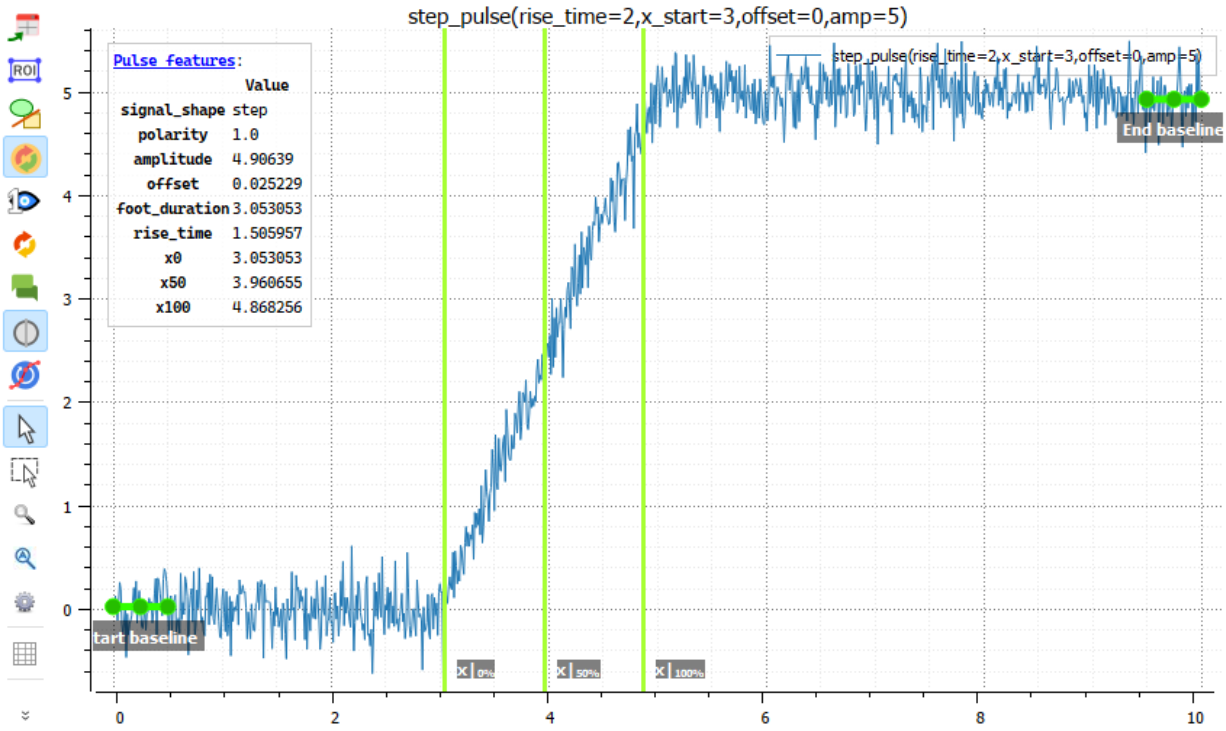


FIG. 33 – Résultats des caractéristiques d'impulsions tels qu'affichés dans la vue Signal.

Fonctionnalités clés :

- **Reconnaissance automatique du signal** : Identifie de façon heuristique le type de signal (échelon, carré ou autre) pour une analyse optimale
- **Détection de polarité** : Détermine automatiquement la polarité positive/négative de l'impulsion en utilisant la comparaison de ligne de base
- **Mesures temporelles complètes** : Extrait le temps de montée, le temps de descente, la largeur à mi-hauteur (FWHM) et les paramètres temporels à des fractions d'amplitude spécifiques
- **Caractérisation de la ligne de base** : Analyse les régions de ligne de base de début et de fin pour un calcul précis des caractéristiques
- **Algorithmes robustes** : Utilise des méthodes statistiques avancées avec tolérance au bruit et gestion d'erreurs

Paramètres :

Paramètre	Description
Forme du signal	Sélection du type de signal : Auto (détection automatique), Échelon ou Carré
Ligne de base début min	Limite X inférieure pour la région de ligne de base de début (initiale)
Ligne de base début max	Limite X supérieure pour la région de ligne de base de début (initiale)
Ligne de base fin min	Limite X inférieure pour la région de ligne de base de fin (finale)
Ligne de base fin max	Limite X supérieure pour la région de ligne de base de fin (finale)
Temps de montée/descente	Niveaux de référence pour la mesure du temps de montée/descente avec des choix prédéfinis : 5%-95% (Haute précision), 10%-90% (Standard IEEE), 20%-80% (Signaux bruités), 25%-75% (Alternative)

Caractéristiques extraites :

L'analyse calcule les caractéristiques d'impulsion suivantes :

Caractéristique	Description
Forme du signal	Type de signal détecté (ÉCHELON, CARRÉ ou AUTRE)
Polarité	Polarité du signal (+1 pour les impulsions positives, -1 pour les négatives)
Amplitude	Amplitude crête-à-crête de l'impulsion
Décalage	Décalage DC (niveau de ligne de base)
Temps de montée	Temps entre les niveaux de référence bas et haut pendant le front montant
Temps de descente	Temps entre les niveaux de référence haut et bas pendant le front descendant (impulsions carrées uniquement)
FWHM	Largeur à mi-hauteur (impulsions carrées uniquement)
x50	Temps auquel le signal atteint 50 % de l'amplitude maximale
x100	Temps auquel le signal atteint 100 % de l'amplitude maximale (début du plateau)
Durée du pied	Durée de la région plate avant la montée de l'impulsion
Plages de ligne de base	Coordonnées des limites de ligne de base de début et de fin extraites

Note : Les résultats sont affichés dans un tableau complet montrant tous les paramètres extraits, et sous forme d'annotations visuelles sur le tracé du signal. Pour les signaux d'échelon, les paramètres liés à la descente (fall_time, fwhm) ne sont pas applicables et s'affichent comme None. Cette fonctionnalité fonctionne au mieux avec des signaux d'impulsion bien définis qui ont des régions de ligne de base claires.

Afficher les résultats

Affiche les résultats de toutes les analyses effectuées sur les signaux sélectionnés. Cela affiche le même tableau que celui affiché après avoir effectué un calcul.

Étiquette des résultats

Active ou désactive la visibilité des étiquettes de résultats sur le graphique. Lorsqu'il est activé, cet élément de menu cochable affiche les annotations des résultats (telles que la largeur à mi-hauteur, les positions des pics, ou d'autres marqueurs d'analyse) directement sur le graphique du signal.

Cette option est synchronisée entre les panneaux Signal et Image et persiste entre les sessions. Elle n'est activée que lorsque des résultats sont disponibles pour le signal sélectionné.

Tracer les résultats

Trace les résultats des analyses effectuées 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).

2.2.8 Options d'affichage pour les signaux

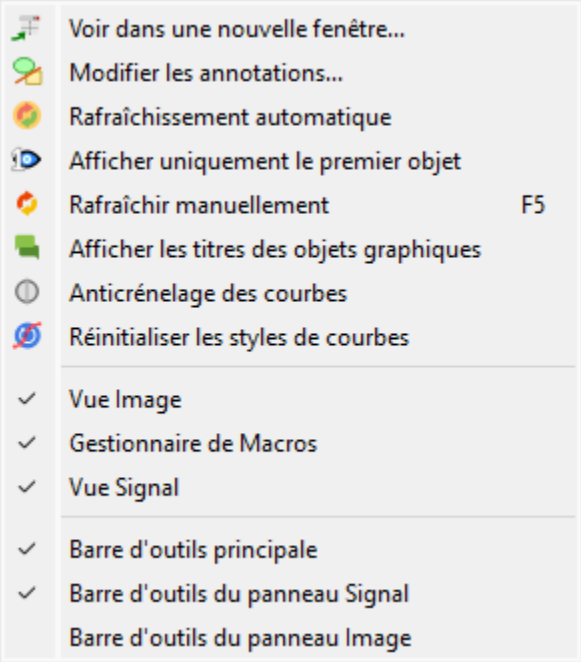


FIG. 34 – Capture d'écran du menu « Affichage ».

Lorsque le « Panneau Signal » est sélectionné, les menus et barres d'outils sont mis à jour pour fournir les actions liées aux signaux.

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 sélectionnés.

Cette option vous permet de visualiser les signaux sélectionnés dans une fenêtre séparée, dans laquelle vous pouvez visualiser les données plus confortablement (par exemple, en maximisant la fenêtre) et vous pouvez également annoter les données.

Lorsque vous cliquez sur le bouton « Annotations » dans la barre d'outils de la nouvelle fenêtre, le mode d'édition des annotations est activé (voir la section « Modifier les annotations » ci-dessous).

Modifier les annotations

Ouvre une nouvelle fenêtre pour modifier les annotations.

Cette option vous permet d'afficher les signaux sélectionnés dans une fenêtre séparée, dans laquelle vous pouvez visualiser les données plus confortablement (par exemple, en maximisant la fenêtre) et ajouter ou modifier des annotations.

Les annotations sont utilisées pour ajouter des commentaires ou des étiquettes aux données, et elles peuvent être utilisées pour mettre en évidence des événements spécifiques ou des régions d'intérêt.

Note : Les annotations sont enregistrées dans les métadonnées du signal, elles sont donc persistantes et seront affichées chaque fois que vous visualiserez le signal.

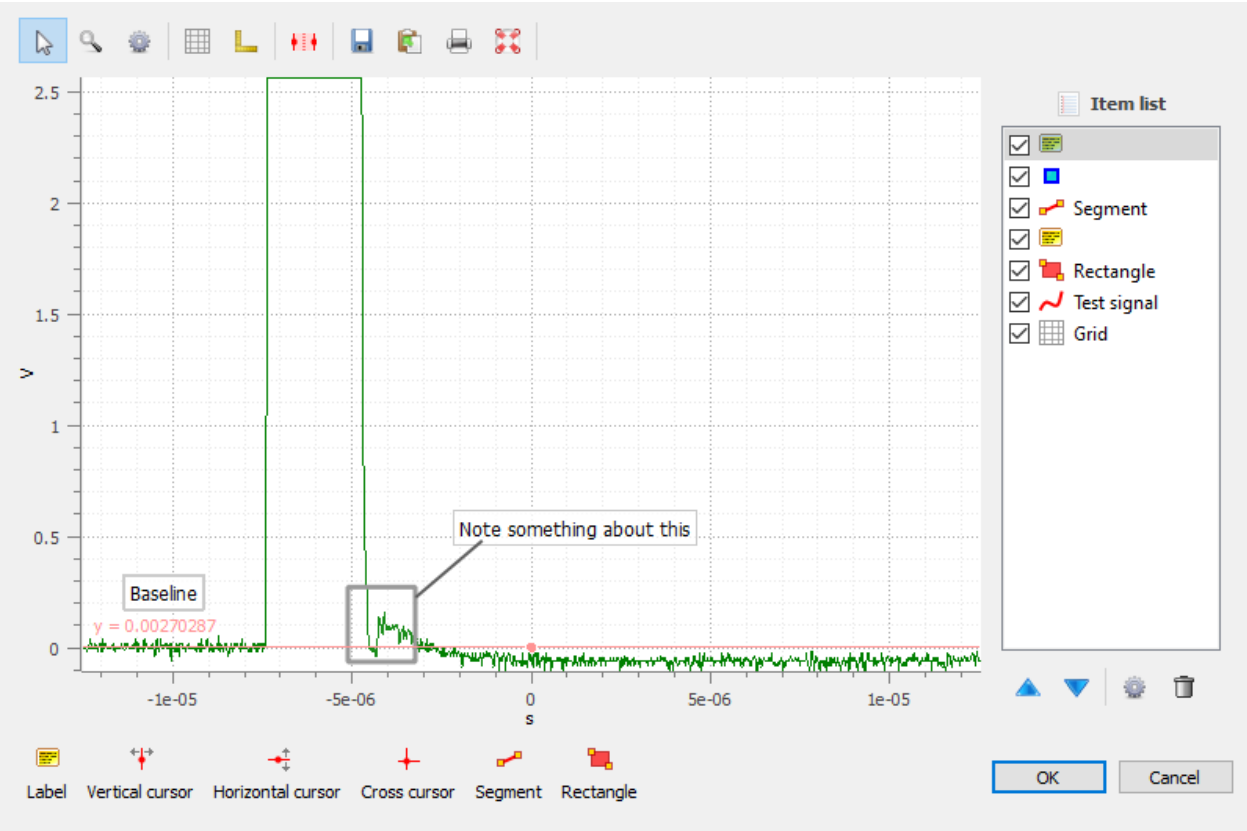


FIG. 35 – Les annotations peuvent être ajoutées dans la vue séparée.

Le mode opératoire typique pour modifier les annotations est le suivant :

1. Sélectionnez l'option « Modifier les annotations ».
2. Dans la nouvelle fenêtre, ajoutez des annotations en cliquant sur les boutons correspondants en bas de la fenêtre.
3. Personnalisez les annotations en modifiant leurs propriétés (par exemple, le texte, la couleur, la position, etc.) en utilisant l'option « Paramètres » dans le menu contextuel des annotations.
4. Quand vous avez terminé, cliquez sur le bouton « OK » pour enregistrer les annotations. Cela fermera la fenêtre et les annotations seront enregistrées dans les métadonnées du signal et seront affichées dans la fenêtre principale.

Note : Les annotations peuvent être copiées d'un signal à un autre en utilisant les fonctionnalités « copier/coller les

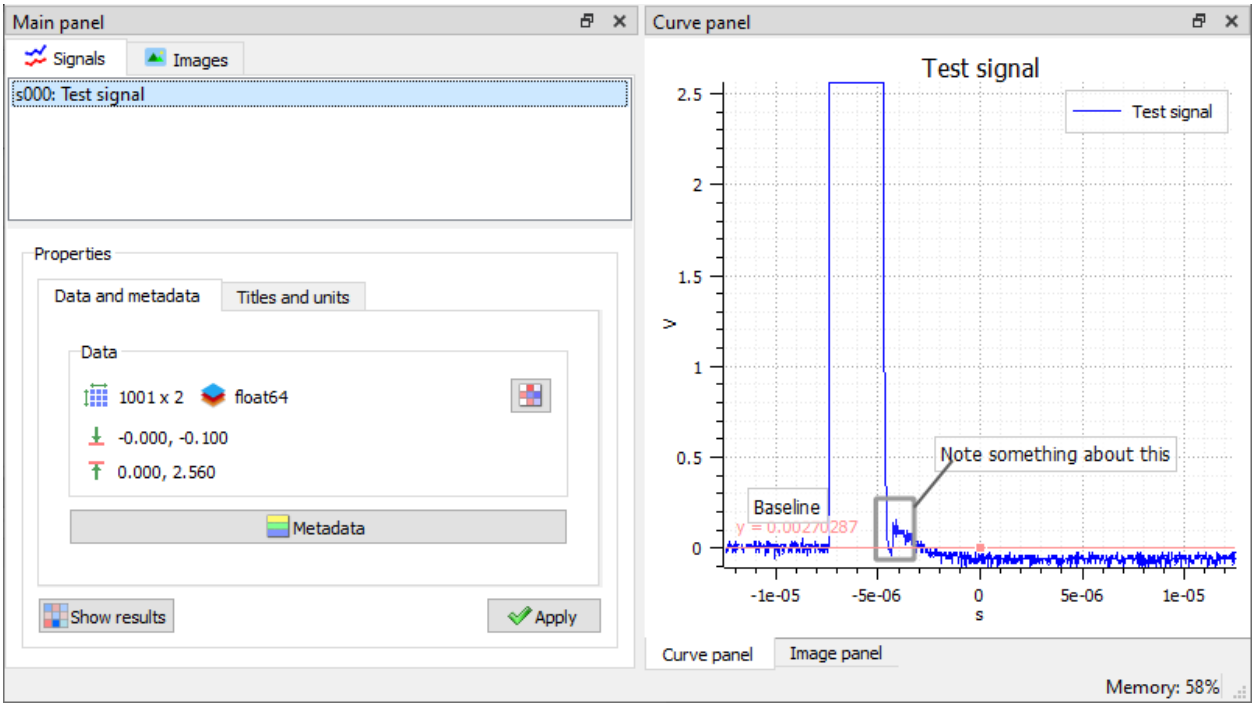


Fig. 36 – Les annotations font désormais partie des métadonnées du signal.

métadonnées ».

Rafraîchissement automatique

Rafraîchit automatiquement la visualisation quand les données changent.

- Si activé (par défaut), la visualisation est automatiquement rafraîchie quand les données changent.
- Si 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.

Afficher seulement le premier signal sélectionné

Basculer entre l'affichage de tous les signaux sélectionnés ou seulement du premier.

Lorsque cette option est activée, seul le premier signal sélectionné est affiché dans la vue du graphique. Cela peut être utile lorsque plusieurs signaux sont sélectionnés et qu'il est (temporairement) préférable de se concentrer sur un seul signal.

Rafraîchir manuellement

Rafraîchir la visualisation manuellement.

Cela déclenche un rafraîchissement de la visualisation. C'est utile quand le rafraîchissement automatique est désactivé, ou quand vous voulez forcer un rafraîchissement de la visualisation.

Afficher les titres des objets graphiques

Affiche/cache les titres des objets graphiques liés aux résultats d'analyse et aux annotations.

Cette option vous permet d'afficher ou de masquer les titres des objets graphiques (par exemple, les titres des résultats d'analyse ou des annotations). Masquer les titres peut être utile lorsque vous voulez visualiser les données sans être perturbé, ou s'il y a trop de titres et qu'ils se chevauchent.

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

Réinitialise le style de courbe au début du cycle.

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.

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.

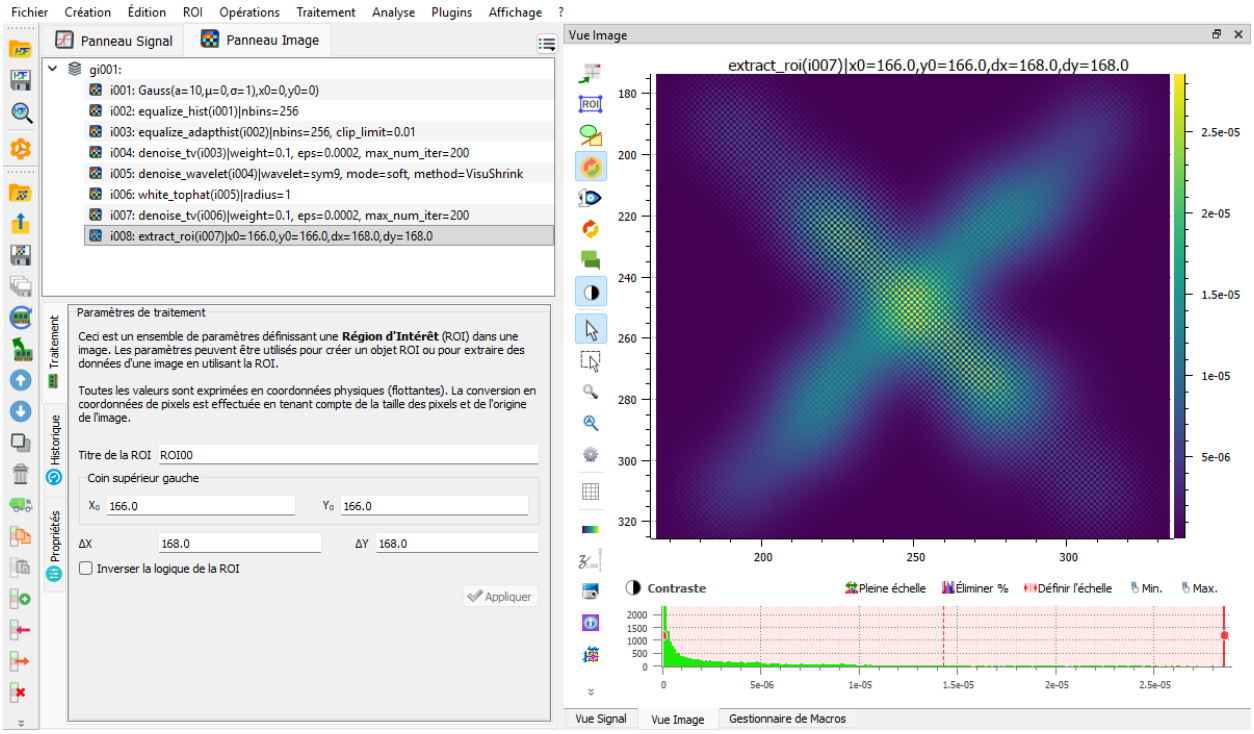


FIG. 37 – Fenêtre principale de DataLab : vue du traitement d'image

2.3.1 Ouvrir et enregistrer des images

Cette section décrit comment ouvrir et enregistrer des images (et des espaces de travail).

Note : Pour créer de nouvelles images à partir de modèles mathématiques, voir *Créer des images*.

Lorsque le « Panneau Image » est sélectionné, les menus et barres d'outils sont mis à jour pour fournir les actions liées aux images.

Le menu « Fichier » vous permet de :

- Ouvrir, enregistrer et fermer des images (voir ci-dessous).
- Enregistrer et restaurer l'espace de travail actuel ou parcourir les fichiers HDF5 (voir *Vue d'ensemble*).
- Modifier les préférences de DataLab (voir *Préférences*).

Fig. 38 – Capture d'écran du menu « Fichier ».

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 MAT	.mat
Fichiers texte	.txt, .csv, .asc
Fichiers Andor SIF	.sif
Fichiers Princeton Instruments SPE	.spe
Fichiers Opticks GEL	.gel
Fichiers Hammamatsu NDPI	.ndpi
Fichiers PCO Camera REC	.rec
Fichiers SPIRICON	.scor-data
Fichiers FXD	.fxd
Images Bitmap	.bmp
Fichiers FT-Lab	.ima

Note : DataLab prend également en charge tout format d'image pouvant être lu par la bibliothèque *imageio*, à condition que le(s) plugin(s) associé(s) soient installé(s) (voir documentation imageio) et que le type de données et la forme du tableau NumPy de sortie soient pris en charge par DataLab.

Pour ajouter un nouveau format de fichier, vous pouvez utiliser l'entrée *imageio_formats* du fichier de configuration de DataLab. Cette entrée est formatée comme l'objet *IMAGEIO_FORMATS* qui représente les formats pris en charge nativement :

```
sigima.config.IMAGEIO_FORMATS = (('*.gel', 'Opticks GEL'), ('*.spe', 'Princeton
Instruments SPE'), ('*.ndpi', 'Hamamatsu Slide Scanner NDPI'), ('*.rec', 'PCO Camera
REC'))
```

Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

Ouvrir depuis un répertoire

Ouvrir plusieurs images depuis un répertoire spécifié.

Enregistrer l'image

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

Enregistrer les images dans un répertoire

Enregistrer toutes les images sélectionnées dans un répertoire spécifié, avec un patron de nom de fichier et un format d'image configurables.

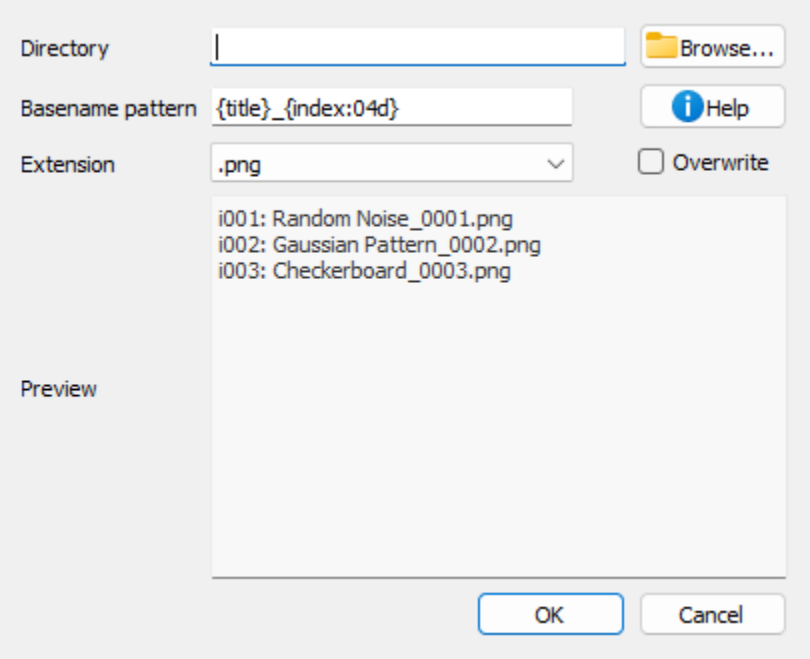


Fig. 39 – Boîte de dialogue Enregistrer les images dans un répertoire.

Lorsque vous sélectionnez « Enregistrer dans un répertoire… » dans le menu Fichier, une boîte de dialogue apparaît où vous pouvez :

- **Répertoire** : Choisissez le répertoire cible où les images seront enregistrées.
- **Nom de fichier** : Définissez un modèle pour les noms de fichiers en utilisant des chaînes de format Python.
- **Extension de fichier** : Sélectionnez le format de sortie (.png, .tiff, .h5ima, etc.).
- **Écraser** : Choisissez si vous souhaitez écraser les fichiers existants.
- **Aperçu** : Voir la liste des fichiers qui seront créés (avec les ID d'objet).

Le modèle de nom de fichier prend en charge les espaces réservés suivants :

- `{title}` : Titre de l'image
- `{index}` : Index de l'image dans la sélection (avec remplissage de zéros)

— `{count}` : Nombre total d'images sélectionnées
— `{xlabel}`, `{xunit}`, `{ylabel}`, `{yunit}`, `{zlabel}`, `{zunit}` : Étiquettes et unités des axes
— `{metadata[key]}` : Accéder aux valeurs des métadonnées

Vous pouvez également utiliser des modificateurs de format, par exemple `{index:03d}` formatera l'index avec un remplissage de zéros sur 3 chiffres (001, 002, 003, etc.).

Importer un fichier texte

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.

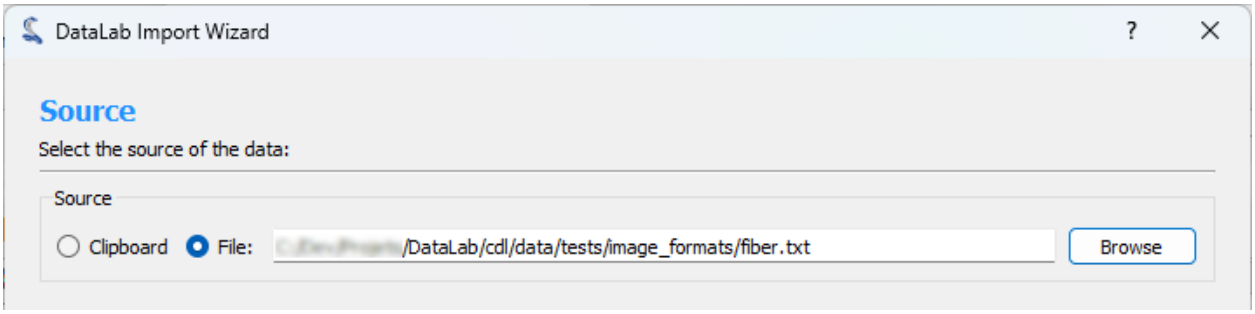


Fig. 40 – Etape 1 : Sélectionner la source

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.

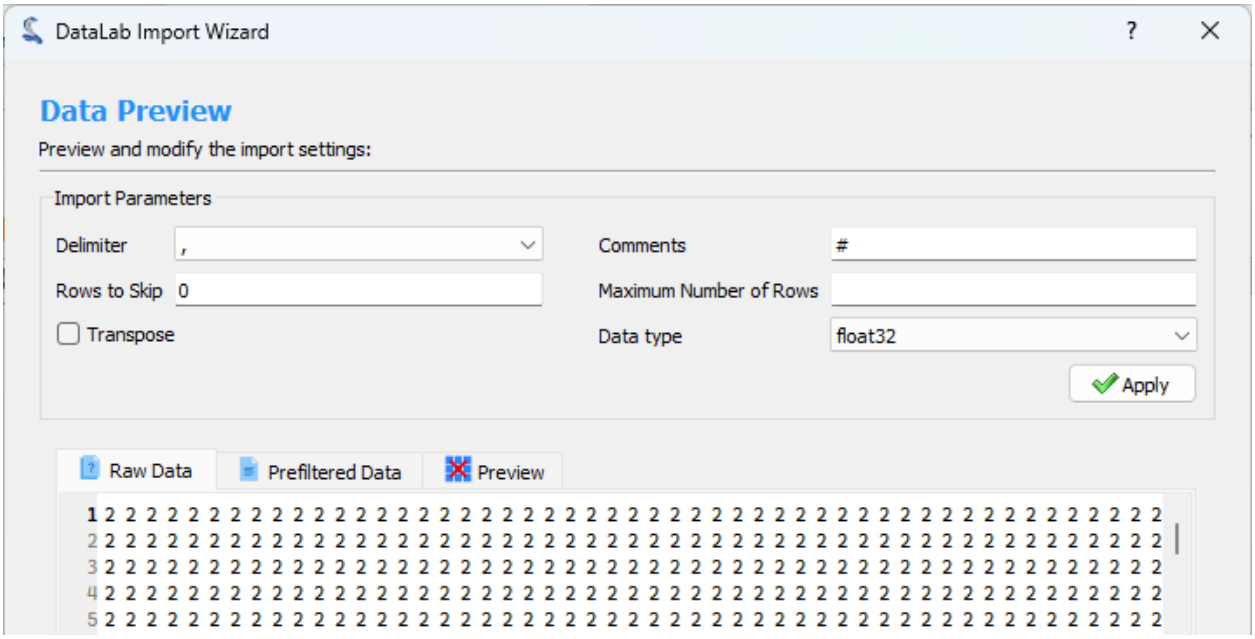


Fig. 41 – Etape 2 : Configurer l'importation

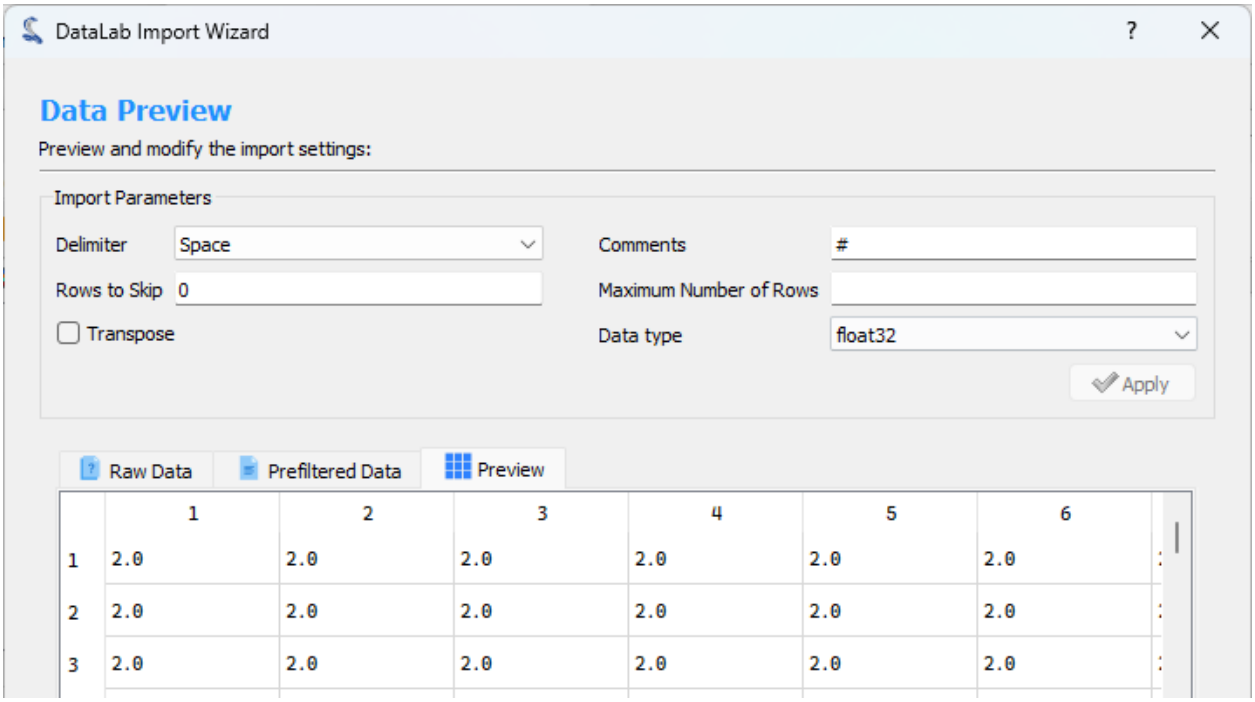


Fig. 42 – Etape 2 : Prévisualiser le résultat

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.

2.3.2 Créer des images

Cette section décrit comment créer de nouvelles images à partir de divers modèles mathématiques.

Lorsque le « Panneau Image » est sélectionné, les menus et barres d'outils sont mis à jour pour fournir les actions relatives aux images.

Le menu « Création » permet de créer de nouvelles images à partir de divers modèles (voir ci-dessous).

Nouvelle image

Créer une nouvelle image à partir de divers modèles (types de données supportés : uint8, uint16, int16, float32, float64) :

Icône	Modèle	Équation
	Zéro	$z[i] = 0$
	Distribution normale	$z[i]$ suit une distribution normale avec moyenne et écart-type configurables
	Distribution de Poisson	$z[i]$ suit une distribution de Poisson avec moyenne configurable
	Distribution uniforme	$z[i]$ suit une distribution uniforme entre deux bornes configurables
	Gaussienne 2D	$z = A \cdot \exp\left(-\frac{\left(\sqrt{(x-x_0)^2+(y-y_0)^2}-\mu\right)^2}{2\sigma^2}\right)$
	Rampe 2D	$z = A(x-x_0) + B(y-y_0) + C$
	Échiquier	Motif de carrés alternés pour l'étalonnage et l'analyse de fréquence spatiale
	Réseau sinusoïdal	$z = A\sin(2\pi(f_x \cdot x + f_y \cdot y) + \varphi) + C$
	Motif d'anneaux	Anneaux circulaires concentriques pour l'analyse radiale
	Étoile de Siemens	Motif de rayons radiaux pour tester la résolution
	Sinc 2D	$z = A \cdot \mathrm{sinc}\left(\frac{\sqrt{(x-x_0)^2+(y-y_0)^2}}{\sigma}\right) + C$

2.3.3 Manipuler les métadonnées et les annotations

Cette section décrit comment manipuler les métadonnées et les annotations dans DataLab.

Les métadonnées d'une image contiennent diverses informations sur l'image ou sa représentation, telles que les paramètres d'affichage, les régions d'intérêt (ROIs), l'historique de la chaîne de traitement, les résultats d'analyse et toute autre information que vous avez pu ajouter aux métadonnées d'une image (ou qui provient du fichier image lui-même).

Voir aussi :

Pour plus d'informations sur les régions d'intérêt (ROIs), voir la section *Régions d'intérêt (ROI)*.

Le menu « Édition » vous permet d'effectuer des opérations d'édition classiques sur l'image ou le groupe d'images actuel (créer/renommer un groupe, déplacer vers le haut/vers le bas, supprimer l'image/le groupe d'images, etc.).

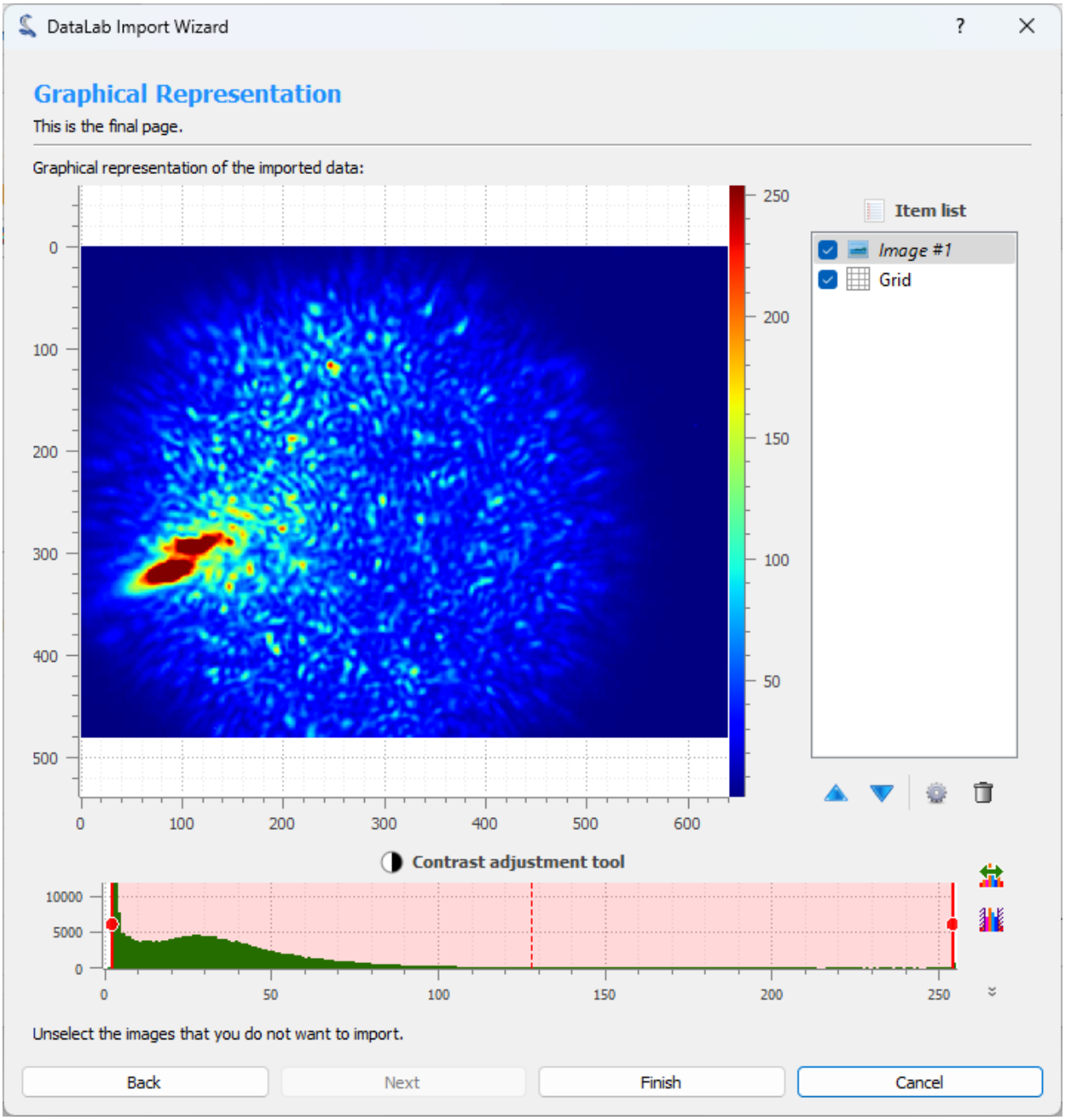


FIG. 43 – Etape 3 : Afficher la représentation graphique

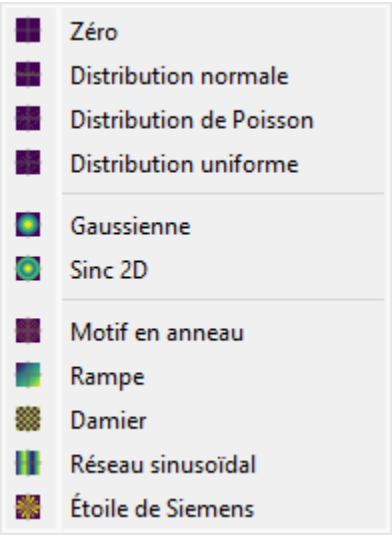


FIG. 44 – Capture d'écran du menu « Création ».

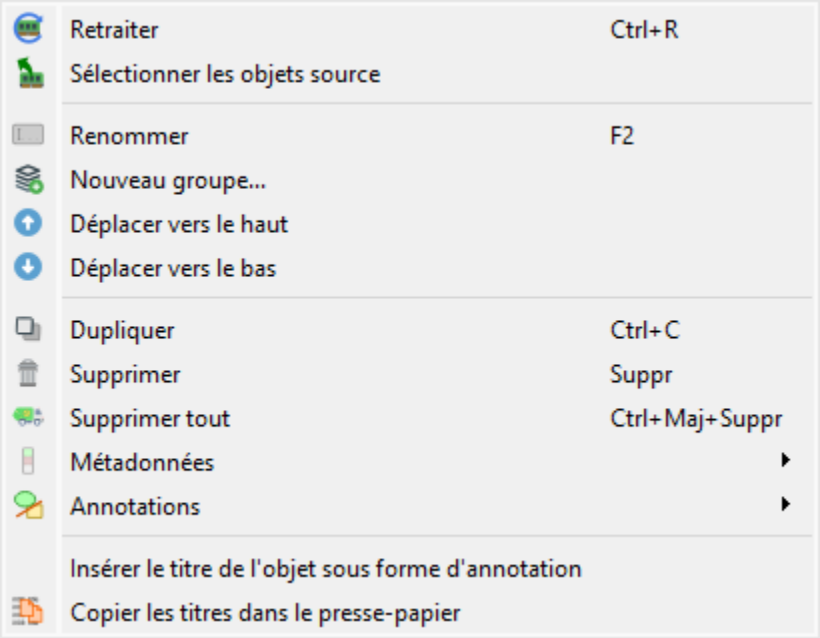


FIG. 45 – Capture d'écran du menu « Édition ».

Comme détaillé ci-dessous, il vous permet également de :

— Naviguer et utiliser l'historique de la chaîne de traitement grâce à des actions telles que « Recalculer » et « Sélectionner les objets sources ».
— Manipuler les métadonnées et les annotations associées à l'image actuelle, grâce aux sous-menus « Métadonnées » et « Annotations » qui offrent les fonctionnalités suivantes.

Recalculer

L'action « Recalculer » vous permet de recalculer l'image (ou les images) sélectionnée(s) en utilisant leurs paramètres de traitement d'origine. Cela est utile lorsque vous souhaitez réexécuter la chaîne de traitement qui a été utilisée pour créer une image, par exemple après avoir modifié les paramètres globaux ou les dépendances.

Note : Cette action n'est disponible que pour les images qui ont été créées par des opérations de traitement et qui ont des paramètres de traitement stockés.

Sélectionner les objets sources

L'action « Sélectionner les objets sources » vous permet de sélectionner l'objet (ou les objets) sources qui ont été utilisés pour créer l'image actuellement sélectionnée. Cela aide à retracer l'historique de traitement et à comprendre quelles images d'origine ont été utilisées comme entrée pour le résultat actuel.

Note : Cette action n'est disponible que lorsqu'une seule image est sélectionnée et que cette image possède des références d'objets sources.

Métadonnées

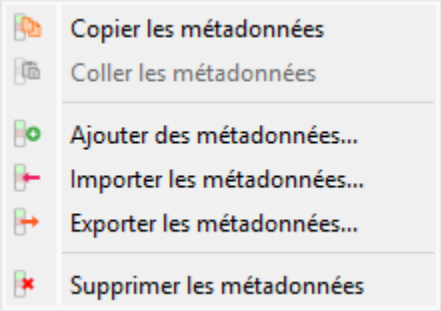


Fig. 46 – Capture d'écran du sous-menu « Métadonnées ».

Copier/coller les métadonnées

Compte tenu du fait que les métadonnées contiennent des informations utiles sur l'image, elles peuvent être copiées et collées d'une image à une autre en sélectionnant les actions « Copier les métadonnées » et « Coller les métadonnées » dans le menu « Édition ».

Cette fonctionnalité vous permet de transférer ces informations d'une image à une autre :

- *Régions d'intérêt (ROIs)* : c'est un moyen très efficace de réutiliser la même ROI sur différentes images et de comparer facilement les résultats de l'analyse sur ces images
- Résultats de calcul, tels qu'une position de barycentre ou une détection de contour (la pertinence du transfert de ces informations dépend du contexte et il appartient à l'utilisateur d'en décider)
- Toute autre information que vous avez pu ajouter aux métadonnées d'une image

Note : Copier les métadonnées d'une image à une autre écrasera les métadonnées de l'image de destination (pour les clés de métadonnées communes aux deux images) ou ajoutera simplement les clés de métadonnées qui ne sont pas présentes dans l'image de destination.

Importer/exporter les métadonnées

Les métadonnées peuvent également être importées et exportées depuis/vers un fichier JSON en utilisant les actions « Importer les métadonnées » et « Exporter les métadonnées » dans le menu « Édition ». C'est exactement la même chose que la fonctionnalité de copier/coller des métadonnées (voir ci-dessus pour plus de détails sur les cas d'utilisation de cette fonctionnalité), mais cela vous permet de sauvegarder les métadonnées dans un fichier et de les importer ultérieurement.

Supprimer les métadonnées

Lors de la suppression des métadonnées en utilisant l'action « Supprimer les métadonnées » dans le menu « Édition », vous serez invité à confirmer la suppression des régions d'intérêt (ROIs) si elles sont présentes dans les métadonnées. Après cette confirmation éventuelle, les métadonnées seront supprimées, ce qui signifie que les résultats d'analyse, les ROIs et toute autre information associée à l'image seront perdus.

Ajouter des métadonnées

L'action « Ajouter des métadonnées » vous permet d'ajouter des éléments de métadonnées personnalisés à une ou plusieurs images sélectionnées. Cela est utile pour étiqueter les images avec des identifiants d'expérience, des noms d'échantillons, des étapes de traitement ou toute autre information personnalisée.

Lorsque vous sélectionnez « Ajouter des métadonnées… » dans le menu Édition, une boîte de dialogue apparaît où vous pouvez :

- **Clé de métadonnées** : Entrez le nom du champ de métadonnées à ajouter
- **Modèle de valeur** : Définir un modèle pour la valeur des métadonnées en utilisant des chaînes de format Python
- **Conversion** : Choisissez comment stocker la valeur (chaîne, flottant, entier ou booléen)
- **Aperçu** : Voir comment les métadonnées seront ajoutées à chaque image sélectionnée

Le modèle de valeur prend en charge les espaces réservés suivants :

- `{title}` : Titre de l'image
- `{index}` : Index basé sur 1 de l'image dans la sélection
- `{count}` : Nombre total d'images sélectionnées
- `{xlabel}`, `{xunit}`, `{ylabel}`, `{yunit}` : Étiquettes et unités des axes
- `{metadata[key]}` : Accéder aux valeurs de métadonnées existantes

Vous pouvez également utiliser des modificateurs de format :

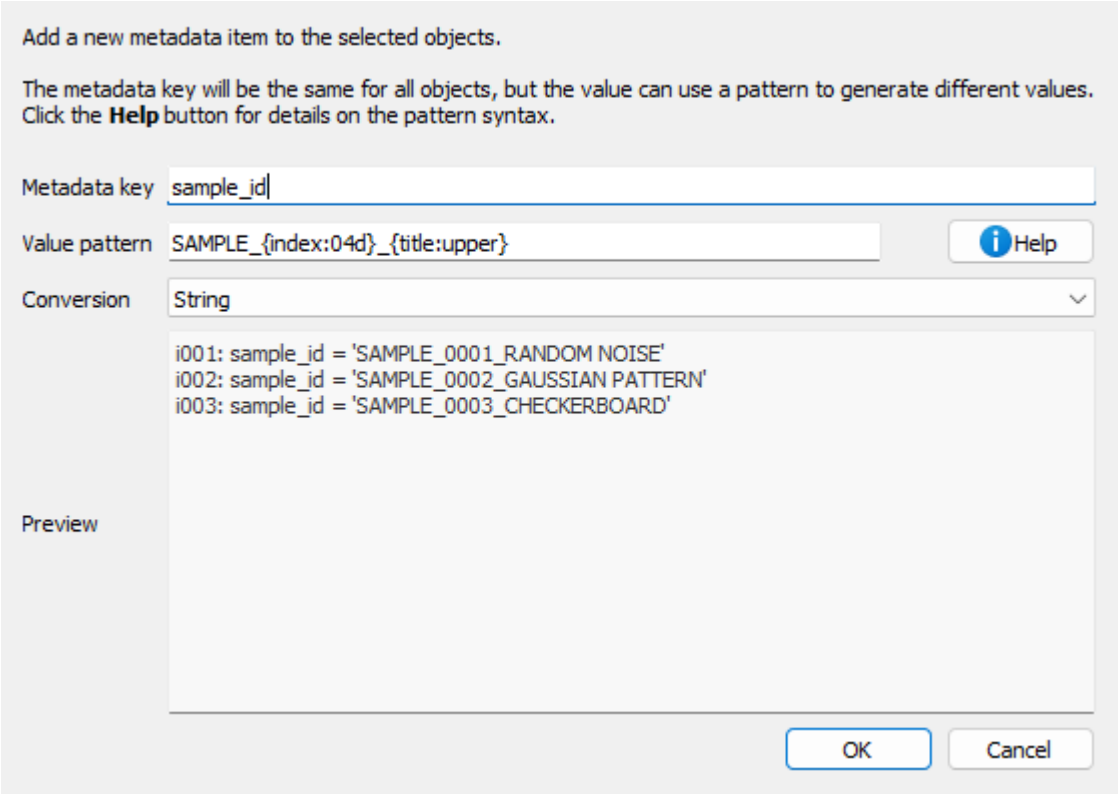


FIG. 47 – Boîte de dialogue Ajouter des métadonnées.

— `{title:upper}` : Convertir en majuscules
— `{title:lower}` : Convertir en minuscules
— `{index:03d}` : Formater en nombre à 3 chiffres avec des zéros devant

Exemples :

— Ajouter le numéro d'acquisition : clé=``acquisition``, modèle=``ACQ_{index :04d}``, conversion=chaîne → Crée des métadonnées comme `acquisition="ACQ_0001"`
— Ajouter le temps d'exposition : clé=``exposure_ms``, modèle=``{metadata[exposure]}``, conversion=float → Copie l'exposition des métadonnées existantes et la convertit en float
— Identifier les images étalonnées : clé=``is_calibrated``, modèle=``true``, conversion=bool → Définit `is_calibrated=True` pour toutes les images sélectionnées.

Annotations

Les annotations sont des éléments visuels qui peuvent être ajoutés aux images pour mettre en évidence des caractéristiques spécifiques, marquer des régions d'intérêt ou ajouter des notes explicatives. DataLab propose un sous-menu dédié dans le menu « Édition » pour gérer les annotations.

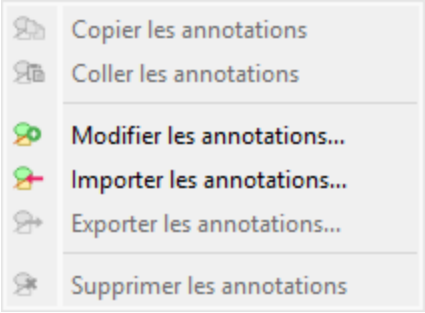


Fig. 48 – Capture d'écran du sous-menu « Annotations ».

Copier/coller les annotations

Les annotations peuvent être copiées d'une image et collées sur une ou plusieurs autres images en utilisant les actions « Copier les annotations » et « Coller les annotations » . Cela est utile lorsque vous souhaitez appliquer les mêmes marqueurs visuels à plusieurs images.

L'action « Coller les annotations » n'est activée que lorsqu'il y a des annotations dans le presse-papiers (c'est-à-dire après avoir utilisé « Copier les annotations »).

Edit annotations

L'action « Éditer les annotations » ouvre une boîte de dialogue où vous pouvez visualiser, ajouter, modifier ou supprimer des annotations de l'image actuelle. Cela offre un moyen visuel de gérer toutes les annotations sur une image.

Importer/exporter les annotations

Les annotations peuvent être enregistrées et chargées à partir de fichiers JSON (extension .dlabann) en utilisant les actions « Importer les annotations » et « Exporter les annotations » . Cela vous permet de :

- Enregistrer des ensembles d'annotations pour une réutilisation ultérieure
- Partager des annotations avec des collègues
- Archiver les annotations séparément des données d'image
- Appliquer les mêmes annotations à différentes images au cours des sessions

L'action « Exporter les annotations » n'est disponible que lorsque l'image sélectionnée a des annotations.

Supprimer les annotations

L'action « Supprimer les annotations » supprime toutes les annotations des images sélectionnées. Cette action n'est activée que lorsque les images sélectionnées ont des annotations.

Note : Les annotations sont stockées séparément des métadonnées et des résultats d'analyse. La suppression des annotations n'affecte pas les ROIs ou d'autres éléments de métadonnées.

Titres des images

Les titres des images peuvent être considérés comme des métadonnées du point de vue de l'utilisateur, même s'ils ne sont pas stockés dans les métadonnées de l'image (mais dans un attribut de l'objet image).

Le menu « Édition » vous permet de :

— « Ajouter le titre de l'objet au graphique » : cette action ajoutera une étiquette en haut de l'image avec son titre, ce qui peut être utile en combinaison avec l'opération « Distribuer sur une grille » (voir *Traitement des images*) pour identifier facilement les images.

— « Copier les titres dans le presse-papiers » : cette action copiera les titres des images sélectionnées dans le presse-papiers, ce qui peut être utile pour les coller dans un éditeur de texte ou dans un tableur.

Exemple du contenu du presse-papiers :

```
g001:
    s001: lorentz(a=1,sigma=1,mu=0,ymin=0)
    s002: derivative(s001)
    s003: wiener(s002)
g002: derivative(g001)
    s004: derivative(s001)
    s005: derivative(s002)
    s006: derivative(s003)
g003: fft(g002)
    s007: fft(s004)
    s008: fft(s005)
    s009: fft(s006)
```

2.3.4 Régions d'intérêt (ROI)

Cette section décrit comment manipuler les régions d'intérêt (ROIs) pour les images dans DataLab.

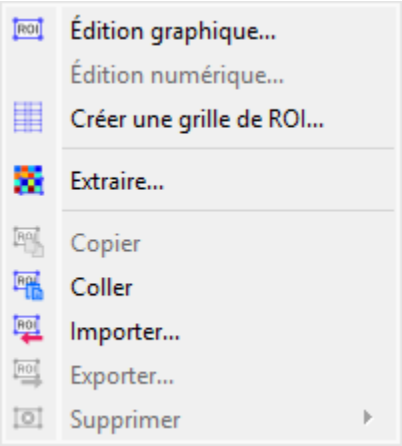


FIG. 49 – Capture d'écran du menu « ROI ».

Le menu « ROI » vous permet de gérer les régions d'intérêt (ROIs) associées à l'image actuelle.

Voir aussi :

Pour plus d'informations sur les métadonnées d'image, voir la section *Manipuler les métadonnées et les annotations*.

Les régions d'intérêt (ROI) sont des zones d'image définies par l'utilisateur pour effectuer des opérations spécifiques, des traitements ou des analyses sur elles.

Les ROI sont prises en compte dans presque toutes les fonctionnalités de calcul de DataLab :

— Les fonctionnalités du menu « Opérations » sont appliquées uniquement sur la ROI si elle est définie (sauf si l'opération modifie la forme des données - comme l'opération de redimensionnement - ou la taille des pixels - comme l'opération de binning).
— Les actions du menu « Traitement » sont effectuées uniquement sur la ROI si elle est définie (sauf si le type de données du signal de destination est différent de celui de la source, comme dans les fonctionnalités d'analyse de Fourier ou comme dans les opérations de seuillage).
— Les actions du menu « Analyse » sont effectuées uniquement sur la ROI si elle est définie.

Note : Les ROI sont stockées en tant que métadonnées et sont donc attachées à l'image.

Le menu « ROI » vous permet de :

— « Édition graphique » : ouvrir une boîte de dialogue pour gérer les ROI associées à l'image sélectionnée (ajouter, supprimer, déplacer, redimensionner, etc.). La boîte de dialogue de définition des ROI est exactement la même que celle de l'extraction des ROI (voir ci-dessous).

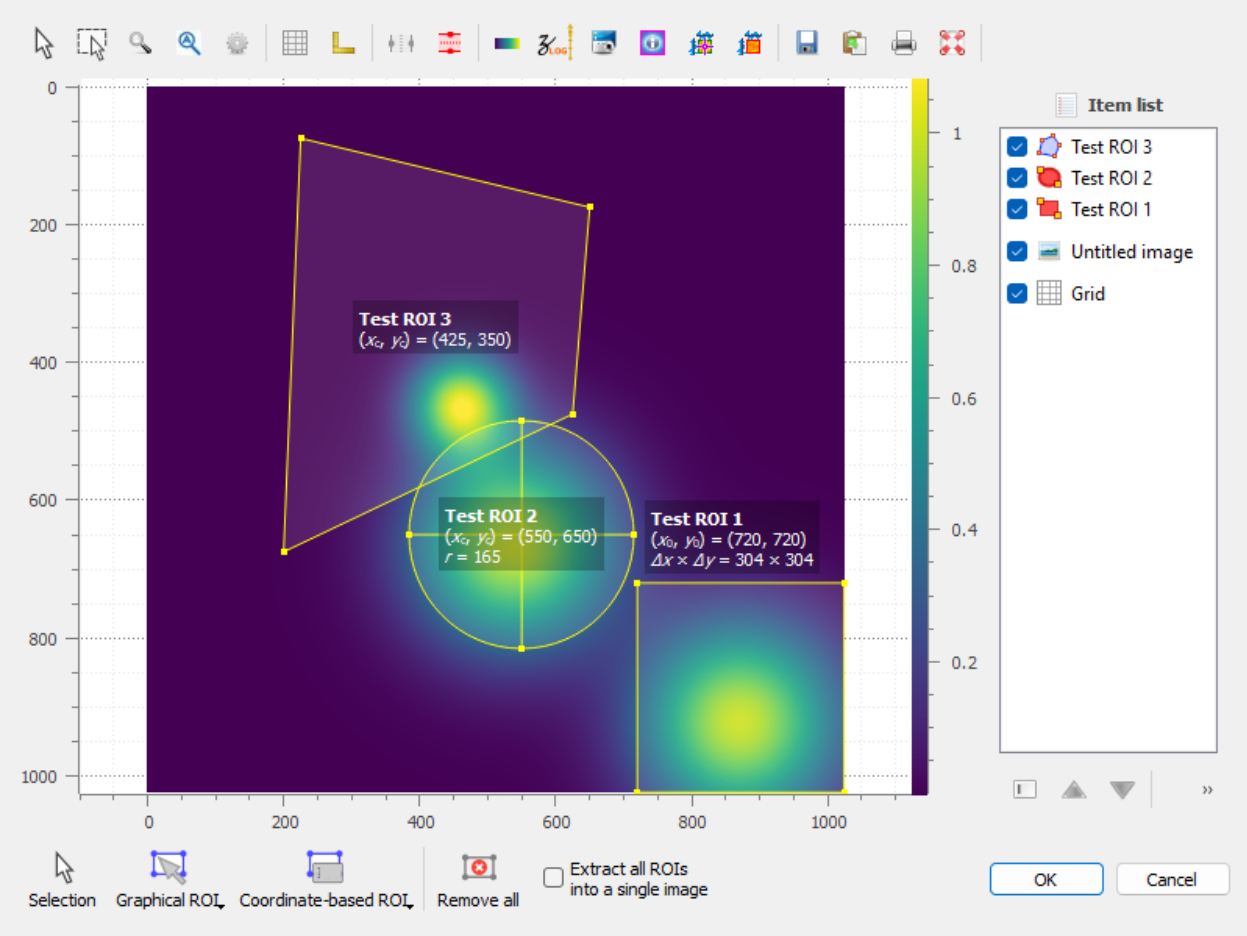


Fig. 50 – Une image avec une ROI.

— « Édition numérique » : ouvrir une boîte de dialogue pour modifier les paramètres des ROI sélectionnées numériquement (c'est-à-dire à l'aide d'un formulaire simple). Cela vous permet de définir ou de modifier des ROI en fonction de valeurs numériques.
— « Créer une grille de ROI » : ouvrir une boîte de dialogue pour créer une grille de ROIs sur l'image sélectionnée. Cela vous permet de définir plusieurs ROIs réparties uniformément sur l'image.

— « Extraire » : extraire la ROI définie des images sélectionnées. Cela créera une nouvelle image pour chaque ROI (ou une seule image, si l'option « Extraire toutes les ROIs dans une seule image » est sélectionnée dans la boîte de dialogue), avec les mêmes métadonnées que l'image d'origine, mais avec les données correspondant uniquement à la ROI. Les nouvelles images seront ajoutées à l'espace de travail actuel.
— « Copier » : copier la ROI définie des images sélectionnées dans le presse-papiers. Cela vous permet de coller la ROI dans une autre image ou de l'utiliser dans d'autres opérations.
— « Coller » : coller la ROI copiée du presse-papiers dans les images sélectionnées. Cela ajoutera la ROI aux images, vous permettant de définir ou de modifier des ROI en fonction de celles précédemment copiées.
— « Importer » : importer des ROI à partir d'un fichier. Cela vous permet de charger des ROI précédemment enregistrées dans l'image actuelle.
— « Exporter » : exporter les ROI définies vers un fichier. Cela vous permet de sauvegarder les ROI pour une utilisation ultérieure ou de les partager avec d'autres.
— « Supprimer tout » : supprimer toutes les ROI définies pour les images sélectionnées.

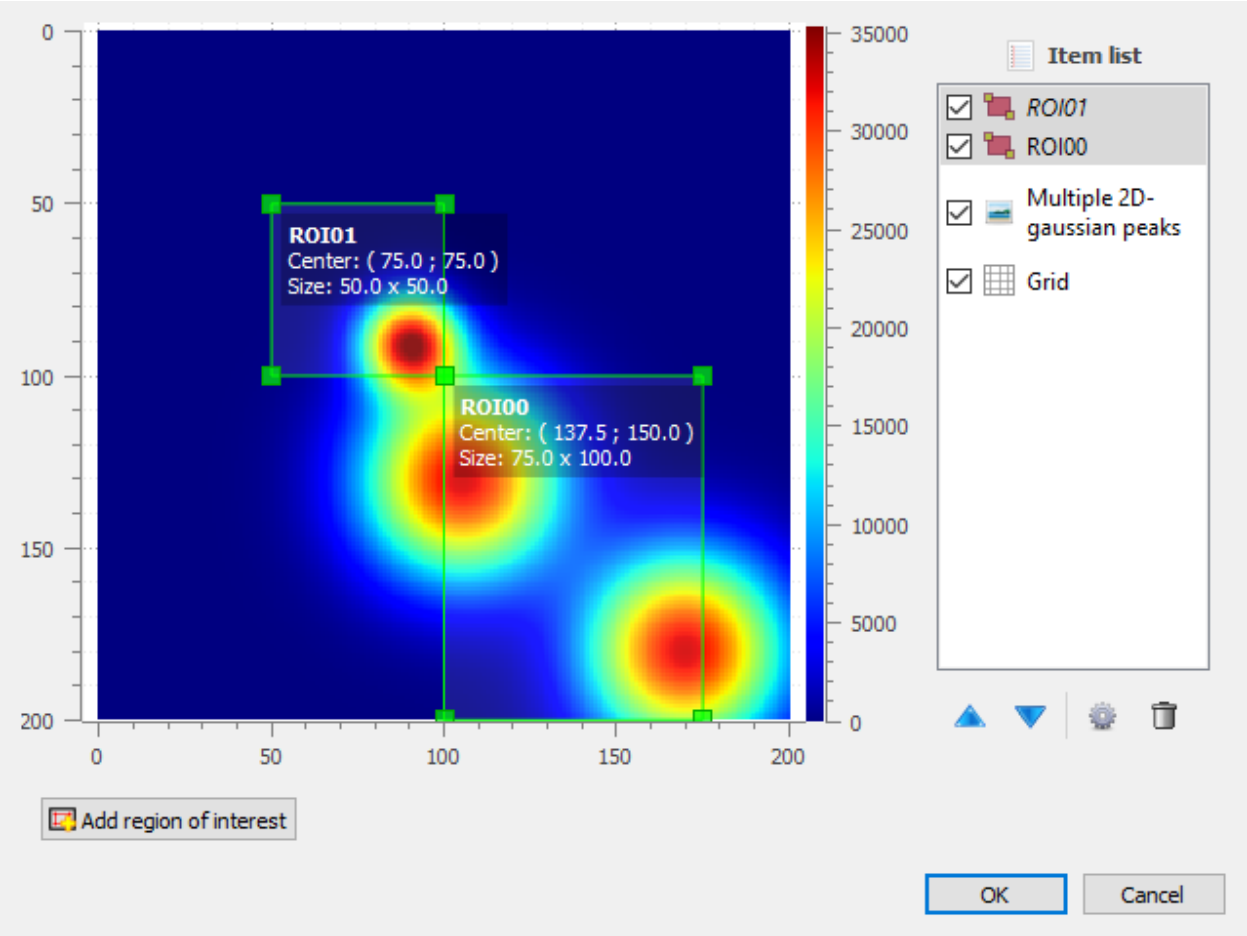


Fig. 51 – Boîte de dialogue d'extraction des ROI : la ROI est définie en déplaçant la position et en ajustant la taille d'une forme rectangulaire.

2.3.5 Opérations sur les images

Cette section décrit les opérations qui peuvent être effectuées sur les images.

Voir aussi :

Traitement des images pour plus d'informations sur les fonctionnalités de traitement d'image, ou *Analyse sur les images* pour des informations sur les fonctionnalités d'analyse des images.

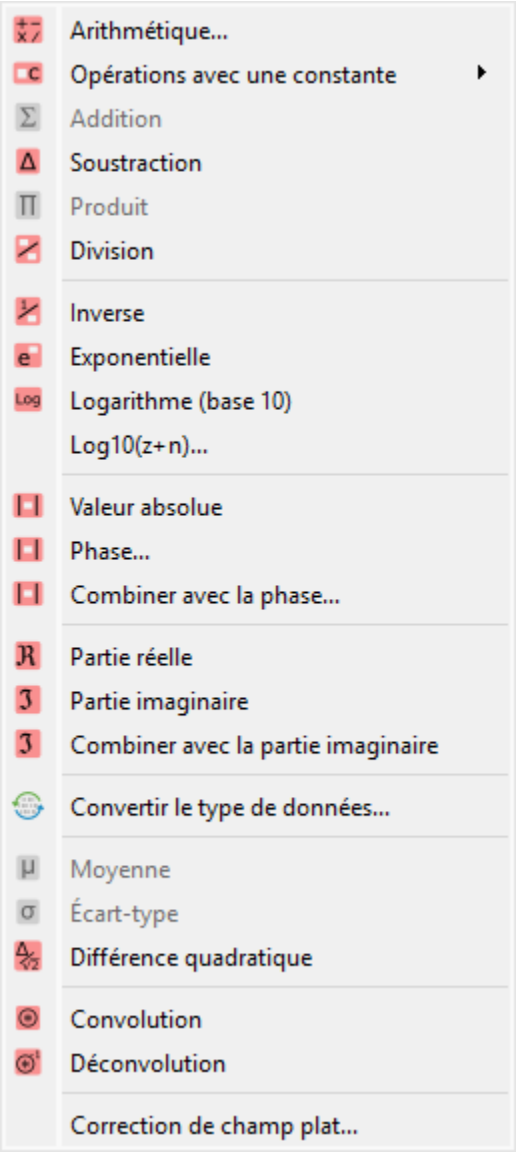


Fig. 52 – Capture d'écran du menu « Opérations ».

Lorsque le « Panneau Image » est sélectionné, les menus et barres d'outils sont mis à jour pour fournir les actions liées aux images.

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.

Opérations avec une constante

Crée une image à partir d'une opération avec une constante sur chaque image sélectionnée :

Opération	Equation
Addition	$z_k = z_{k-1} + conv(c)$
Soustraction	$z_k = z_{k-1} - conv(c)$
Multiplication	$z_k = conv(z_{k-1} \times c)$
Division	$z_k = conv(\frac{z_{k-1}}{c})$

où c est la valeur constante et $conv$ est la fonction de conversion qui gère la conversion du type de données (en conservant le même type de données que l'image d'entrée).

Opérations arithmétiques de base

Les opérations arithmétiques sont effectuées pixel par pixel entre les images sélectionnées.

Opération	Description
Somme	$z_M = \sum_{k=0}^{M-1} z_k$
Soustraction	$z_2 = z_1 - z_0$
Produit	$z_M = \prod_{k=0}^{M-1} z_k$
Division	$z_2 = \frac{z_1}{z_0}$
Inverse	$z_2 = \frac{1}{z_1}$
Exponentielle	$z_2 = \exp(z_1)$
Logarithme (base 10)	$z_2 = \log_{10}(z_1)$

Fonctions mathématiques de base

Fonction	Description
Exponentielle	$z_k = \exp(z_k)$
Logarithme (base 10)	$z_k = \log_{10}(z_k)$
Log10(z+10)	$z_k = \log_{10}(z_k + n)$ (utile si le fond de l'image est nul)

Convolution et Déconvolution

Opération	Implémentation
Convolution	Basée sur scipy.signal.convolve
Déconvolution	Déconvolution dans le domaine fréquentiel

Valeur absolue et opérations sur les images complexes

Opération	Description
Valeur absolue	$z_k = \lvert z_{k-1} \rvert$
Phase (argument)	`sigima.proc.image.phase()`
Combiner avec la phase	Considère l'image courante comme le module et permet de sélectionner une image représentant la phase pour les fusionner en une image complexe `sigima.proc.image.complex_from_magnitude_phase()`
Partie réelle	$z_k = \Re(z_{k-1})$
Partie imaginaire	$z_k = \Im(z_{k-1})$
Combiner avec la partie imaginaire	Considère l'image courante comme la partie réelle et permet de sélectionner une image représentant la partie imaginaire pour les fusionner en une image complexe `sigima.proc.image.complex_from_real_imag()`

Conversion du type de données

L'action « Convertir le type de données » permet de convertir le type de données des images sélectionnées. Pour les types de données entiers, la conversion est effectuée en rognant les valeurs à la plage de nouveaux types de données avant de convertir effectivement le type de données. Pour les types de données à virgule flottante, la conversion est directe.

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

Statistiques entre les images

Crée une nouvelle image à partir d'une opération statistique sur chaque pixel des images sélectionnées :

Opération	Description
Moyenne	$z_M = \frac{1}{M} \sum_{k=0}^{M-1} z_k$
Écart-type	$z_M = \sqrt{\frac{1}{M} \sum_{k=0}^{M-1} (y_k - \bar{y})^2}$
Soustraction quadratique	$z_2 = \frac{z_1 - z_0}{\sqrt{2}}$

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. »*n*

2.3.6 Traitement des images

Cette section décrit les fonctionnalités de traitement d'image disponibles dans DataLab.

Voir aussi :

Opérations sur les images pour plus d'informations sur les opérations qui peuvent être effectuées sur les images, ou *Analyse sur les images* pour des informations sur les fonctionnalités d'analyse des images.

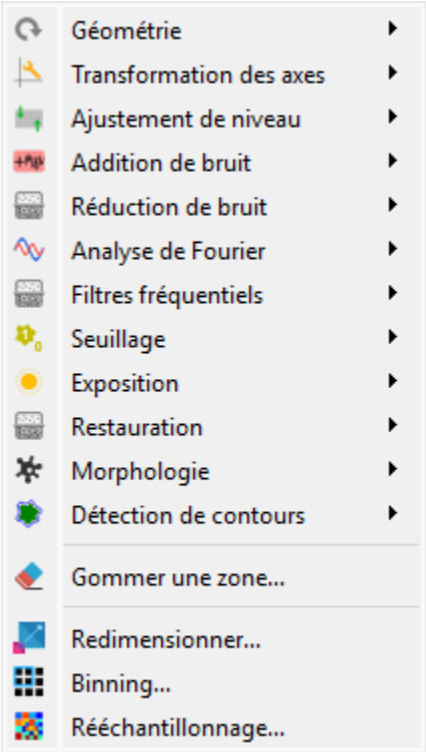


FIG. 53 – Capture d'écran du menu « Traitement ».

Lorsque le « Panneau Image » est sélectionné, les menus et barres d'outils sont mis à jour pour fournir les actions liées aux images.

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.

Géométrie

Symétrie et rotation

Crée une nouvelle image en effectuant une symétrie axiale ou une rotation des données de l'image sélectionnée. L'image peut être transformée par symétrie axiale horizontale, verticale ou diagonale (transposition).

Opération	Description
Symétrie horizontale	Effet miroir le long de l'axe vertical
Symétrie diagonale	Transposer l'image (permuter les axes X/Y)
Symétrie verticale	Effet miroir le long de l'axe horizontal
Rotation de 90° à droite	Rotation de l'image de 90° dans le sens horaire
Rotation de 90° à gauche	Rotation de l'image de 90° dans le sens antihoraire
Rotation de…	Rotation de l'image d'un angle quelconque (défini par l'utilisateur)

Distribuer les images sur une grille

Fonctionnalité	Description
Distribuer sur une grille	Distribuer les images sélectionnées sur une grille régulière
Réinitialiser les positions	Réinitialiser les positions des images sélectionnées aux coordonnées de la première image (x0, y0)

Transformation des axes

Définir des coordonnées uniformes

Crée une nouvelle image avec des coordonnées uniformes (c'est-à-dire avec une taille de pixel constante dans les directions X et Y).

Étalonnage polynomial

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

Note : L'étalonnage linéaire a été supprimé de l'interface utilisateur car il s'agit d'un cas particulier d'étalonnage polynomial de degré 1.

Ajustement des niveaux

Normalisation

Crée une image à partir de la normalisation de chaque image sélectionnée par maximum, amplitude, somme, énergie ou RMS :

Normalisation	Equation
Maximum	$z_1 = \frac{z_0}{z_{\max}}$
Amplitude	$z_1 = \frac{z_0}{z_{\max} - z_{\min}}$
Aire	$z_1 = \frac{z_0}{\sum_{i=0}^{N-1} z_i}$
Energie	$z_1 = \frac{z_0}{\sqrt{\sum_{n=0}^{N} \lvert z_0[n] \rvert^2}}$
RMS	$z_1 = \frac{z_0}{\sqrt{\frac{1}{N} \sum_{n=0}^{N} \lvert z_0[n] \rvert^2}}$

Ecrêtage

Applique un écrêtage sur chaque image sélectionnée.

Soustraction d'offset

Crée une image à partir du résultat d'une correction d'offset sur chaque image sélectionnée. Cette opération est réalisée en soustrayant la valeur de fond de l'image, qui est estimée par la valeur moyenne d'une zone rectangulaire définie par l'utilisateur.

Ajout de bruit

Génère de nouvelles images en ajoutant le même bruit à chaque image sélectionnée. Les types de bruit disponibles sont :

Bruit	Description
Gaussien	Distribution normale
Uniforme	Distribution uniforme
Poisson	Distribution de Poisson

Réduction de bruit

Crée une image à partir du résultat d'un débruitage sur chaque image sélectionnée.

Les filtres suivants sont disponibles :

Filtre	Formule/implémentation
Filtre gaussien	scipy.ndimage.gaussian_filter
Moyenne mobile	scipy.ndimage.uniform_filter
Médiane mobile	scipy.ndimage.median_filter
Filtre de Wiener	scipy.signal.wiener

Analyse de Fourier

Complément de zéros

Crée une image à partir du résultat d'un complément de zéros sur chaque image sélectionnée.

Les paramètres suivants sont disponibles :

Paramètre	Description
Stratégie	Stratégie de complément de zéros (voir ci-dessous)
Lignes	Nombre de lignes à ajouter (si *strategy* est "custom")
Colonnes	Nombre de colonnes à ajouter (si *strategy* est "custom")
Position	Position des zéros ajoutés : "bottom-right", "centered"

Les stratégies de complément de zéros font référence à la méthode utilisée pour ajouter des zéros à l'image, et peuvent être l'une des suivantes :

Stratégie	Description
next_pow2	Puissance de 2 supérieure (ex. : 512, 1024, …)
multiple_of_64	Multiple de 64 supérieur (ex. : 512, 576, …)
custom	Taille personnalisée (définie par l'utilisateur)

Fonctions liées à la FFT

Crée une image à partir du résultat d'une analyse de Fourier sur chaque image sélectionnée.

Les fonctions suivantes sont disponibles :

Fonction	Description	Formule/implémentation
FFT	Transformée de Fourier rapide	numpy.fft.fft2
FFT inverse	Transformée de Fourier rapide inverse	numpy.fft.ifft2
Spectre d'amplitude	Optionnel : sortie en décibels (dB)	$z_1 = \lvert\mathrm{FFT}\left(z_0\right)\rvert$ ou $z_1 = 20\log_{10}\left(\lvert\mathrm{FFT}\left(z_0\right)\rvert\right)$ (dB)
Spectre de phase	Phase de la FFT exprimée en degrés, en utilisant numpy.angle	$z_1 = \angle\left(\mathrm{FFT}\left(z_0\right)\right)$
Densité spectrale de puissance	Optionnel : sortie en décibels (dB)	$z_1 = \lvert\mathrm{FFT}\left(z_0\right)\rvert^2$ ou $z_1 = 10\log_{10}\left(\lvert\mathrm{FFT}\left(z_0\right)\rvert^2\right)$ (dB)

Note : La FFT et la FFT inverse sont effectuées avec décalage de fréquence si l'option est activée dans les paramètres de DataLab (voir *Préférences*).

Filtre fréquentiels

Les filtres fréquentiels suivants sont disponibles :

Méthode	Description
Butterworth	Filtre de Butterworth, basé sur skimage.filters.butterworth
Filtre gaussien	Filtre gaussien

Seuillage

Crée une image à partir du résultat d'un seuillage sur chaque image, éventuellement basé sur des paramètres définis par l'utilisateur (« Seuillage paramétrique »).

Les paramètres suivants sont disponibles lors de la sélection de « Seuillage paramétrique » :

Paramètre	Description
Méthode de seuillage	La méthode de seuillage à utiliser (voir le tableau ci-dessous)
Classes	Nombre de classes pour le calcul de l'histogramme
Valeur	Valeur de seuil
Opération	Opération à appliquer (> ou <)

Les méthodes de seuillage suivantes sont disponibles :

Méthode	Implémentation
Manuel	Seuillage manuel (paramètres définis par l'utilisateur)
ISODATA	skimage.filters.threshold_isodata
Li	skimage.filters.threshold_li
Moyenne	skimage.filters.threshold_mean
Minimum	skimage.filters.threshold_minimum
Otsu	skimage.filters.threshold_otsu
Triangle	skimage.filters.threshold_triangle
Yen	skimage.filters.threshold_yen

Note : L'option « Toutes les méthodes de seuillage » permet d'appliquer toutes les méthodes de seuillage à la même image. Combinée avec l'option « distribuer sur une grille », cela permet de comparer les différentes méthodes de seuillage sur la même image.

Exposition

Crée une image à partir du résultat d'une correction d'exposition sur chaque image sélectionnée.

Les fonctions suivantes sont disponibles :

Fonction	Implémentation	Commentaires
Correction gamma	skimage.exposure.adjust_gamr	
Correction logarithmique	skimage.exposure.adjust_log	
Correction sigmoïde	skimage.exposure.adjust_sigm	
Egalisation d'histogramme	skimage.exposure.equalize_his	
Egalisation d'histogramme adaptative	skimage.exposure.equalize_ad	Algorithme CLAHE (Contrast Limited Adaptive Histogram Equalization)
Ajustement des niveaux	skimage.exposure.rescale_inte	Réduit ou étend la plage de répartition des niveaux de l'image

Restauration

Crée une image à partir du résultat d'une restauration sur chaque image sélectionnée.

Les fonctions suivantes sont disponibles :

Fonction	Implémentation	Commentaires
Débruitage par variation totale	skimage.restoration.denoise_tv	
Débruitage par filtre bilatéral	skimage.restoration.denoise_bi	
Débruitage par ondelettes	skimage.restoration.denoise_w	
Débruitage par Top-Hat	skimage.morphology.white_to	Débruite l'image en soustrayant sa transformation Top-Hat

Note : L'option « Toutes les méthodes de débruitage » permet d'appliquer toutes les méthodes de débruitage à la même image. Combinée avec l'option « distribuer sur une grille », cela permet de comparer les différentes méthodes de débruitage sur la même image.

Morphologie

Crée une image à partir du résultat d'opérations morphologiques sur chaque image sélectionnée, en utilisant un disque comme empreinte.

Les fonctions suivantes sont disponibles :

Fonction	Implémentation
Top-Hat (disque)	skimage.morphology.white_tophat
Top-Hat dual (disque)	skimage.morphology.black_tophat
Erosion (disque)	skimage.morphology.erosion
Dilatation (disque)	skimage.morphology.dilation
Ouverture (disque)	skimage.morphology.opening
Fermeture (disque)	skimage.morphology.closing

Note : L'option « Toutes les opérations morphologiques » permet d'appliquer toutes les opérations morphologiques à la même image. Combinée avec l'option « distribuer sur une grille », cela permet de comparer les différentes opérations morphologiques sur la même image.

Contours

Crée une image à partir du résultat d'un filtrage de contours sur chaque image sélectionnée.

Les fonctions suivantes sont disponibles :

Fonction	Implémentation
Filtre de Canny	skimage.feature.canny
Filtre de Farid	skimage.filters.farid
Filtre de Farid (horizontal)	skimage.filters.farid_h
Filtre de Farid (vertical)	skimage.filters.farid_v
Filtre de Laplace	skimage.filters.laplace
Filtre de Prewitt	skimage.filters.prewitt
Filtre de Prewitt (horizontal)	skimage.filters.prewitt_h
Filtre de Prewitt (vertical)	skimage.filters.prewitt_v
Filtre de Roberts	skimage.filters.roberts
Filtre de Scharr	skimage.filters.scharr
Filtre de Scharr (horizontal)	skimage.filters.scharr_h
Filtre de Scharr (vertical)	skimage.filters.scharr_v
Filtre de Sobel	skimage.filters.sobel
Filtre de Sobel (horizontal)	skimage.filters.sobel_h
Filtre de Sobel (vertical)	skimage.filters.sobel_v

Note : L'option « Tous les filtres de contours » permet d'appliquer tous les algorithmes de filtrage de contours à la même image. Combinée avec l'option « distribuer sur une grille », cela permet de comparer les différents filtres de contours sur la même image.

Gommer une zone

Gomme une zone de l'image définie par une région d'intérêt (ROI).

Note : La région à effacer est définie par l'utilisateur via une boîte de dialogue. Elle peut consister en une seule ROI ou plusieurs ROIs avec différentes formes (rectangulaire, circulaire, etc.). Notez que la ROI définie ici n'est liée à aucun objet ; elle est utilisée uniquement pour spécifier la zone à effacer dans l'image. En particulier, elle est indépendante de la ROI de l'image, le cas échéant (cette dernière est affichée dans la boîte de dialogue comme une zone masquée).

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).

Rééchantillonnage

Génère de nouvelles images en rééchantillonnant chaque image sélectionnée. Les paramètres suivants sont disponibles :

Paramètre	Description
math :*x_{min}*	Coordonnée x minimale de l'image de sortie
x_{max}	Coordonnée x maximale de l'image de sortie
y_{min}	Coordonnée y minimale de l'image de sortie
y_{max}	Coordonnée y maximale de l'image de sortie
Mode	Mode de définition de la taille de l'image : "Taille des pixels" ou "Taille de l'image de sortie". Le mode "Taille des pixels" permet de définir la taille des pixels de la nouvelle image, tandis que le mode "Taille de l'image de sortie" permet de définir le nombre de pixels de la nouvelle image.
X	Taille des pixels selon la direction x (si le mode "Taille des pixels" est sélectionné)
Y	Taille des pixels selon la direction y (si le mode "Taille des pixels" est sélectionné)
Largeur	Largeur de l'image de sortie en pixels (si le mode "Taille de l'image de sortie" est sélectionné)
Hauteur	Hauteur de l'image de sortie en pixels (si le mode "Taille de l'image de sortie" est sélectionné)
Méthode d'interpolation	Méthode d'interpolation à utiliser : "nearest", "linear", "cubic"
Valeur de remplissage	Valeur à utiliser pour les points en dehors du domaine de l'image d'entrée (si None, la fonction utilise NaN pour l'extrapolation)

2.3.7 Analyse sur les images

Cette section décrit les fonctionnalités d'analyse d'images disponibles dans DataLab.

Voir aussi :

Opérations sur les images pour plus d'informations sur les opérations pouvant être effectuées sur les images, ou *Traitement des images* pour des informations sur les fonctionnalités de traitement des images.

Lorsque le « Panneau Image » est sélectionné, les menus et barres d'outils sont mis à jour pour fournir les actions liées aux images.

Le menu Analyse » 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, une « analyse » est une fonctionnalité de calcul d'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.

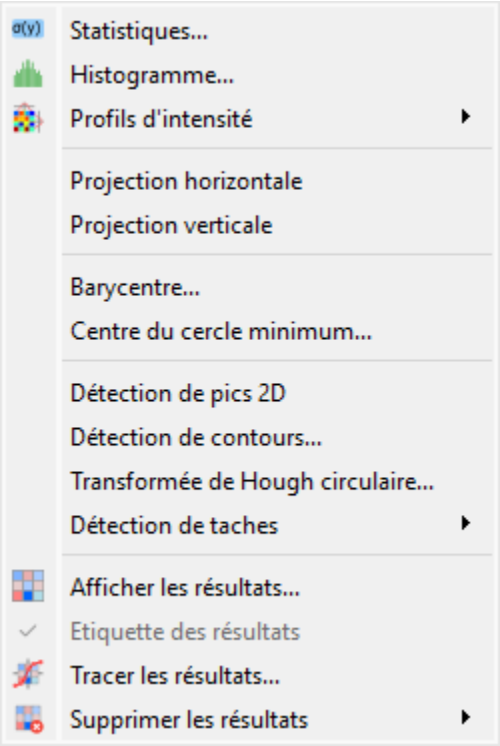


FIG. 54 – Capture d'écran du menu Analyse ».

Statistiques

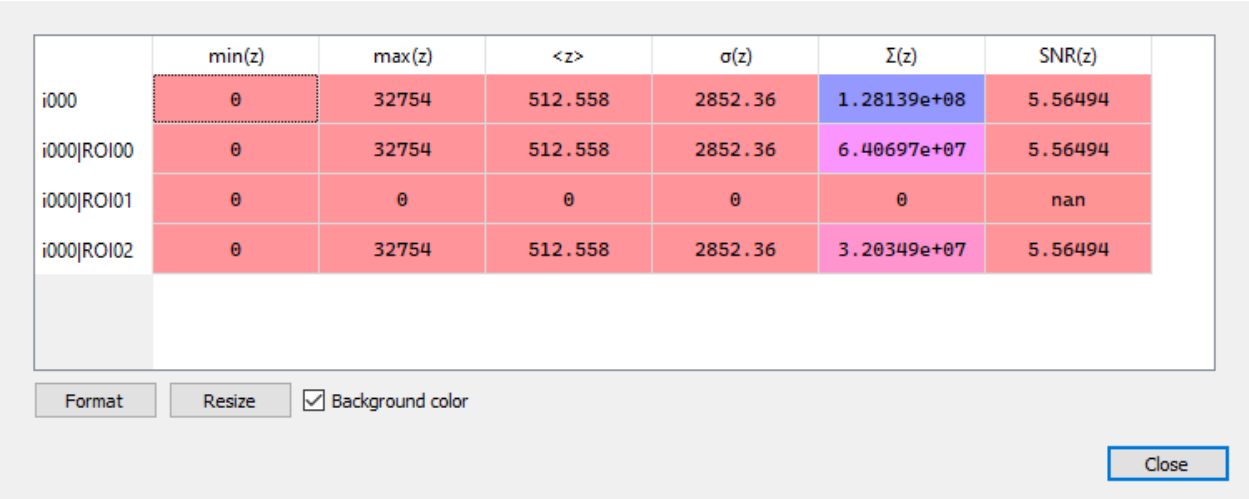
Calcule des statistiques sur les images sélectionnées et affiche un tableau récapitulatif.

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



	min(z)	max(z)	<z>	σ(z)	Σ(z)	SNR(z)
i000	0	32754	512.558	2852.36	1.28139e+08	5.56494
i000|ROI00	0	32754	512.558	2852.36	6.40697e+07	5.56494
i000|ROI01	0	0	0	0	0	nan
i000|ROI02	0	32754	512.558	2852.36	3.20349e+07	5.56494

FIG. 55 – 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).

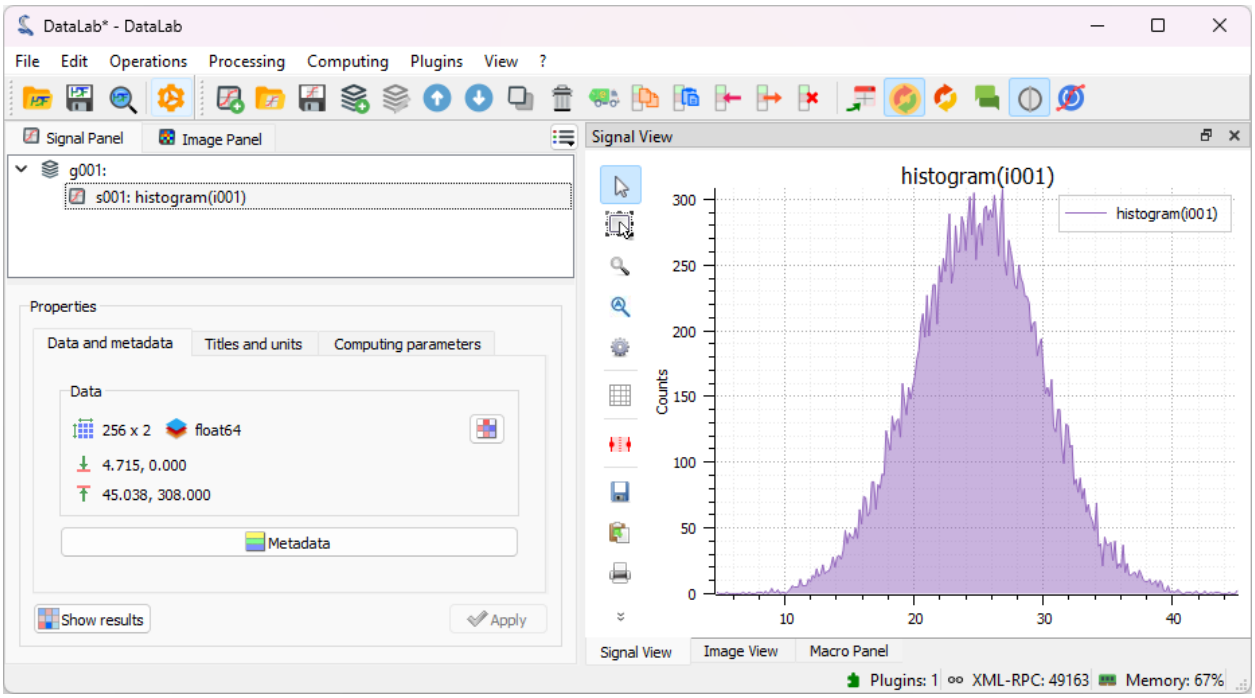


FIG. 56 – Exemple d'histogramme.

Profils d'intensité

Profil rectiligne

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

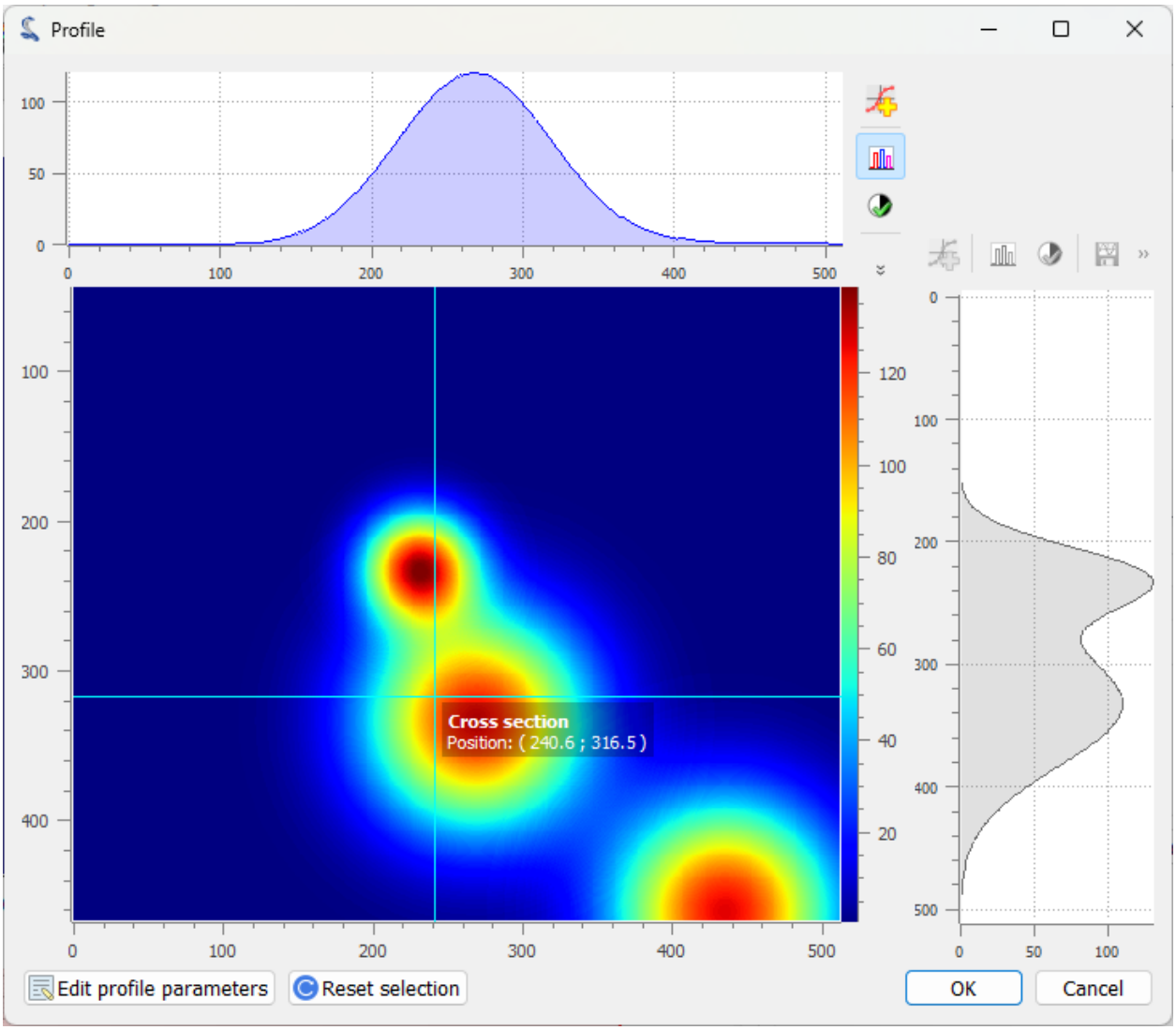


FIG. 57 – Boîte de dialogue d'extraction de profil. Les paramètres peuvent être également définis manuellement (bouton « Editer les paramètres du profil »).

Profil le long d'un segment

Extraire un profil le long d'un segment de chaque image sélectionnée et créer un nouveau signal à partir de chacun 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 chacun de ces profils.

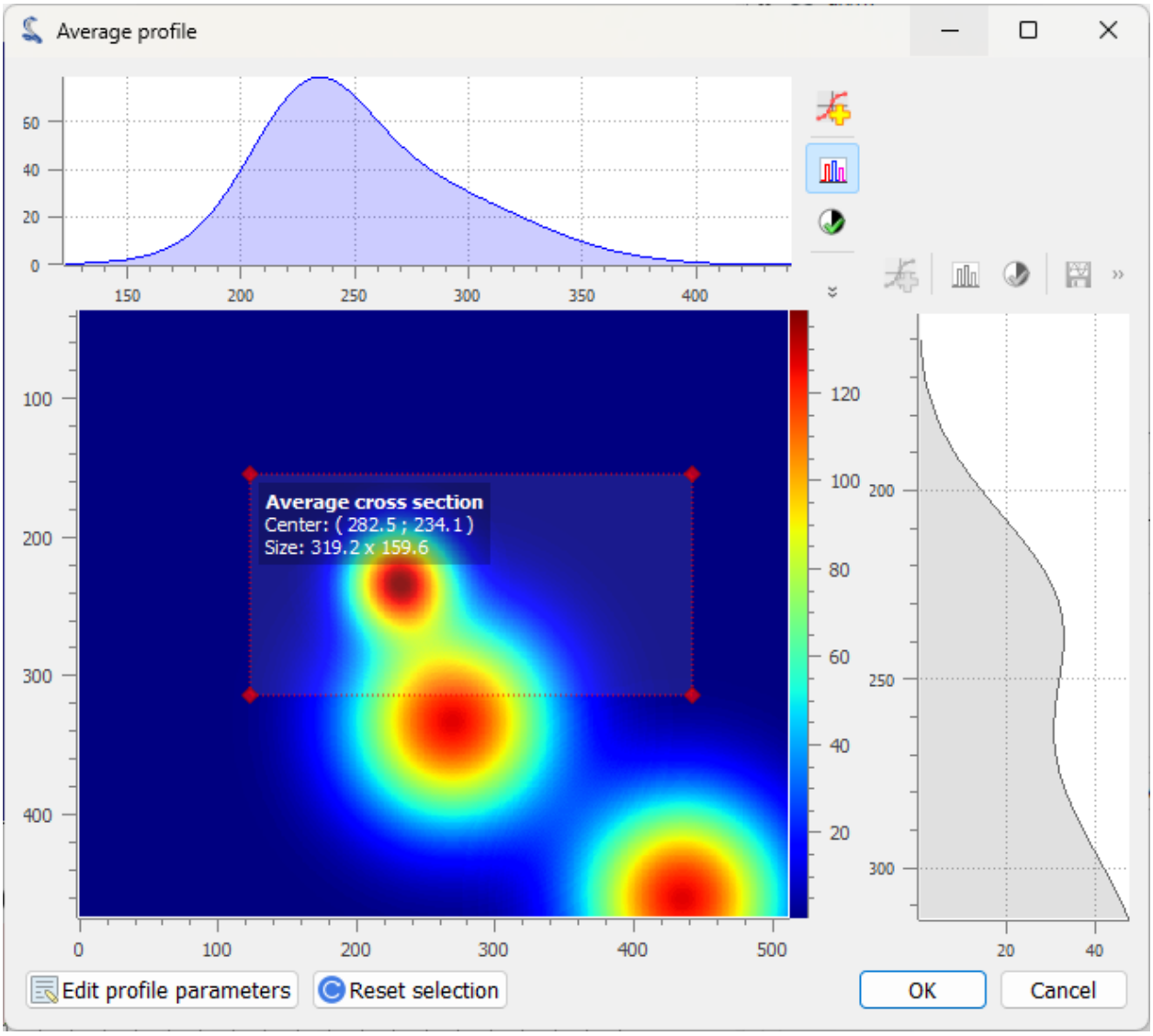


Fig. 58 – Boîte de dialogue d'extraction de profil moyen : la zone est définie par un rectangle. Les paramètres peuvent être également définis manuellement (bouton « Editer les paramètres du profil »).

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

Projections horizontale et verticale

Calcule les profils de projection horizontale et verticale :
- Projection horizontale : somme des valeurs des pixels le long de chaque ligne (projection sur l'axe des x).
- Projection verticale : somme des valeurs des pixels le long de chaque colonne (projection sur l'axe des y).

Barycentre

Calcule le barycentre de l'image en utilisant un algorithme adaptatif robuste.

DataLab utilise la fonction `sigima.tools.image.get_centroid_auto()` pour estimer le barycentre de l'image.

Cette fonction combine la robustesse d'une approche basée sur la transformée de Fourier (comme discuté par Weisshaar et al.) avec des mécanismes de secours conçus pour gérer les cas pathologiques, tels que :
- formes tronquées ou asymétriques,
- objets décentrés,
- bruit de fond important.

En interne, `sigima.tools.image.get_centroid_auto()` compare le résultat de Fourier avec deux estimateurs alternatifs (une méthode basée sur le profil projeté `sigima.tools.image.get_projected_profile_centroid()` et un barycentre standard de *scikit-image*), et sélectionne le plus cohérent.

Cette stratégie garantit des résultats précis et stables sur une large gamme de types d'images, allant des données de laboratoire propres aux acquisitions bruyantes ou partielles.

Centre du cercle minimum

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

Détection de pics 2D

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

Voir aussi :

Voir *Détection de pics 2D* pour plus de détails sur l'algorithme et les paramètres associés.

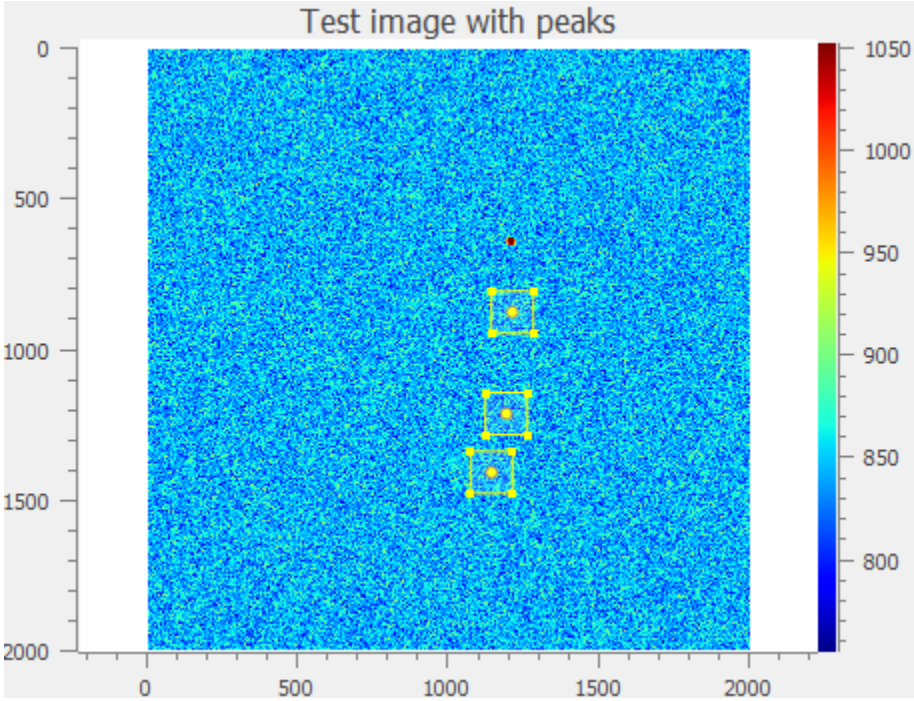


FIG. 59 – Exemple de détection de pics 2D.

Détection de contours

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

Voir aussi :

Voir *Détection de contours* pour plus de détails sur l'algorithme et les paramètres associés.

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

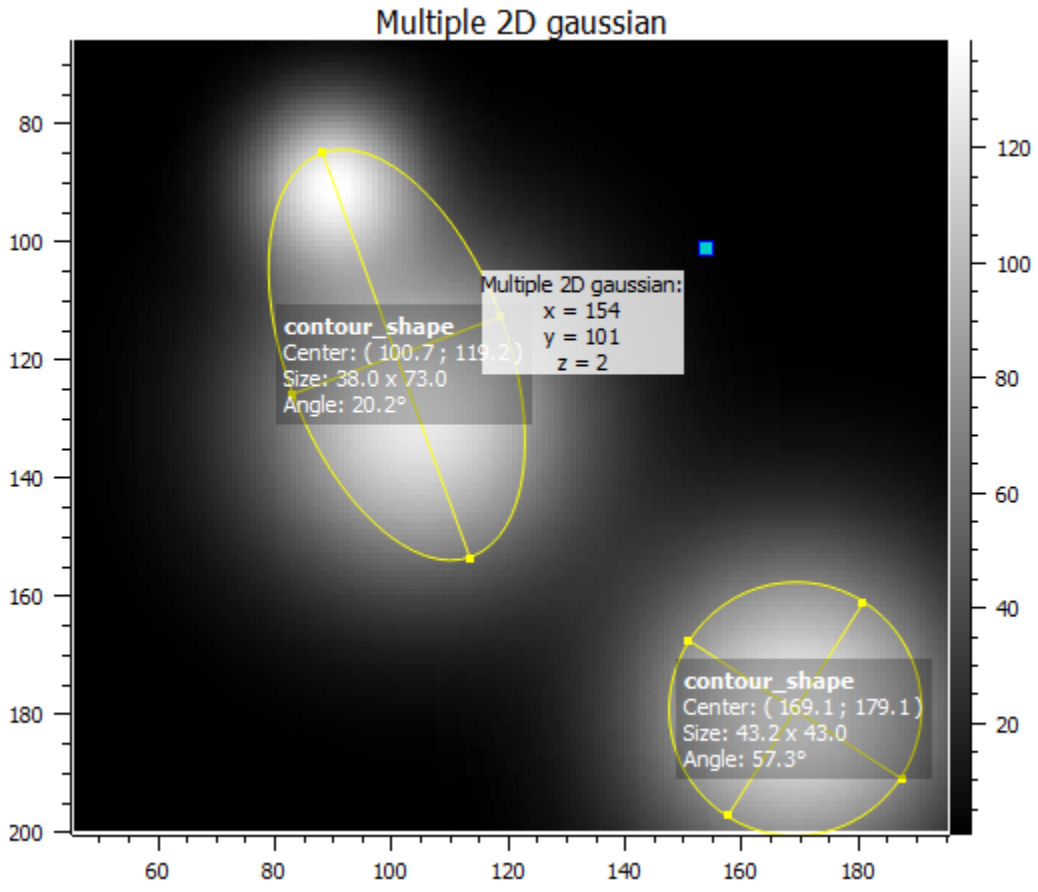


FIG. 60 – Exemple de détection de contours.

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.

Note : Création automatique de ROIs pour les fonctionnalités de détection

Toutes les fonctionnalités de détection (détection de pics 2D, détection de contours, transformée de Hough circulaire et détection de taches) prennent en charge la création automatique de ROIs autour des objets détectés. Cette fonctionnalité :

- Crée des ROIs rectangulaires ou circulaires autour de chaque objet détecté
- Dimensionne automatiquement les ROIs en fonction de la distance minimale entre les détections
- Nécessite au moins 2 objets détectés pour déterminer la taille appropriée de la ROI
- Permet le traitement ultérieur des régions détectées individuellement

Pour utiliser cette fonctionnalité, activez « Créer des régions d'intérêt » dans la boîte de dialogue des paramètres de détection et choisissez la géométrie de ROI souhaitée.

Afficher les résultats

Affiche les résultats de toutes les analyses effectuées sur les images sélectionnées. Cela affiche le même tableau que celui affiché après avoir effectué un calcul.

Étiquette des résultats

Active ou désactive la visibilité des étiquettes de résultats sur le graphique. Lorsqu'il est activé, cet élément de menu cochable affiche les annotations des résultats (telles que les marqueurs de centroïde, les contours détectés, les cercles de taches ou d'autres formes d'analyse) directement sur le graphique de l'image.

Cette option est synchronisée entre les panneaux Signal et Image et persiste entre les sessions. Elle n'est activée que lorsque des résultats sont disponibles pour l'image sélectionnée.

Tracer les résultats

Trace les résultats des analyses effectuées 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).

2.3.8 Options d'affichage pour les images

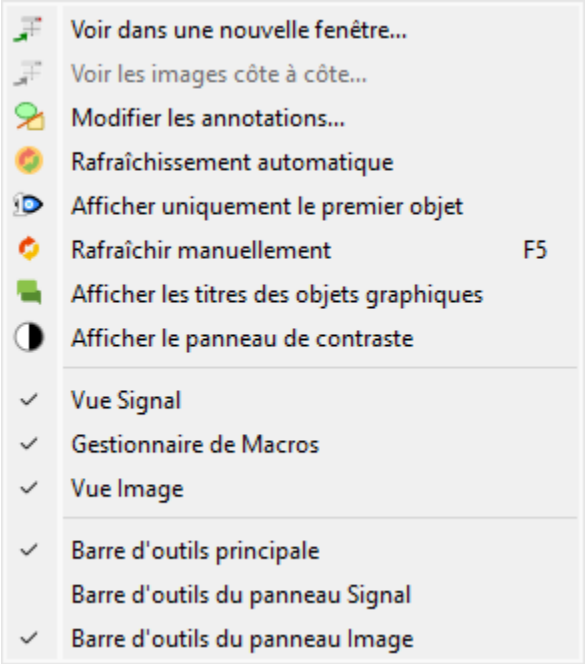


Fig. 61 – Capture d'écran du menu « Affichage ».

Lorsque le « Panneau Image » est sélectionné, les menus et barres d'outils sont mis à jour pour fournir les actions liées aux images.

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.

Voir dans une nouvelle fenêtre

Ouvre une nouvelle fenêtre pour visualiser les images sélectionnées.

Cette option permet d'afficher les images sélectionnées dans une nouvelle fenêtre, dans laquelle vous pouvez visualiser les données plus confortablement (par exemple, en maximisant la fenêtre) et ajouter ou modifier des annotations.

Lorsque vous cliquez sur le bouton « Annotations » dans la barre d'outils de la nouvelle fenêtre, le mode d'édition des annotations est activé (voir la section « Modifier les annotations » ci-dessous).

Voir les images côte à côte

Ouvre une nouvelle fenêtre pour visualiser les images sélectionnées côte à côte.

Cette option permet d'afficher les images sélectionnées dans une nouvelle fenêtre, disposées en grille, avec un zoom et un panoramique synchronisés. Cela est utile pour comparer plusieurs images simultanément.

Modifier les annotations

Ouvre une nouvelle fenêtre pour modifier les annotations.

Cette option vous permet de modifier les annotations des images sélectionnées dans une fenêtre séparée. Cela équivaut à sélectionner l'option « Voir dans une nouvelle fenêtre » puis à cliquer sur le bouton « Annotations » dans la barre d'outils de la nouvelle fenêtre.

Les annotations sont utilisées pour ajouter du texte, des lignes, des rectangles, des ellipses et d'autres formes géométriques aux images. Elles sont utiles pour mettre en évidence des régions d'intérêt, ajouter des commentaires, marquer des points, etc.

Note : Les annotations sont enregistrées dans les métadonnées de l'image, elles sont donc persistantes et seront affichées chaque fois que vous visualiserez l'image.

Le mode opératoire typique pour modifier les annotations est le suivant :

1. Sélectionnez l'option « Modifier les annotations ».
2. Dans la nouvelle fenêtre, ajoutez des annotations en cliquant sur les boutons correspondants en bas de la fenêtre.
3. Personnalisez les annotations en modifiant leurs propriétés (par exemple, le texte, la couleur, la position, etc.) en utilisant l'option « Paramètres » dans le menu contextuel des annotations.
4. Quand vous avez terminé, cliquez sur le bouton « OK » pour enregistrer les annotations. Cela fermera la fenêtre et les annotations seront enregistrées dans les métadonnées de l'image et seront affichées dans la fenêtre principale.

Note : Les annotations peuvent être copiées d'une image à une autre en utilisant les fonctionnalités « copier/coller les métadonnées ».

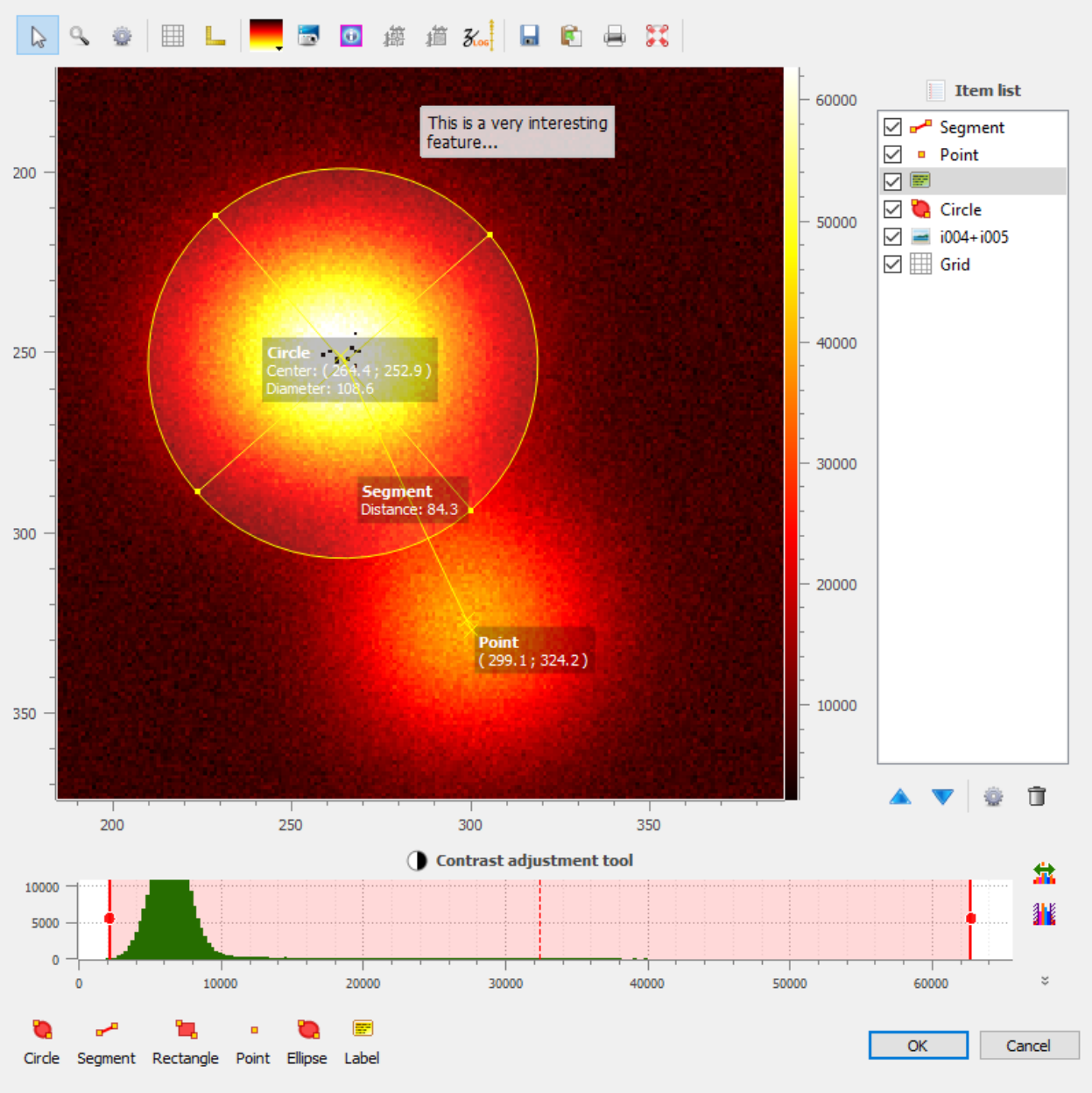


FIG. 62 – Les annotations peuvent être ajoutées dans la vue séparée.

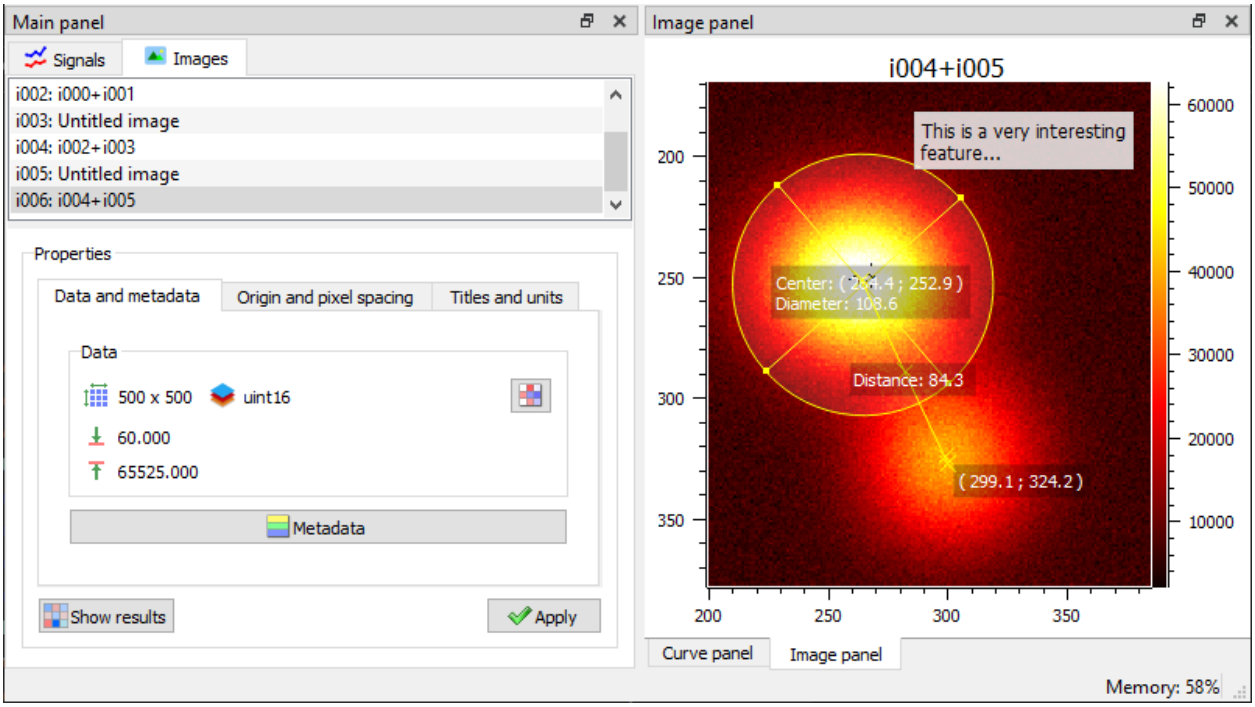


FIG. 63 – Les annotations font désormais partie des métadonnées de l'image.

Rafraîchissement automatique

Rafraîchit automatiquement la visualisation quand les données changent :

- Si activé (par défaut), la visualisation est automatiquement rafraîchie quand les données changent.
- Si désactivé, la visualisation n'est pas rafraîchie tant que vous ne l'avez pas rafraîchie manuellement en utilisant l'entrée de menu « Rafraîchir manuellement ».

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.

Afficher seulement la première image sélectionnée

Basculer entre l'affichage de toutes les images sélectionnées ou seulement de la première.

Lorsque cette option est activée, seule la première image sélectionnée est affichée dans la vue du graphique. Cela peut être utile lorsque plusieurs images sont sélectionnées et qu'il est (temporairement) préférable de se concentrer sur une seule image.

Rafraîchir manuellement

Rafraîchir la visualisation manuellement.

Cela déclenche un rafraîchissement de la visualisation. C'est utile quand le rafraîchissement automatique est désactivé, ou quand vous voulez forcer un rafraîchissement de la visualisation.

Afficher le panneau de contraste

Affiche/cache le panneau de réglage du contraste.

Afficher les titres des objets graphiques

Affiche/cache les titres des objets graphiques liés aux résultats d'analyse et aux annotations.

Cette option vous permet d'afficher ou de masquer les titres des objets graphiques (par exemple, les titres des résultats d'analyse ou des annotations). Masquer les titres peut être utile lorsque vous voulez visualiser les données sans être perturbé, ou s'il y a trop de titres et qu'ils se chevauchent.

2.3.9 Détection de pics 2D

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

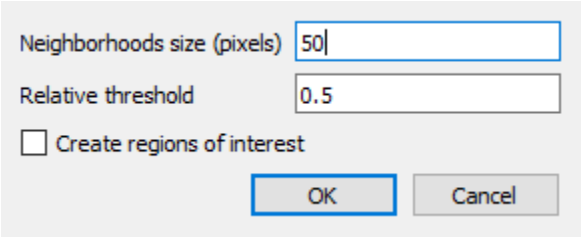


Fig. 64 – Paramètres de détection de pics 2D.

La fonctionnalité s'utilise de la façon suivante :

- Créer ou ouvrir une image dans l'espace de travail de DataLab
- Sélectionner « Détection de pics 2D » dans le menu Analyse »
- Saisir les paramètres « Taille de voisinage » et « Seuil relatif »
- Optionnellement, activer « Créer des régions d'intérêt » pour créer automatiquement des ROIs autour de chaque pic détecté :
  - Choisir la géométrie de la ROI : « Rectangle » ou « Cercle »
  - La taille de la ROI est automatiquement calculée en fonction de la distance minimale entre les pics détectés (pour éviter le chevauchement)
  - Cette fonctionnalité nécessite au moins 2 pics détectés
  - Les ROIs créées peuvent être utiles pour le traitement ultérieur de chaque zone de pic (par exemple, détection de contours, mesures, etc.)

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

FIG. 65 – Résultats d'une détection de pics 2D (voir le test « peak2d_app.py »)

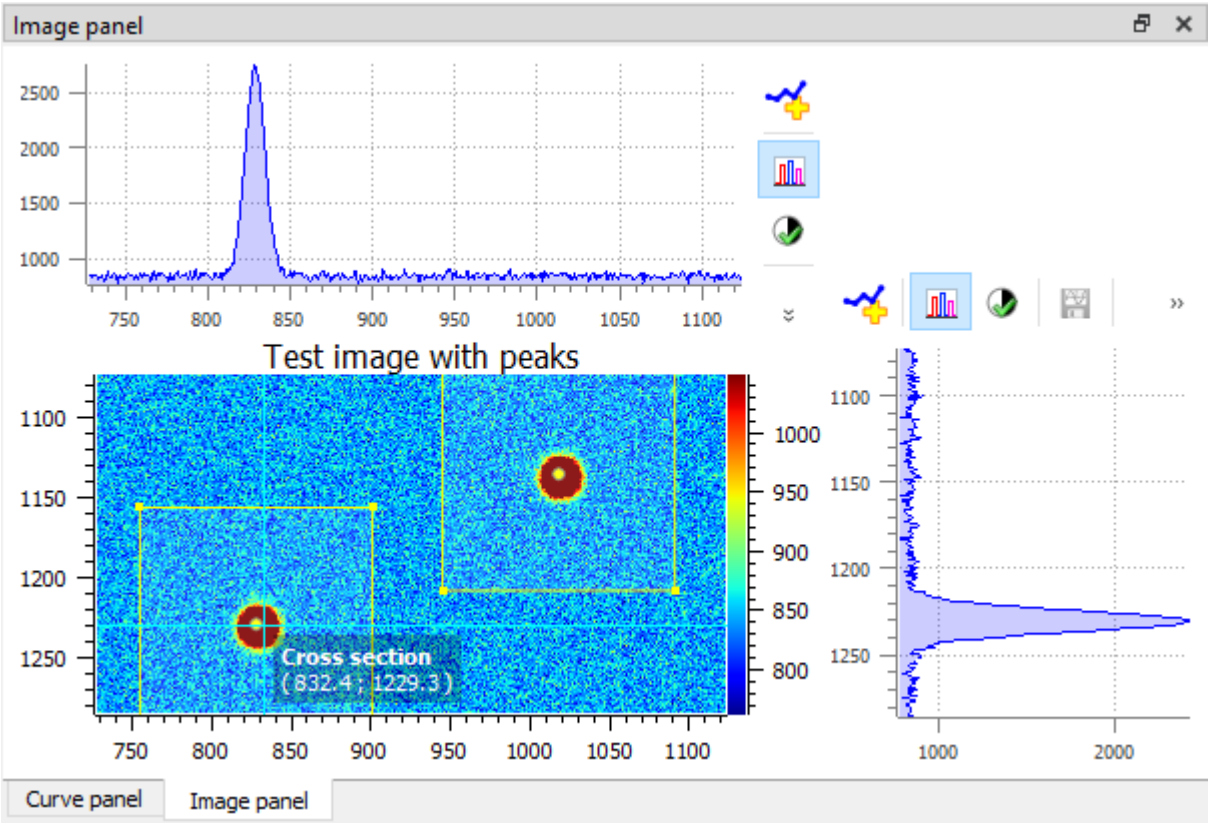


FIG. 66 – Exemple de détection de pics 2D.

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
- « Créer des régions d'intérêt » : si activé, crée automatiquement des ROIs autour de chaque pic détecté (nécessite au moins 2 pics)
- « Géométrie de la ROI » : forme des ROIs (« Rectangle » ou « Cercle »)

La fonctionnalité est basée sur la fonction `get_2d_peaks_coords` du module `sigima.tools` :

```
@check_2d_array(non_constant=True)
def get_2d_peaks_coords(
    data: np.ndarray, size: int | None = None, level: float = 0.5
) -> np.ndarray:
    """Detect peaks in image data, return coordinates.

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

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

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

    Returns:
        Coordinates of peaks
    """
    if size is None:
        size = max(min(data.shape) // 40, 50)
    data_max = spi.maximum_filter(data, size)
    data_min = spi.minimum_filter(data, size)
    data_diff = data_max - data_min
    diff = (data_max - data_min) > get_absolute_level(data_diff, level)
    maxima = data == data_max
    maxima[diff == 0] = 0
    labeled, _num_objects = spi.label(maxima)
    slices = spi.find_objects(labeled)
    coords = []
    for dy, dx in slices:
        x_center = int(0.5 * (dx.start + dx.stop - 1))
        y_center = int(0.5 * (dy.start + dy.stop - 1))
        coords.append((x_center, y_center))
```

(suite sur la page suivante)

(suite de la page précédente)

```
    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)
```

2.3.10 Détection de contours

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

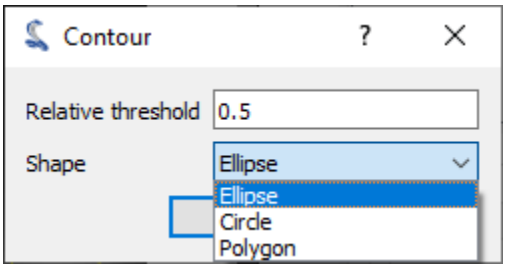


Fig. 67 – Paramètres de détection de contours.

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 Analyse »
- Saisir le paramètre « Forme » (« Ellipse », « Cercle » ou « Polygone »)
- Optionnellement, activer « Créer des régions d'intérêt » pour créer automatiquement des ROIs autour de chaque contour détecté :
  - Choisir la géométrie de la ROI : « Rectangle » ou « Cercle »
  - La taille de la ROI est automatiquement calculée en fonction de la distance minimale entre les contours détectés (pour éviter le chevauchement)
  - Cette fonctionnalité nécessite au moins 2 contours détectés
  - Les ROIs créées peuvent être utiles pour le traitement ultérieur de chaque zone de contour (par exemple, détection de pics, mesures, etc.)

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 `sigima.proc` :

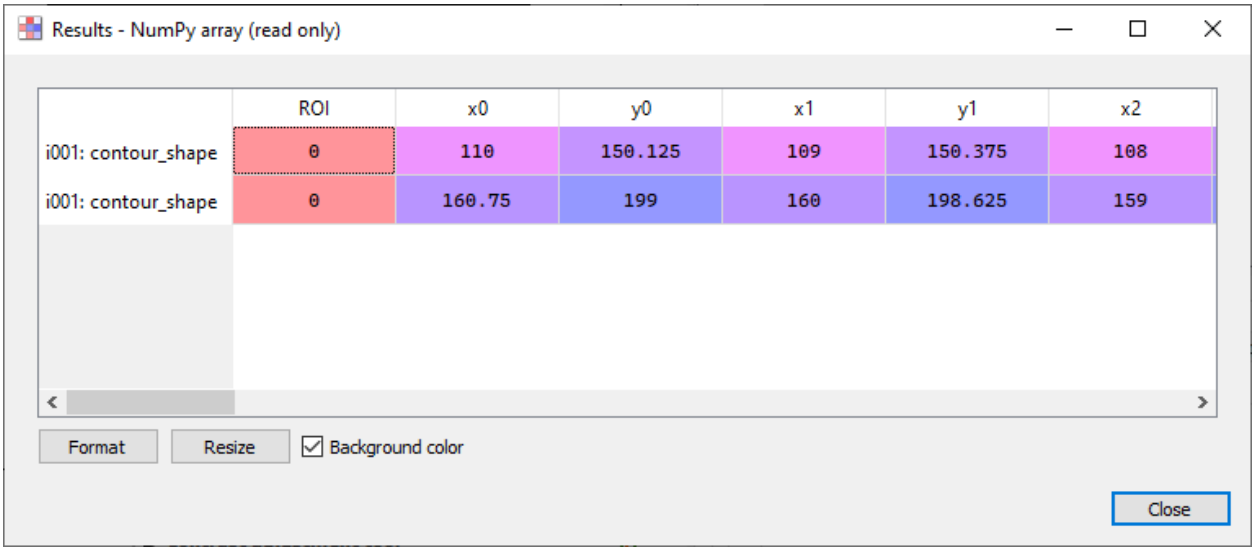


FIG. 68 – Résultats de la détection de contours (cf. test « contour_app.py »)

```
def get_contour_shapes(
    data: np.ndarray | ma.MaskedArray,
    shape: ContourShape = ContourShape.ELLIPSE,
    level: float = 0.5,
) -> np.ndarray:
    """Find iso-valued contours in a 2D array, above relative level (.5 means␣
↪FWHM).

    Args:
        data: Input data
        shape: Shape to fit. Default is ELLIPSE
        level: Relative level (default: 0.5)

    Returns:
        Coordinates of shapes fitted to contours
    """
    # pylint: disable=too-many-locals
    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 == ContourShape.CIRCLE:
            model = measure.CircleModel()
            if model.estimate(contour):
                yc, xc, r = model.params
```

(suite sur la page suivante)

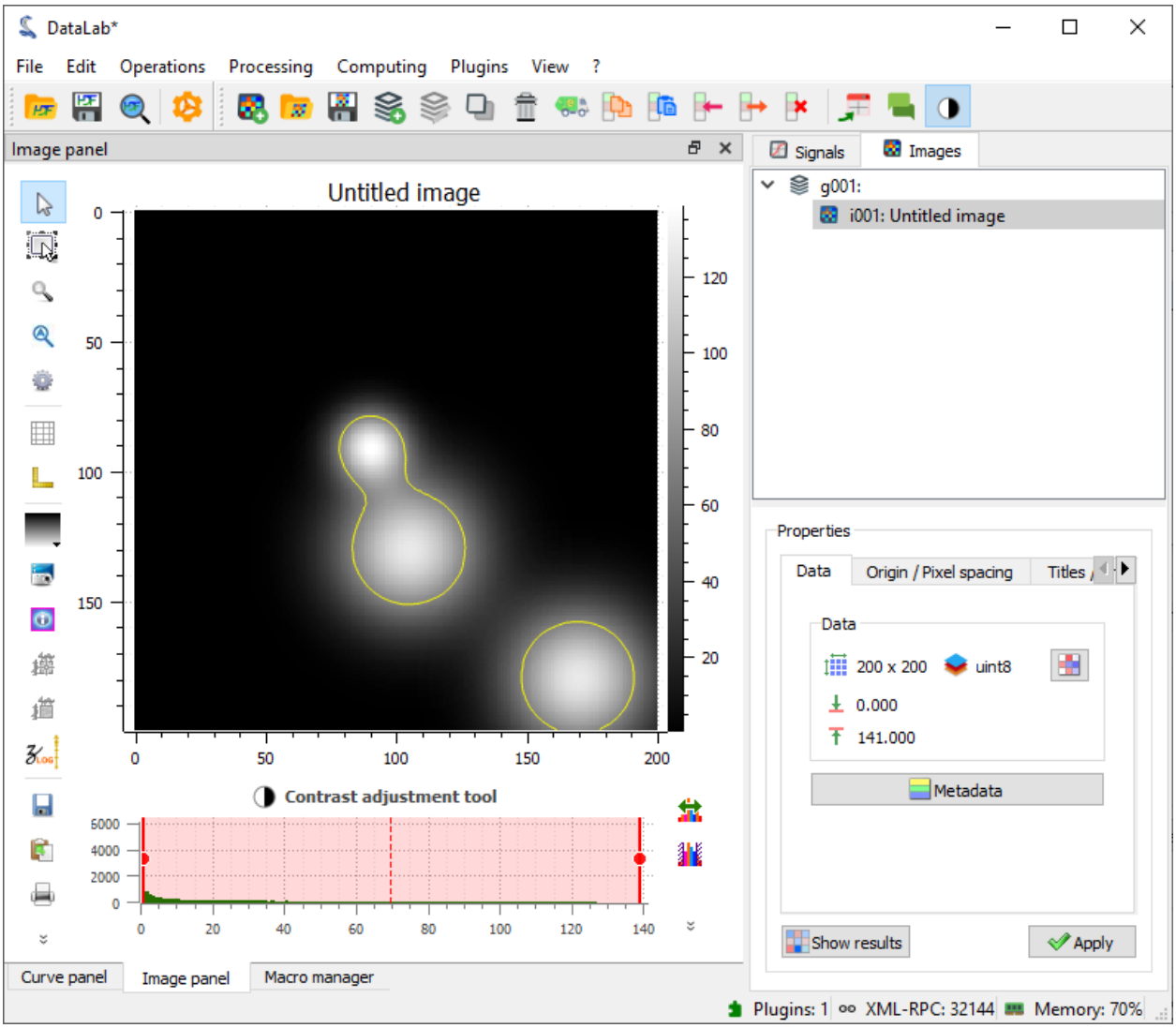


FIG. 69 – Exemple de détection de contours.

(suite de la page précédente)

```
                if r <= 1.0:
                    continue
                coords.append([xc, yc, r])
        elif shape == ContourShape.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 == ContourShape.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 == ContourShape.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)
```

2.4 Validation

DataLab est une plateforme d'analyse et de visualisation de données scientifiques qui peut être utilisée dans diverses disciplines scientifiques, y compris la biologie, la physique et l'astronomie. Le point commun de toutes ces disciplines est la nécessité de valider les résultats de l'analyse informatique par rapport à des données de référence. Il s'agit d'une étape critique dans la méthode scientifique et elle est essentielle pour la reproductibilité et la confiance dans les résultats. C'est ce que nous appelons la **validation technique**.

DataLab est également utilisé dans des applications industrielles, où la validation des résultats est essentielle pour garantir la qualité du processus, mais une gamme plus large de validation est nécessaire pour s'assurer qu'un maximum de cas d'utilisation sont couverts d'un point de vue fonctionnel. C'est ce que nous appelons la **validation fonctionnelle**.

Ainsi, la validation de DataLab peut être classée en deux types :

- La **validation technique** : garantit que les résultats de l'analyse informatique sont précis et fiables.
- La **validation fonctionnelle** : vérifie que le logiciel se comporte comme prévu dans une variété de cas d'utilisation (suite de tests unitaires automatisés classique).

2.4.1 Validation technique

La validation technique de DataLab est basée sur deux concepts clés : les **données de référence** et la **validation analytique**.

Données de référence

Les données de référence sont des données connues pour être correctes. Elles sont utilisées pour valider les résultats de l'analyse informatique.

Dans DataLab, les données de référence peuvent être obtenues à partir de diverses sources, y compris :
- Données expérimentales
- Données simulées
- Données synthétiques
- Données provenant d'une source de confiance

Validation analytique

La validation analytique est le processus de comparaison des résultats de l'analyse informatique avec les données de référence. Cela est fait pour s'assurer que les résultats sont précis et fiables.

Dans DataLab, la validation analytique est mise en œuvre à l'aide de diverses techniques, y compris :
- Validation croisée avec un modèle analytique (provenant d'une source de confiance, par exemple SciPy ou NumPy)
- Analyse statistique
- Inspection visuelle
- Examen par un expert

Périmètre

Le périmètre de la validation technique dans DataLab inclut toutes les fonctions de calcul qui opèrent sur les objets de signal et d'image de DataLab (c'est-à-dire `sigima.objects.SignalObj` et `sigima.objects.ImageObj`).

Cela inclut des fonctions pour (toutes les fonctions sont nommées `compute_<function_name>`) :
- Le traitement du signal (`sigima.proc.signal`)
- Le traitement d'image (`sigima.proc.image`)

Implémentation

Les tests sont implémentés en utilisant le cadre pytest.

Lors de l'écriture d'un nouveau test de validation technique, les règles suivantes doivent être suivies concernant la fonction de test :
- La fonction de test doit être nommée :
  - `test_signal_<function_name>` pour les fonctions de calcul de signal
  - `test_image_<function_name>` pour les fonctions de calcul d'image

Note : Le préfixe `signal` ou `image` est utilisé pour indiquer le type d'objet sur lequel la fonction opère. Il peut être omis si la fonction opère exclusivement sur un type d'objet (par exemple, `test_adjust_gamma` est la fonction de test pour la fonction `compute_adjust_gamma`, qui opère sur des images).

- La fonction de test doit être marquée avec le décorateur `@pytest.mark.validation`.

En suivant ces règles, on s'assure que :

- Les tests sont facilement identifiés comme des tests de validation technique.
- Les tests peuvent être exécutés séparément à l'aide de l'interface en ligne de commande (voir *Exécuter les tests de validation technique*).
- Les tests sont automatiquement découverts pour synthétiser l'état de validation des fonctions de calcul (voir *Etat de validation de DataLab*).

Exécution des tests

Dans DataLab, les tests de validation technique sont disséminés dans la suite de tests du projet, mais ils peuvent également être exécutés séparément à l'aide de l'interface en ligne de commande.

Voir aussi :

Voir le paragraphe *Exécuter les tests de validation technique* pour plus d'informations sur la façon d'exécuter les tests de validation technique.

2.4.2 Validation fonctionnelle

Stratégie

La validation fonctionnelle de DataLab est basée sur une stratégie de test classique, avec un fort accent sur les tests automatisés. À l'exception d'un ou deux tests manuels (par exemple, un test de charge), tous les tests sont automatisés (plus de 99% des tests sont automatisés).

L'écriture des tests suit le principe TDD (Test-Driven Development) :

- Losqu'une *nouvelle fonctionnalité* est développée, le développeur écrit d'abord les tests. Les tests sont ensuite exécutés pour s'assurer qu'ils échouent. Le développeur implémente ensuite la fonctionnalité, et les tests sont exécutés à nouveau pour s'assurer qu'ils réussissent.
- Lorsqu'un *bug est signalé*, le développeur écrit un test qui reproduit le bug. Le test est exécuté pour s'assurer qu'il échoue. Le développeur corrige ensuite le bug, et le test est exécuté à nouveau pour s'assurer qu'il réussit.

Selon le niveau d'abstraction, des tests unitaires et/ou des tests d'application sont écrits. Lors de l'écriture des deux types de tests, le développeur commence par les tests unitaires puis écrit les tests d'application.

Types de tests

La validation fonctionnelle de DataLab repose sur deux types principaux de tests :

- Les **tests unitaires** (scripts de test nommés `*_unit_test.py`) : testent les fonctions ou méthodes individuelles. Tous les tests unitaires sont automatisés.
- Les **tests applicatifs** (scripts de test nommés `*_app_test.py`) : testent l'interaction entre les composants (tests d'intégration), ou l'application dans son ensemble. Tous les tests applicatifs sont automatisés.

Implémentation

Les tests sont implémentés en utilisant le framework pytest. De nombreux tests existants peuvent être dérivés pour créer de nouveaux tests.

Exécution des tests

Pour exécuter les tests, le développeur utilise l'interface en ligne de commande. Voir la section *Exécuter l'ensemble des tests* pour plus d'informations sur la façon d'exécuter les tests de validation fonctionnelle.

2.4.3 Etat de validation de DataLab

Validation fonctionnelle

Dans DataLab, la validation fonctionnelle est basée sur une stratégie de test classique (voir *Validation fonctionnelle*).

La couverture de test est d'environ 90%, avec plus de 200 tests.

Validation technique

Ce paragraphe fournit l'état de validation des fonctions de calcul dans DataLab (c'est ce que nous appelons validation technique, voir *Validation technique*).

Note : Il s'agit d'un travail en cours : les tableaux ci-dessous sont mis à jour en continu à mesure que de nouvelles fonctions sont validées ou que le code de test est adapté (les tableaux sont générés à partir du code de test). Certaines fonctions sont déjà validées mais n'apparaissent pas encore dans la liste ci-dessous, tandis que d'autres sont encore en cours de validation.

Avertissement : L'état de validation ne doit pas être confondu avec la couverture de test. L'état de validation indique si la fonction a été validée par rapport à des données de référence ou des modèles analytiques. La couverture de test indique le pourcentage du code qui est exécuté par la suite de tests, mais elle ne prend pas nécessairement en compte la justesse des résultats (la couverture de test de DataLab est d'environ 90%).

Tableau 3 – Statistiques de validation

Catégorie	Signal	Image	Total
Nombre de fonctions de calcul	95	115	210
Nombre de fonctions de calcul validées	95	115	210
Pourcentage de fonctions de calcul validées	100%	100%	100%

Fonctions de calcul signal

Le tableau ci-dessous montre l'état de validation des fonctions de calcul signal dans DataLab. Il est généré automatiquement à partir du code source.

Tableau 4: Etat de validation de DataLab

Fonctions de calcul	Description	Fonction de test
`absolute`	Compute absolute value with `numpy.absolute`	test_signal_absolute
`add_gaussian_noise`	Add normal noise to the input signal	test_signal_add_gaussian_noise
`add_poisson_noise`	Add Poisson noise to the input signal	test_signal_add_poisson_noise

suite sur la page suivante

Tableau 4 – suite de la page précédente

Fonctions de calcul	Description	Fonction de test
add_uniform_noise	Add uniform noise to the input signal	test_signal_add_uniform_noise
addition	Compute the element-wise sum of multiple signals	test_signal_addition
addition_constant	Compute the sum of a signal and a constant value	test_signal_addition_constant
allan_deviation	Calculer la déviation d'Allan	test_signal_allan_deviation
allan_variance	Compute Allan variance	test_signal_allan_variance
apply_window	Calculer la fenêtrage	test_signal_apply_window
arithmetic	Perform an arithmetic operation on two signals	test_signal_arithmetic
astype	Convert data type	test_signal_astype
average	Compute the element-wise average of multiple signals	test_signal_average
bandpass	Compute band-pass filter	test_signal_bandpass
bandstop	Compute band-stop filter	test_signal_bandstop
bandwidth_3db	Compute bandwidth at -3 dB	test_signal_bandwidth_3db
calibration	Compute linear calibration	test_signal_calibration
cdf_fit	Compute CDF fit	test_signal_cdf_fit
clip	Compute maximum data clipping	test_signal_clip
complex_from_magnitude_pha	Combine magnitude and phase signals into a complex signal	test_signal_complex_from_magnitude_phase
complex_from_real_imag	Combine two real signals into a complex signal using real + i * imag	test_signal_complex_from_real_imag
contrast	Calculer le contraste	test_signal_contrast
convolution	Compute convolution of two signals	test_signal_convolution
deconvolution	Calculer la déconvolution	test_signal_deconvolution
derivative	Calculer la dérivée	test_signal_derivative
detrending	Detrend data	test_signal_detrending
difference	Compute the element-wise difference between two signals	test_signal_difference
difference_constant	Compute the difference between a signal and a constant value	test_signal_difference_constant
division	Compute the element-wise division between two signals	test_signal_division
division_constant	Compute the division of a signal by a constant value	test_signal_division_constant
dynamic_parameters	Compute Dynamic parameters	test_dynamic_parameters
evaluate_fit	Evaluate fit function from src1 on the x-axis of src2	test_signal_evaluate_fit
exp	Compute exponential with numpy.exp	test_signal_exp
exponential_fit	Calculer l'ajustement exponentiel	test_signal_exponential_fit
extract_pulse_features	Extract pulse features	test_signal_extract_pulse_features
extract_roi	Extract single region of interest from data	test_signal_extract_roi
extract_rois	Extract multiple regions of interest from data	test_signal_extract_rois
fft	Calculer la FFT	test_signal_fft
full_width_at_y		test_signal_full_width_at_y
fw1e2	Compute FW at $1/e^2$	test_signal_fw1e2
fwhm	Fonctions de calcul	test_signal_fwhm
gaussian_filter	Compute gaussian filter	test_signal_gaussian_filter
gaussian_fit	Calculer la déviation d'Allan	test_signal_gaussian_fit

suite sur la page suivante

Tableau 4 – suite de la page précédente

Fonctions de calcul	Description	Fonction de test
hadamard_variance	Compute Hadamard variance	test_signal_hadamard_variance
highpass	Compute high-pass filter	test_signal_highpass
histogram	Calculer l'histogramme	test_signal_histogram
ifft	Compute the inverse FFT	test_signal_ifft
imag	Compute imaginary part	test_signal_imag
integral	Calculer l'intégrale	test_signal_integral
interpolate	Interpolate data	test_signal_interpolate
inverse	Compute the element-wise inverse of a signal	test_signal_inverse
linear_fit	Calculer l'ajustement linéaire	test_signal_linear_fit
log10	Compute Log10 with numpy.log10	test_signal_log10
lorentzian_fit	Calculer l'ajustement lorentzien	test_signal_lorentzian_fit
lowpass	Compute low-pass filter	test_signal_lowpass
magnitude_spectrum	Compute magnitude spectrum	test_signal_magnitude_spectrum
modified_allan_variance	Compute Modified Allan variance	test_signal_modified_allan_variance
moving_average	Compute moving average	test_signal_moving_average
moving_median	Compute moving median	test_signal_moving_median
normalize	Normalize data	test_signal_normalize
offset_correction	Correct offset : subtract the mean value of the signal in the specified range	test_signal_offset_correction
overlapping_allan_variance	Compute Overlapping Allan variance	test_signal_overlapping_allan_variance
peak_detection	Peak detection	test_signal_peak_detection
phase	Compute the phase (argument) of a complex signal	test_signal_phase
phase_spectrum	Compute phase spectrum	test_signal_phase_spectrum
piecewiseexponential_fit	Calculer l'ajustement exponentiel par morceaux (augmentation-décroissance)	test_signal_piecewiseexponential_fit
planckian_fit	Calculer l'ajustement de Planck	test_signal_planckian_fit
polynomial_fit	Calculer l'ajustement polynomial	test_polynomial_fit
power	Compute power with numpy.power	test_signal_power
product	Compute the element-wise product of multiple signals	test_signal_product
product_constant	Compute the product of a signal and a constant value	test_signal_product_constant
psd	Compute power spectral density	test_signal_psd
quadratic_difference	Compute the normalized difference between two signals	test_signal_quadratic_difference
real	Calculer la partie réelle	test_signal_real
resampling	Resample data	test_signal_resampling
reverse_x	Reverse x-axis	test_signal_reverse_x
sampling_rate_period	Compute sampling rate and period	test_signal_sampling_rate_period
sigmoid_fit	Calculer l'ajustement sigmoïde	test_signal_sigmoid_fit
signals_to_image	Combine multiple signals into an image	test_signal_signals_to_image
sinusoidal_fit	Calculer l'ajustement sinusoïdal	test_sinusoidal_fit
sqrt	Compute square root with numpy.sqrt	test_signal_sqrt
standard_deviation	Compute the element-wise standard deviation of multiple signals	test_signal_standard_deviation
stats	Compute statistics on a signal	test_signal_stats_unit
time_deviation	Compute Time Deviation (TDEV)	test_signal_time_deviation

suite sur la page suivante

Tableau 4 – suite de la page précédente

Fonctions de calcul	Description	Fonction de test
`to_cartesian`	Convert polar coordinates to Cartesian coordinates	test_signal_to_cartesian
`to_polar`	Convert Cartesian coordinates to polar coordinates	test_signal_to_polar
`total_variance`	Compute Total variance	test_signal_total_variance
`transpose`	Transpose signal (swap X and Y axes)	test_signal_transpose
`twohalfgaussian_fit`	Compute two-half-Gaussian fit	test_signal_twohalfgaussian_fit
`voigt_fit`	Calculer l'ajustement de Voigt	test_signal_voigt_fit
`wiener`	Compute Wiener filter	test_signal_wiener
`x_at_minmax`		test_signal_x_at_minmax
`x_at_y`		test_signal_x_at_y
`xy_mode`	Simulate the X-Y mode of an oscilloscope	test_signal_xy_mode
`y_at_x`		test_signal_y_at_x
`zero_padding`	Compute zero padding	test_signal_zero_padding

Fonctions de calcul image

Le tableau ci-dessous montre l'état de validation des fonctions de calcul image dans DataLab. Il est généré automatiquement à partir du code source.

Tableau 5: Etat de validation des fonctions de calcul image

Fonctions de calcul	Description	Fonction de test
`absolute`	Compute absolute value with `numpy.absolute`	test_image_absolute
`add_gaussian_noise`	Add Gaussian (normal) noise to the input image	test_image_add_gaussian_noise
`add_poisson_noise`	Add Poisson noise to the input image	test_image_add_poisson_noise
`add_uniform_noise`	Add uniform noise to the input image	test_image_add_uniform_noise
`addition`	Add images in the list and return the result image object	test_image_addition
`addition_constant`	Add **dst** and a constant value and return the new result image object	test_image_addition_constant
`adjust_gamma`	Gamma correction	test_adjust_gamma
`adjust_log`	Calculer l'ajustement logarithmique	test_adjust_log
`adjust_sigmoid`	Calculer l'ajustement sigmoïde	test_adjust_sigmoid
`arithmetic`	Compute arithmetic operation on two images	test_image_arithmetic
`astype`	Convert image data type	test_image_astype
`average`	Compute the average of images in the list and return the result image object	test_image_average
`average_profile`	Compute horizontal or vertical average profile	test_average_profile
`binning`	Binning : image pixel binning (or aggregation)	test_binning
`black_tophat`	Compute Black Top-Hat	test_black_tophat
`blob_dog`	Compute blobs using Difference of Gaussian method	test_image_blob_dog

suite sur la page suivante

Tableau 5 – suite de la page précédente

Fonctions de calcul	Description	Fonction de test
blob_doh	Compute blobs using Determinant of Hessian method	test_image_blob_doh
blob_log	Compute blobs using Laplacian of Gaussian method	test_image_blob_log
blob_opencv	Compute blobs using OpenCV	test_image_blob_opencv
butterworth	Compute Butterworth filter	test_butterworth
calibration	Calculer la calibration polynomiale	test_image_calibration
canny	Compute Canny filter	test_canny
centroid	Compute centroid	test_image_centroid
clip	Apply clipping	test_image_clip
closing	Compute morphological closing	test_closing
complex_from_magnitude_pha	Combine magnitude and phase images into a complex image	test_image_complex_from_magnitude_phase
complex_from_real_imag	Combine two real images into a complex image using real + i * imag	test_image_complex_from_real_imag
contour_shape	Compute contour shape	test_contour_shape
convolution	Convolve an image with a kernel	test_image_convolution
deconvolution	Deconvolve a kernel from an image using Fast Fourier Transform (FFT)	test_image_deconvolution
denoise_bilateral	Compute bilateral filter denoising	test_denoise_bilateral
denoise_tophat	Denoise using White Top-Hat	test_denoise_tophat
denoise_tv	Compute Total Variation denoising	test_denoise_tv
denoise_wavelet	Compute Wavelet denoising	test_denoise_wavelet
difference	Compute difference between two images	test_image_difference
difference_constant	Subtract a constant value from an image and return the new result image object	test_image_difference_constant
dilation	Calculer la dilatation	test_dilation
division	Compute division between two images	test_image_division
division_constant	Divide an image by a constant value and return the new result image object	test_image_division_constant
enclosing_circle	Compute minimum enclosing circle	test_image_enclosing_circle
equalize_adapthist	Adaptive histogram equalization	test_equalize_adapthist
equalize_hist	Histogram equalization	test_equalize_hist
erase	Erase an area of the image using the mean value of the image	test_erase
erosion	Calculer l'érosion	test_erosion
exp	Compute exponential with numpy.exp	test_image_exp
extract_roi	Extract single ROI	test_image_extract_roi
extract_rois	Extract multiple regions of interest from data	test_image_extract_rois
farid	Compute Farid filter	test_farid
farid_h	Compute horizontal Farid filter	test_farid_h
farid_v	Compute vertical Farid filter	test_farid_v
fft	Calculer la FFT	test_image_fft
flatfield	Calculer la correction de champ plat	test_flatfield
fliph	Flip data horizontally	test_image_fliph
flipv	Flip data vertically	test_image_flipv
gaussian_filter	Compute gaussian filter	test_image_gaussian_filter
gaussian_freq_filter	Apply a Gaussian filter in the frequency domain	test_gaussian_freq_filter
histogram	Compute histogram of the image data,	test_image_histogram

suite sur la page suivante

Tableau 5 – suite de la page précédente

Fonctions de calcul	Description	Fonction de test
horizontal_projection	Compute the sum of pixel intensities along each col. (projection on the x-axis)	test_image_horizontal_projection
hough_circle_peaks	Compute Hough circles	test_image_hough_circle_peaks
ifft	Compute inverse FFT	test_image_ifft
imag	Compute imaginary part	test_image_imag
inverse	Compute the inverse of an image and return the new result image object	test_image_inverse
laplace	Compute Laplace filter	test_laplace
line_profile	Compute horizontal or vertical profile	test_line_profile
log10	Compute log10 with numpy.log10	test_image_log10
log10_z_plus_n	Compute log10(z+n) with numpy.log10	test_image_log10_z_plus_n
magnitude_spectrum	Compute magnitude spectrum	test_image_magnitude_spectrum
moving_average	Compute moving average	test_image_moving_average
moving_median	Compute moving median	test_image_moving_median
normalize		test_image_normalize
offset_correction	Apply offset correction	test_image_offset_correction
opening	Compute morphological opening	test_opening
peak_detection	Compute 2D peak detection	test_image_peak_detection
phase	Compute the phase (argument) of a complex image	test_image_phase
phase_spectrum	Compute phase spectrum	test_image_phase_spectrum
prewitt	Compute Prewitt filter	test_prewitt
prewitt_h	Compute horizontal Prewitt filter	test_prewitt_h
prewitt_v	Compute vertical Prewitt filter	test_prewitt_v
product	Multiply images in the list and return the result image object	test_image_product
product_constant	Multiply **dst** by a constant value and return the new result image object	test_image_product_constant
psd	Compute power spectral density	test_image_psd
quadratic_difference	Compute quadratic difference between two images	test_image_quadratic_difference
radial_profile	Compute radial profile around the centroid	test_radial_profile
real	Calculer la partie réelle	test_image_real
resampling	Resample image to new coordinate grid using interpolation	test_image_resampling
rescale_intensity	Rescale image intensity levels	test_rescale_intensity
resize	Fonction de zoom	test_image_resize
roberts	Compute Roberts filter	test_roberts
rotate	Rotate data	test_image_rotate
rotate270	Rotate data 270°	test_image_rotate270
rotate90	Rotate data 90°	test_image_rotate90
scharr	Compute Scharr filter	test_scharr
scharr_h	Compute horizontal Scharr filter	test_scharr_h
scharr_v	Compute vertical Scharr filter	test_scharr_v
segment_profile	Compute segment profile	test_segment_profile
set_uniform_coords	Convert image to uniform coordinate system	test_set_uniform_coords
sobel	Compute Sobel filter	test_sobel
sobel_h	Compute horizontal Sobel filter	test_sobel_h
sobel_v	Compute vertical Sobel filter	test_sobel_v

suite sur la page suivante

Tableau 5 – suite de la page précédente

Fonctions de calcul	Description	Fonction de test
`standard_deviation`	Compute the element-wise standard deviation of multiple images	test_image_standard_deviation
`stats`	Compute statistics on an image	test_image_stats_unit
`threshold`	Compute the threshold, using one of the available algorithms	test_threshold
`threshold_isodata`	Compute the threshold using the Isodata algorithm with default parameters	test_threshold_isodata
`threshold_li`	Compute the threshold using the Li algorithm with default parameters	test_threshold_li
`threshold_mean`	Compute the threshold using the Mean algorithm	test_threshold_mean
`threshold_minimum`	Compute the threshold using the Minimum algorithm with default parameters	test_threshold_minimum
`threshold_otsu`	Compute the threshold using the Otsu algorithm with default parameters	test_threshold_otsu
`threshold_triangle`	Compute the threshold using the Triangle algorithm with default parameters	test_threshold_triangle
`threshold_yen`	Compute the threshold using the Yen algorithm with default parameters	test_threshold_yen
`translate`	Translate data	test_image_translate
`transpose`	Transpose image	test_image_transpose
`vertical_projection`	Compute the sum of pixel intensities along each row (projection on the y-axis)	test_image_vertical_projection
`white_tophat`	Compute White Top-Hat	test_white_tophat
`wiener`	Compute Wiener filter	test_image_wiener
`zero_padding`	Zero-padding : add zeros to image borders	test_image_zero_padding

2.5 Fonctionnalités avancées

Cette section décrit les fonctionnalités avancées de DataLab.

2.5.1 Plugins

DataLab prend en charge une architecture de plugin robuste, permettant aux utilisateurs d'étendre les fonctionnalités de l'application sans modifier son noyau. Les plugins peuvent introduire de nouveaux outils de traitement, des formats d'importation/exportation de données ou des éléments d'interface graphique personnalisés, le tout intégré de manière transparente dans la plateforme.

Qu'est-ce qu'un plugin ?

Un plugin est un module Python qui est automatiquement chargé par DataLab au démarrage. Il peut définir de nouvelles fonctionnalités ou modifier des fonctionnalités existantes.

Pour être reconnu comme un plugin, le fichier doit :

- Être un module Python dont le nom **commence par** `datalab_` (par exemple `datalab_myplugin.py`),
- Contenir une classe qui **hérite de** `datalab.plugins.PluginBase`,
- Inclure un attribut de classe nommé `PLUGIN_INFO`, qui doit être une instance de `datalab.plugins.PluginInfo`.

Cet objet *PLUGIN_INFO* est utilisé par DataLab pour récupérer des métadonnées telles que le nom du plugin, le type et l'intégration dans le menu.

Note : Seuls les fichiers Python dont les noms commencent par `datalab_` seront analysés pour les plugins.

DataLab prend en charge trois catégories de plugins, chacune ayant son propre objectif et mécanisme d'enregistrement :

- **Plugins de traitement et de visualisation** Ajoutent des actions personnalisées pour le traitement des signaux ou des images. Cela peut inclure de nouvelles fonctions de calcul, des outils de visualisation de données ou des boîtes de dialogue interactives. Intégré dans un sous-menu dédié du menu « Plugins ».
- **Plugins d'entrée/sortie** Définissent de nouveaux formats de fichiers (lecture et/ou écriture) gérés de manière transparente par le framework d'entrée/sortie de DataLab. Ces plugins étendent la compatibilité avec des formats de données personnalisés ou tiers.
- **Plugins HDF5** Plugins spéciaux qui prennent en charge les fichiers HDF5 avec des structures d'arbre spécifiques au domaine. Ceux-ci permettent à DataLab d'interpréter des signaux ou des images organisés de manière non standard.

Où est positionné un plugin ?

Les plugins sont automatiquement découverts au démarrage à partir de plusieurs emplacements :

- Le répertoire des plugins utilisateur : Typiquement *~/.DataLab/plugins* sur Linux/macOS ou *C :/Users/YourName/.DataLab/plugins* sur Windows.
- Un répertoire de plugin personnalisé : Configurable dans les préférences de DataLab.
- Le répertoire de distribution autonome : Si vous utilisez une version gelée (autonome), le dossier *plugins* situé à côté de l'exécutable est analysé.
- Le dossier interne *datalab/plugins* (non recommandé pour les plugins utilisateur) : Cet emplacement est réservé aux plugins intégrés ou fournis et ne doit pas être modifié manuellement.

Comment développer un plugin ?

La méthode recommandée pour développer un plugin est de dériver d'un exemple existant et de l'adapter à vos besoins. Vous pouvez explorer le code source dans le dossier *datalab/plugins* ou vous référer aux exemples fournis par la communauté.

Note : La plupart des fonctionnalités de traitement des signaux et des images de DataLab ont été externalisées dans une bibliothèque dédiée appelée **Sigima** (https://sigima.readthedocs.io/fr/latest/). Lors du développement de plugins DataLab, vous importerez et utiliserez généralement de nombreuses fonctions et fonctionnalités de Sigima pour effectuer des tâches de traitement, d'analyse et de visualisation des données. Sigima fournit un ensemble complet d'outils pour la manipulation des données scientifiques qui peut être exploité directement dans vos plugins.

Pour développer dans votre environnement Python habituel (par exemple, avec un IDE comme Spyder), vous pouvez :

1. **Installer DataLab dans votre environnement Python**, en utilisant l'une des méthodes suivantes :

— *Paquet Conda*
— *Gestionnaire de paquets pip*
— *Paquet Wheel*
— *Paquet source*

2. **Ou ajoutez manuellement le package `datalab` à votre chemin Python** :
   — Téléchargez le code source depuis la page PyPI,
   — Décompressez l'archive,
   — Ajoutez le répertoire *datalab* à votre PYTHONPATH (par exemple, en utilisant le *Gestionnaire PYTHONPATH* dans Spyder).

Note : Même si vous avez installé *datalab* dans votre environnement, vous ne pouvez pas exécuter l'application DataLab complète directement depuis un IDE. Vous devez lancer DataLab via la ligne de commande ou en utilisant le raccourci créé par l'installateur pour tester correctement votre plugin.

Exemple : plugin de traitement

Voici un exemple minimal d'un plugin qui imprime un message lorsqu'il est activé :

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

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

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

from __future__ import annotations

import sigima.tests.data as test_data
from sigima.io.image import ImageIORegistry
from sigima.io.signal import SignalIORegistry
from sigima.tests import helpers

from datalab.config import _
from datalab.plugins import PluginBase, PluginInfo
from datalab.utils.qthelpers import create_progress_bar

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


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

    PLUGIN_INFO = PluginInfo(
        name=_("Test data"),
        version="1.0.0",
```

(suite sur la page suivante)

(suite de la page précédente)

```
        description=_("Testing DataLab functionalities"),
    )

    def load_test_objs(
        self, registry_class: type[SignalIORegistry | ImageIORegistry], title: str
    ) -> None:
        """Load all test objects from a given registry class

        Args:
            registry_class: Registry class (SignalIORegistry or ImageIORegistry)
            title: Progress bar title

        Returns:
            List of (filename, object) tuples
        """
        test_objs = list(helpers.read_test_objects(registry_class))
        with create_progress_bar(self.signalpanel, title, max_=len(test_objs)) as prog:
            for i_obj, (_fname, obj) in enumerate(test_objs):
                prog.setValue(i_obj + 1)
                if prog.wasCanceled():
                    break
                if obj is not None:
                    self.proxy.add_object(obj)

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

    # Image processing features -------------------------------------------------
    def create_peak_image(self) -> None:
        """Create 2D peak image"""
        obj = self.imagepanel.new_object(add_to_panel=False)
        if obj is not None:
            param = test_data.PeakDataParam.create(size=max(obj.data.shape))
            self.imagepanel.processor.update_param_defaults(param)
            if param.edit(self.main):
                obj.data, _coords = 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_type=True)
        if newparam is not None:
            obj = test_data.create_sincos_image(newparam)
            self.proxy.add_object(obj)

    def create_noisy_gaussian_image(self) -> None:
        """Create 2D noisy gauss image"""
        newparam = self.edit_new_image_parameters(hide_height=True, hide_type=True)
        if newparam is not None:
```

(suite sur la page suivante)

(suite de la page précédente)

```
            obj = test_data.create_noisy_gaussian_image(newparam, add_annotations=False)
            self.proxy.add_object(obj)

    def create_multigaussian_image(self) -> None:
        """Create 2D multi gauss image"""
        newparam = self.edit_new_image_parameters(hide_height=True, hide_type=True)
        if newparam is not None:
            obj = test_data.create_multigaussian_image(newparam)
            self.proxy.add_object(obj)

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

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

    def create_annotated_image(self) -> None:
        """Create annotated image"""
        obj = test_data.create_annotated_image()
        self.proxy.add_object(obj)

    def create_grid_gaussian_image(self) -> None:
        """Create image with a grid of gaussian spots"""
        param = test_data.GridOfGaussianImages(_("Grid of Gaussian Images"))
        if param.edit(self.main):
            obj = test_data.create_grid_of_gaussian_images(param)
            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(
                _("Load spectrum of paracetamol"),
                triggered=self.create_paracetamol_signal,
                select_condition="always",
            )
            sah.new_action(
                _("Load all test signals"),
                triggered=lambda regclass=SignalIORegistry,
                title=_("Load all test signals"): self.load_test_objs(regclass, title),
                select_condition="always",
```

(suite sur la page suivante)

(suite de la page précédente)

```
                separator=True,
            )
        # Image Panel ---------------------------------------------------------
        iah = self.imagepanel.acthandler
        with iah.new_menu(_("Test data")):
            iah.new_action(
                _("Create image with peaks"),
                triggered=self.create_peak_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 gaussian image"),
                triggered=self.create_noisy_gaussian_image,
                select_condition="always",
            )
            iah.new_action(
                _("Create 2D multi gaussian image"),
                triggered=self.create_multigaussian_image,
                select_condition="always",
            )
            iah.new_action(
                _("Create annotated image"),
                triggered=self.create_annotated_image,
                select_condition="always",
            )
            iah.new_action(
                _("Create 2D step image"),
                triggered=self.create_2dstep_image,
                select_condition="always",
            )
            iah.new_action(
                _("Create ring image"),
                triggered=self.create_ring_image,
                select_condition="always",
            )
            iah.new_action(
                _("Create image with a grid of gaussian spots"),
                triggered=self.create_grid_gaussian_image,
                select_condition="always",
            )
            iah.new_action(
                _("Load all test images"),
                triggered=lambda regclass=ImageIORegistry,
                title=_("Load all test images"): self.load_test_objs(regclass, title),
                select_condition="always",
                separator=True,
```

(suite sur la page suivante)

(suite de la page précédente)

```
            )
```

Exemple : plugin d'entrée/sortie

Voici un exemple simple d'un plugin qui ajoute de nouveaux formats de fichiers à DataLab.

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

"""
Image file formats Plugin for DataLab
-------------------------------------

This plugin is an example of DataLab plugin.
It provides image file formats from cameras, scanners, and other acquisition devices.
"""

import struct

import numpy as np
from sigima.io.base import FormatInfo
from sigima.io.image.base import SingleImageFormatBase

# ==============================================================================
# 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 = "<lllllffl"

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

(suite sur la page suivante)

(suite de la page précédente)

```
            self.load(fname)

    def __repr__(self) -> str:
        """Return a string representation of the object"""
        info = (
            ("Image width", f"{self.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,
                self.quantlevels,
                self.maxlevel,
                self.minlevel,
                self.comment_length,
            ) = unpacked_rec
            if self.__debug:
                print(unpacked_rec)
                print(self)
            data_file.seek(128 + self.comment_length)
            if self.pixeltype == 0:
                size, dtype = 4, np.float32
            elif self.pixeltype == 1:
                size, dtype = 2, np.uint16
```

(suite sur la page suivante)

(suite de la page précédente)

```
            elif self.pixeltype == 2:
                size, dtype = 1, np.uint8
            else:
                raise NotImplementedError(f"Unsupported pixel type: {self.pixeltype}")
            block = data_file.read(self.nbrows * self.nbcols * size)
        data = np.frombuffer(block, dtype=dtype)
        self.data = data.reshape(self.nbrows, self.nbcols)


class FXDImageFormat(SingleImageFormatBase):
    """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(SingleImageFormatBase):
    """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
```

(suite sur la page suivante)

(suite de la page précédente)

```
        Args:
            filename (str): path to XYZ file

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

    @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).

Migration de la v0.20 à la v1.0

Si vous avez des plugins existants écrits pour DataLab v0.20, veuillez consulter le *guide de migration* pour des instructions détaillées sur la mise à jour de vos plugins pour fonctionner avec DataLab v1.0.

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 datalab.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 unregister_all_plugins()

Désenregistrer tous les plugins

classmethod get_plugin_info(*html : bool = True*) → str

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

Paramètres

html – retourner du texte formaté en html (par défaut : True)

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

Informations sur le plugin

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

Métaclasse mixte pour éviter les conflits

class datalab.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*) → 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)

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_height : bool = False, hide_width : bool = False, hide_type : bool = True, hide_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_height** – masquer le paramètre de la hauteur de l'image (par défaut : False)
— **hide_width** – masquer le paramètre de la largeur de l'image (par défaut : False)
— **hide_type** – masquer le paramètre de type d'image (par défaut : True)
— **hide_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.DLMainWindow*) → 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

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

Découvrir les plugins en utilisant la convention de nommage

Renvoie
Liste des plugins découverts (en tant que classes)

datalab.plugins.**discover_v020_plugins**() → list[tuple[str, str]]

Découvrir les plugins v0.20 (avec le préfixe `cdl_`) sans les importer

Renvoie
Liste des tuples (nom_du_plugin, chemin_du_répertoire) pour les plugins v0.20 découverts

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

Instancier et obtenir les plugins disponibles

Renvoie
Liste des plugins disponibles (en tant qu'instances)

2.5.2 Migration de DataLab v0.20 vers v1.0

Avertissement : Note de compatibilité critique :

DataLab v1.0 introduit des **changements majeurs** qui ne sont **pas rétrocompatibles** avec v0.20.

- **Les plugins** développés pour v0.20 **ne fonctionneront pas** et doivent être mis à jour
- **Les changements d'API** nécessitent des modifications de code pour toutes les intégrations personnalisées

Veuillez lire ce guide attentivement avant de passer à la v1.0.

DataLab v1.0 introduit des changements architecturaux importants par rapport à la v0.20. Le changement le plus important est l'"**externalisation des fonctionnalités de traitement du signal et de l'image** dans une bibliothèque séparée appelée **Sigima**.

Ce changement architectural améliore la modularité, la testabilité et permet la réutilisation des fonctions de traitement dans d'autres projets. Cependant, cela nécessite des mises à jour des plugins existants pour s'adapter à la nouvelle API.

Ce guide décrit les étapes pour migrer les plugins de DataLab v0.20 vers v1.0.

- *Qu'est-ce qui a changé dans la v1.0 ?*
  - *Changements architecturaux majeurs*
  - *Renommage du package*
  - *Nouveaux objets résultat*
  - *Interface processeur unifiée*
- *Mise à jour des imports*
  - *Changements des chemins d'import*
- *Migration du code des plugins*
  - *Exemple 1 : Plugin de traitement simple*
  - *Exemple 2 : Plugin utilisant des fonctionnalités intégrées*
  - *Exemple 3 : Plugin créant des objets et gérant les résultats*
- *Travailler avec les objets résultat*
  - *Modifications des objets résultat*
  - *Utilisation de TableResult et GeometryResult*
  - *Récupération des résultats depuis les métadonnées*
- *Modifications des méthodes du processeur*
  - *Utilisation de run_feature()*
  - *Renommage des méthodes*
  - *Fonctionnalités intégrées utilisant run_feature*
- *Tester votre plugin*
- *Problèmes courants et solutions*
  - *Erreurs d'import*
  - *Erreurs d'attribut sur le processeur*
  - *Incompatibilité d'objet résultat*
  - *Paramètres manquants*
- *Ressources supplémentaires*
- *Obtenir de l'aide*

Qu'est-ce qui a changé dans la v1.0 ?

Changements architecturaux majeurs

Le changement le plus significatif dans DataLab v1.0 est la création de la **bibliothèque Sigima**, qui contient désormais :

- **Objets signal et image** (`SignalObj`, `ImageObj`, classes ROI)
- **Paramètres de traitement** (toutes les classes `*Param`)
- **Fonctions de traitement** (modules `sigima.proc.signal` et `sigima.proc.image`)
- **Algorithmes** pour le traitement du signal et de l'image, ou conversion de types de données, transformation de coordonnées, … (module `sigima.tools`)
- **Fonctions d'entrée/sortie** (lecture et écriture de signaux/images)
- **Utilitaires de données de test** (génération de signaux et d'images de test)
- **Objets résultat** (`TableResult` et `GeometryResult`, remplaçant `ResultProperties` et `ResultShape`)

Ce qui reste dans DataLab :

- **Composants d'interface graphique** (panneaux, widgets, intégration de tracés)
- **Système de plugins** (module `datalab.plugins`)
- **Logique applicative** (fenêtre principale, configuration, gestion de projet)
- **Adaptateurs de métadonnées de résultats** (`TableAdapter`, `GeometryAdapter` pour stocker les résultats dans les métadonnées des objets)

Renommage du package

Le package DataLab a été renommé de `cdl` à `datalab` par souci de clarté :

```
# v0.20
import cdl.obj
import cdl.param
from cdl.plugins import PluginBase

# v1.0
import sigima.objects
import sigima.params
from datalab.plugins import PluginBase
```

Nouveaux objets résultat

DataLab v1.0 introduit deux nouveaux types de résultats immuables :

- **TableResult** : Remplace `ResultProperties` pour les résultats scalaires tabulaires (par exemple, statistiques, mesures)
- **GeometryResult** : Remplace `ResultShape` pour les résultats géométriques (par exemple, caractéristiques détectées, contours)

Ces nouveaux types de résultats sont orientés calcul et exempts de logique spécifique à l'application (par exemple, Qt, métadonnées). Tous les comportements liés aux métadonnées ont été migrés vers la couche applicative de DataLab en utilisant des **classes d'adaptation** (`TableAdapter`, `GeometryAdapter`).

Interface processeur unifiée

Les méthodes du processeur ont été unifiées et renommées :

- La plupart des méthodes spécifiques `compute_*` utilisent désormais la méthode générique `run_feature()`
- Conventions de nommage mises à jour : `compute_11` → `compute_1_to_1`, `compute_10` → `compute_1_to_0`, etc.

Mise à jour des imports

Changements des chemins d'import

Le tableau suivant donne l'équivalence entre les imports DataLab v0.20 et v1.0.

Le tableau ci-dessous présente la correspondance entre l'API DataLab v0.20 et v1.0. Pour la plupart des entrées, seule l'instruction d'import doit être mise à jour.

Tableau 6: Tableau de compatibilité DataLab v0.20 vers v1.0

DataLab v0.20	DataLab v1.0
Renommages de modules/packages	
`cdl`	`datalab`
`cdl.obj`	`sigima.objects`
`cdl.param`	`sigima.params`
`cdl.computation.image`	`sigima.proc.image`
`cdl.computation.signal`	`sigima.proc.signal`
`cdl.plugins`	`datalab.plugins`
Création et manipulation d'objets	
`cdl.obj.create_image()`	`sigima.objects.create_image()`
`cdl.obj.create_signal()`	`sigima.objects.create_signal()`
`cdl.obj.ImageObj`	`sigima.objects.ImageObj`
`cdl.obj.SignalObj`	`sigima.objects.SignalObj`
`cdl.obj.create_image_roi()`	`sigima.objects.create_image_roi()`
`cdl.obj.create_signal_roi()`	`sigima.objects.create_signal_roi()`
`cdl.obj.ImageROI`	`sigima.objects.ImageROI`
`cdl.obj.SignalROI`	`sigima.objects.SignalROI`
Classes de paramètres	
`cdl.param.BinningParam`	`sigima.params.BinningParam`
`cdl.param.MovingMedianParam`	`sigima.params.MovingMedianParam`
`cdl.param.BlobOpenCVParam`	`sigima.params.BlobOpenCVParam`
`cdl.param.GaussianParam`	`sigima.params.GaussianParam`
`cdl.param.*Param`	`sigima.params.*Param`
Enveloppes de fonctions de calcul	
`cdl.computation.image.Wrap11Func`	`sigima.proc.image.Wrap1to1Func`
`cdl.computation.signal.Wrap11Func`	`sigima.proc.signal.Wrap1to1Func`
`cdl.computation.image.dst_11()`	`sigima.proc.image.dst_1to1()`
`cdl.computation.signal.dst_11()`	`sigima.proc.signal.dst_1to1()`
Méthodes du processeur (*p* est une instance de processeur de signal ou d'image)	
`p.compute_11(...)`	`p.compute_1_to_1(...)`
`p.compute_10(...)`	`p.compute_1_to_0(...)`
`p.compute_n1(...)`	`p.compute_n_to_1(...)`
`p.compute_binning(param)`	`p.run_feature("binning", param)`

suite sur la page suivante

Tableau 6 – suite de la page précédente

DataLab v0.20	DataLab v1.0
`p.compute_moving_median(param)`	`p.run_feature("moving_median", param)`
`p.compute_blob_opencv(param)`	`p.run_feature("blob_opencv", param)`
`p.compute_*(param)`	`p.run_feature("feature_name", param)`
Objets résultat	
`cdl.obj.ResultProperties`	`sigima.objects.TableResult`
`cdl.obj.ResultShape`	`sigima.objects.GeometryResult`
`result.add_to(obj)`	`TableAdapter(result).add_to(obj)`
`ResultShape.from_metadata_entry()`	`GeometryAdapter.from_metadata_entry()`
`result.to_html(obj)`	`ResultHtmlGenerator.generate_html(result, obj)`
Fonctions de données de test	
`cdl.tests.data.*`	`sigima.tests.data.*`
`cdl.tests.data.create_paracetamol_signal()`	`sigima.tests.data.create_paracetamol_signal()`
`cdl.tests.data.add_gaussian_noise_to_signal(`	`sigima.tests.data.add_gaussian_noise_to_signal()`
`cdl.tests.data.create_noisygauss_image()`	`sigima.tests.data.create_noisy_gaussian_image()`
`cdl.tests.data.create_multigauss_image()`	`sigima.tests.data.create_multigaussian_image()`
Nouveaux paramètres d'image	
`edit_new_image_parameters(hide_image_dtype=T`	`edit_new_image_parameters(hide_dtype=True)`
`edit_new_image_parameters(hide_image_type=Tı`	`edit_new_image_parameters(hide_type=True)`
`edit_new_image_parameters(hide_image_height=`	`edit_new_image_parameters(hide_height=True)`
Fonctions d'entrée/sortie	
`cdl.io.image.read_image()`	`sigima.io.image.read_image()`
`cdl.io.image.write_image()`	`sigima.io.image.write_image()`
`cdl.io.signal.read_signal()`	`sigima.io.signal.read_signal()`
`cdl.io.signal.write_signal()`	`sigima.io.signal.write_signal()`
Énumérations	
`cdl.obj.ImageDatatypes`	`sigima.objects.ImageDatatypes`
`cdl.obj.ImageTypes`	`sigima.objects.ImageTypes`
Fonctionnalités spécifiques à DataLab	
`cdl.config._`	`datalab.config._`
`cdl.plugins.PluginBase`	`datalab.plugins.PluginBase`
`cdl.plugins.PluginInfo`	`datalab.plugins.PluginInfo`

Migration du code des plugins

Exemple 1 : Plugin de traitement simple

Voici comment un plugin de filtre personnalisé simple doit être mis à jour :

DataLab v0.20 :

```
import numpy as np
import scipy.ndimage as spi

import cdl.computation.image as cpi
import cdl.obj
```

(suite sur la page suivante)

(suite de la page précédente)

```
import cdl.param
import cdl.plugins


def weighted_average_denoise(data: np.ndarray) -> np.ndarray:
    """Apply a custom denoising filter to an image."""
    def filter_func(values: np.ndarray) -> float:
        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):
            for name, func in (("Weighted average denoise", weighted_average_denoise),):
                # Wrap function to handle ImageObj objects
                wrapped_func = cpi.Wrap11Func(func)
                acth.new_action(
                    name, triggered=lambda: proc.compute_11(wrapped_func, title=name)
                )
```

DataLab v1.0 :

```
import numpy as np
import scipy.ndimage as spi
import sigima.proc.image as sipi

import datalab.plugins


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

(suite sur la page suivante)

(suite de la page précédente)

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

    PLUGIN_INFO = datalab.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):
            for name, func in (("Weighted average denoise", weighted_average_denoise),):
                # Wrap function to handle ImageObj objects
                wrapped_func = sipi.Wrap1to1Func(func)
                acth.new_action(
                    name,
                    triggered=lambda: proc.compute_1_to_1(wrapped_func, title=name),
                )
```

Changements clés :

1. `cdl.computation.image` → `sigima.proc.image`
2. `cdl.plugins` → `datalab.plugins`
3. `Wrap11Func` → `Wrap1to1Func`
4. `compute_11` → `compute_1_to_1`

Exemple 2 : Plugin utilisant des fonctionnalités intégrées

DataLab v0.20 :

```
import cdl.obj
import cdl.param
import cdl.plugins


class ExtractBlobs(cdl.plugins.PluginBase):
    """DataLab Example Plugin"""

    PLUGIN_INFO = cdl.plugins.PluginInfo(
        name="Extract blobs (example)",
        version="1.0.0",
        description="This is an example plugin",
    )

    def preprocess(self) -> None:
        """Preprocess image"""
        panel = self.imagepanel
```

(suite sur la page suivante)

(suite de la page précédente)

```
        param = cdl.param.BinningParam.create(sx=2, sy=2)
        panel.processor.compute_binning(param)
        panel.processor.compute_moving_median(cdl.param.MovingMedianParam.create(n=5))

    def detect_blobs(self) -> None:
        """Detect circular blobs"""
        panel = self.imagepanel
        param = cdl.param.BlobOpenCVParam()
        param.filter_by_color = False
        param.min_area = 600.0
        param.max_area = 6000.0
        param.filter_by_circularity = True
        param.min_circularity = 0.8
        param.max_circularity = 1.0
        panel.processor.compute_blob_opencv(param)

    def create_actions(self) -> None:
        """Create actions"""
        acth = self.imagepanel.acthandler
        with acth.new_menu(self.PLUGIN_INFO.name):
            acth.new_action("Preprocess image", triggered=self.preprocess)
            acth.new_action("Detect circular blobs", triggered=self.detect_blobs)
```

DataLab v1.0 :

```
import sigima.objects
import sigima.params

import datalab.plugins


class ExtractBlobs(datalab.plugins.PluginBase):
    """DataLab Example Plugin"""

    PLUGIN_INFO = datalab.plugins.PluginInfo(
        name="Extract blobs (example)",
        version="1.0.0",
        description="This is an example plugin",
    )

    def preprocess(self) -> None:
        """Preprocess image"""
        panel = self.imagepanel
        param = sigima.params.BinningParam.create(sx=2, sy=2)
        panel.processor.run_feature("binning", param)
        panel.processor.run_feature(
            "moving_median", sigima.params.MovingMedianParam.create(n=5)
        )

    def detect_blobs(self) -> None:
        """Detect circular blobs"""
        panel = self.imagepanel
```

(suite sur la page suivante)

(suite de la page précédente)

```
        param = sigima.params.BlobOpenCVParam()
        param.filter_by_color = False
        param.min_area = 600.0
        param.max_area = 6000.0
        param.filter_by_circularity = True
        param.min_circularity = 0.8
        param.max_circularity = 1.0
        panel.processor.run_feature("blob_opencv", param)

    def create_actions(self) -> None:
        """Create actions"""
        acth = self.imagepanel.acthandler
        with acth.new_menu(self.PLUGIN_INFO.name):
            acth.new_action("Preprocess image", triggered=self.preprocess)
            acth.new_action("Detect circular blobs", triggered=self.detect_blobs)
```

Changements clés :

1. `cdl.param` → `sigima.params`
2. `cdl.plugins` → `datalab.plugins`
3. `compute_binning(param)` → `run_feature("binning", param)`
4. `compute_moving_median(param)` → `run_feature("moving_median", param)`
5. `compute_blob_opencv(param)` → `run_feature("blob_opencv", param)`

Exemple 3 : Plugin créant des objets et gérant les résultats

DataLab v0.20 :

```
import numpy as np
import cdl.obj as dlo
import cdl.tests.data as test_data
from cdl.computation import image as cpima
from cdl.config import _
from cdl.plugins import PluginBase, PluginInfo


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"),
    )
```

(suite sur la page suivante)

(suite de la page précédente)

```
    def add_noise_to_image(self) -> None:
        """Add noise to image"""
        self.imagepanel.processor.compute_11(
            add_noise_to_image,
            paramclass=dlo.NormalRandomParam,
            title=_("Add noise"),
        )

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

    def create_actions(self) -> None:
        """Create actions"""
        iah = self.imagepanel.acthandler
        with iah.new_menu(_("Test data")):
            iah.new_action(_("Add noise to image"), triggered=self.add_noise_to_image)
            iah.new_action(
                _("Create 2D noisy gauss image"),
                triggered=self.create_noisygauss_image,
                select_condition="always",
            )
```

DataLab v1.0 :

```
import numpy as np
import sigima.objects
import sigima.tests.data as test_data
from sigima.proc import image as sipi
from datalab.config import _
from datalab.plugins import PluginBase, PluginInfo


def add_noise_to_image(
    src: sigima.objects.ImageObj, p: sigima.objects.NormalDistribution2DParam
) -> sigima.objects.ImageObj:
    """Add gaussian noise to image"""
    dst = sipi.dst_1to1(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(
```

(suite sur la page suivante)

(suite de la page précédente)

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

    def add_noise_to_image(self) -> None:
        """Add noise to image"""
        self.imagepanel.processor.compute_1_to_1(
            add_noise_to_image,
            paramclass=sigima.objects.NormalDistribution2DParam,
            title=_("Add noise"),
        )

    def create_noisy_gaussian_image(self) -> None:
        """Create 2D noisy gauss image"""
        newparam = self.edit_new_image_parameters(
            hide_height=True, hide_type=True
        )
        if newparam is not None:
            obj = test_data.create_noisy_gaussian_image(newparam, add_annotations=False)
            self.proxy.add_object(obj)

    def create_actions(self) -> None:
        """Create actions"""
        iah = self.imagepanel.acthandler
        with iah.new_menu(_("Test data")):
            iah.new_action(_("Add noise to image"), triggered=self.add_noise_to_image)
            iah.new_action(
                _("Create 2D noisy gaussian image"),
                triggered=self.create_noisy_gaussian_image,
                select_condition="always",
            )
```

Changements clés :

1. `cdl.obj` → `sigima.objects`
2. `cdl.tests.data` → `sigima.tests.data`
3. `cdl.computation.image` → `sigima.proc.image`
4. `dst_11` → `dst_1to1`
5. `compute_11` → `compute_1_to_1`
6. `dlo.NormalRandomParam` → `sigima.objects.NormalDistribution2DParam`
7. `create_noisygauss_image` → `create_noisy_gaussian_image`
8. `hide_image_height` → `hide_height`, `hide_image_type` → `hide_type`

Travailler avec les objets résultat

Modifications des objets résultat

L'architecture des objets résultat a été complètement repensée dans la v1.0 :

Objets résultat v0.20 :

- `ResultProperties` : Pour les résultats tabulaires (statistiques, etc.)
- `ResultShape` : Pour les résultats géométriques (caractéristiques détectées, etc.)
- Ces objets contenaient du code spécifique à Qt et de la logique de gestion des métadonnées
- Les méthodes comme `add_to(obj)` et `from_metadata_entry()` faisaient partie de la classe résultat

Objets résultat v1.0 :

- `TableResult` : Résultats tabulaires immuables, orientés calcul
- `GeometryResult` : Résultats géométriques immuables, orientés calcul
- Aucune dépendance Qt ou métadonnées
- La gestion des métadonnées a été déplacée vers les classes d'adaptation DataLab

Utilisation de TableResult et GeometryResult

Dans la v1.0, lorsque votre plugin reçoit des résultats d'opérations `compute_1_to_0`, vous travaillerez avec des objets `TableResult` ou `GeometryResult`.

Exemple 1 : Calcul des paramètres dynamiques du signal (TableResult)

Cet exemple montre comment calculer les paramètres dynamiques (ENOB, SINAD, THD, etc.) et travailler avec le `TableResult` résultant :

```
import sigima.params
import sigima.proc.signal
from datalab.adapters_metadata import TableAdapter

# Get the current signal from the panel
panel = self.signalpanel
obj = panel.objview.get_current_object()

# Compute dynamic parameters
param = sigima.params.DynamicParam.create(full_scale=1.0)
table = sigima.proc.signal.dynamic_parameters(obj, param)

# Access specific values from the table
enob = table["enob"][0]  # Effective Number of Bits
sinad = table["sinad"][0]  # Signal-to-Noise and Distortion Ratio
thd = table["thd"][0]  # Total Harmonic Distortion

# Convert to dictionary for easy access
result_dict = table.as_dict()
print(f"ENOB: {result_dict['enob']}")
print(f"SINAD: {result_dict['sinad']} dB")

# Store result in object metadata using adapter
adapter = TableAdapter(table)
adapter.add_to(obj)
```

(suite sur la page suivante)

(suite de la page précédente)

```
# Convert to pandas DataFrame for further processing
df = table.to_dataframe()
print(df)
```

Exemple 2 : Détection de pics dans une image (GeometryResult)

Cet exemple montre comment détecter des pics dans une image et travailler avec le `GeometryResult` résultant :

```
import sigima.params
import sigima.proc.image
from datalab.adapters_metadata import GeometryAdapter

# Get the current image from the panel
panel = self.imagepanel
obj = panel.objview.get_current_object()

# Configure peak detection parameters
param = sigima.params.Peak2DDetectionParam.create(
    size=50,  # Neighborhood size
    threshold=0.5,  # Relative threshold
    create_rois=True  # Create ROI for each peak
)

# Compute peak detection
geometry = sigima.proc.image.peak_detection(obj, param)

# Access geometry data
coords = geometry.coords  # numpy array of shape (n_peaks, 2)
n_peaks = len(coords)
print(f"Detected {n_peaks} peaks")

# Iterate over detected peaks
for i, (x, y) in enumerate(coords):
    print(f"Peak {i+1}: x={x:.2f}, y={y:.2f}")

# Check geometry kind
from sigima.objects import KindShape
assert geometry.kind == KindShape.POINT

# Store geometry in object metadata using adapter
adapter = GeometryAdapter(geometry)
adapter.add_to(obj)
```

Exemple 3 : Calcul de la bande passante (GeometryResult avec segments)

Cet exemple montre comment calculer la bande passante d'un signal, qui renvoie une géométrie de segment :

```
import sigima.proc.signal
from datalab.adapters_metadata import GeometryAdapter

# Get the current signal from the panel
panel = self.signalpanel
obj = panel.objview.get_current_object()
```

(suite sur la page suivante)

(suite de la page précédente)

```
# Compute -3dB bandwidth
geometry = sigima.proc.signal.bandwidth_3db(obj)

# Access segment coordinates [x0, y0, x1, y1]
x0, y0, x1, y1 = geometry.coords[0]
print(f"Bandwidth segment: ({x0}, {y0}) to ({x1}, {y1})")

# Calculate bandwidth length
bandwidth = geometry.segments_lengths()[0]
print(f"Bandwidth@-3dB: {bandwidth:.2f}")

# Store in metadata
adapter = GeometryAdapter(geometry)
adapter.add_to(obj)
```

Récupération des résultats depuis les métadonnées

Pour récupérer les résultats stockés dans les métadonnées de l'objet :

```
from datalab.adapters_metadata import TableAdapter, GeometryAdapter

obj = ...  # Some signal or image object

# Iterate over all table results
for adapter in TableAdapter.iterate_from_obj(obj):
    result = adapter.result  # TableResult instance
    title = adapter.title
    df = result.to_dataframe()

# Iterate over all geometry results
for adapter in GeometryAdapter.iterate_from_obj(obj):
    result = adapter.result  # GeometryResult instance
    title = adapter.title
    coords = result.coords
```

Modifications des méthodes du processeur

L'interface du processeur a été unifiée dans la v1.0. La plupart des méthodes spécifiques `compute_*` utilisent désormais la méthode générique `run_feature()`.

Utilisation de run_feature()

La méthode `run_feature()` est une interface unifiée pour exécuter des fonctionnalités de traitement :

```
# v1.0 - Generic interface
proc = panel.processor

# Simple features (no parameters)
proc.run_feature("normalize")
proc.run_feature("fft")

# Features with parameters
param = sigima.params.GaussianParam.create(sigma=2.0)
proc.run_feature("gaussian_filter", param)

# Features with direct function reference
import sigima.proc.image as sipi
proc.run_feature(sipi.normalize)
proc.run_feature(sipi.gaussian_filter, param)
```

Renommage des méthodes

Plusieurs méthodes du processeur ont été renommées par souci de cohérence :

v0.20	v1.0
`compute_11(func, ...)`	`compute_1_to_1(func, ...)`
`compute_10(func, ...)`	`compute_1_to_0(func, ...)`
`compute_n1(func, ...)`	`compute_n_to_1(func, ...)`
`compute_1n(func, ...)`	`compute_1_to_n(func, ...)`
`compute_21(func, ...)`	`compute_2_to_1(func, ...)`

Fonctionnalités intégrées utilisant run_feature

La plupart des fonctionnalités de traitement intégrées qui avaient auparavant des méthodes dédiées `compute_*` utilisent désormais `run_feature()` :

```
# v0.20
proc.compute_binning(param)
proc.compute_moving_median(param)
proc.compute_blob_opencv(param)
proc.compute_gaussian_filter(param)

# v1.0
proc.run_feature("binning", param)
```

(suite sur la page suivante)

(suite de la page précédente)

```
proc.run_feature("moving_median", param)
proc.run_feature("blob_opencv", param)
proc.run_feature("gaussian_filter", param)
```

Certaines fonctionnalités complexes ont toujours des méthodes dédiées :

```
# These still use dedicated methods in v1.0
proc.compute_roi_extraction(roi)
proc.compute_multigaussianfit()
proc.compute_peak_detection(param)
```

Tester votre plugin

Après la migration, testez minutieusement votre plugin :

1. **Vérification des imports** : Vérifiez que tous les imports fonctionnent correctement
2. **Création de paramètres** : Vérifiez que les classes de paramètres sont correctement instanciées
3. **Création d'objets** : Testez les fonctions de création d'objets signal/image
4. **Opérations de traitement** : Vérifiez que toutes les fonctionnalités de traitement fonctionnent comme prévu
5. **Gestion des résultats** : Vérifiez que les résultats sont correctement générés et stockés
6. **Intégration de l'interface graphique** : Testez que les menus et actions apparaissent correctement

Problèmes courants et solutions

Erreurs d'import

Problème : `ModuleNotFoundError: No module named 'cdl'`

Solution : Mettez à jour tous les imports de `cdl.*` vers soit `datalab.*` soit `sigima.*` selon le module.

Erreurs d'attribut sur le processeur

Problème : `AttributeError: 'ImageProcessor' object has no attribute 'compute_binning'`

Solution : Remplacez les appels de méthode `compute_*` par des appels `run_feature()` :

```
# Wrong (v0.20 style)
proc.compute_binning(param)

# Correct (v1.0 style)
proc.run_feature("binning", param)
```

Incompatibilité d'objet résultat

Problème : `AttributeError: 'TableResult' object has no attribute 'add_to'`

Solution : Utilisez des classes d'adaptation pour les opérations de métadonnées :

```
# Wrong (v0.20 style)
result.add_to(obj)

# Correct (v1.0 style)
from datalab.adapters_metadata import TableAdapter
TableAdapter(result).add_to(obj)
```

Paramètres manquants

Problème : `ImportError: cannot import name 'SomeParam' from 'datalab'`

Solution : Importez les paramètres depuis `sigima.params` au lieu de `cdl.param` :

```
# Wrong
from cdl.param import GaussianParam

# Correct
from sigima.params import GaussianParam
```

Ressources supplémentaires

- Documentation DataLab v1.0
- Documentation Sigima
- Exemples de plugins
- *Référence de l'API DataLab*
- Référence de l'API Sigima

Obtenir de l'aide

Si vous rencontrez des problèmes pendant la migration :

1. Consultez le suivi des problèmes GitHub
2. Consultez les exemples de plugins intégrés dans le répertoire `datalab/plugins`

2.5.3 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.

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.

Fig. 70 – Le Gestonnaire de Macros dans DataLab.

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*.

2.5.4 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 à distance DataLab (c'est-à-dire sans avoir à installer l'ensemble du package DataLab et ses dépendances sur votre environnement), vous pouvez utiliser le package Sigima qui fournit un client distant simple pour DataLab. Pour l'installer, il suffit d'exécuter : *pip install sigima*.

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 datalab.control.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 : `datalab.tests.features.control.remoteclient_app_test`
- Voir le module : `datalab.tests.features.control.remoteclient_unit`

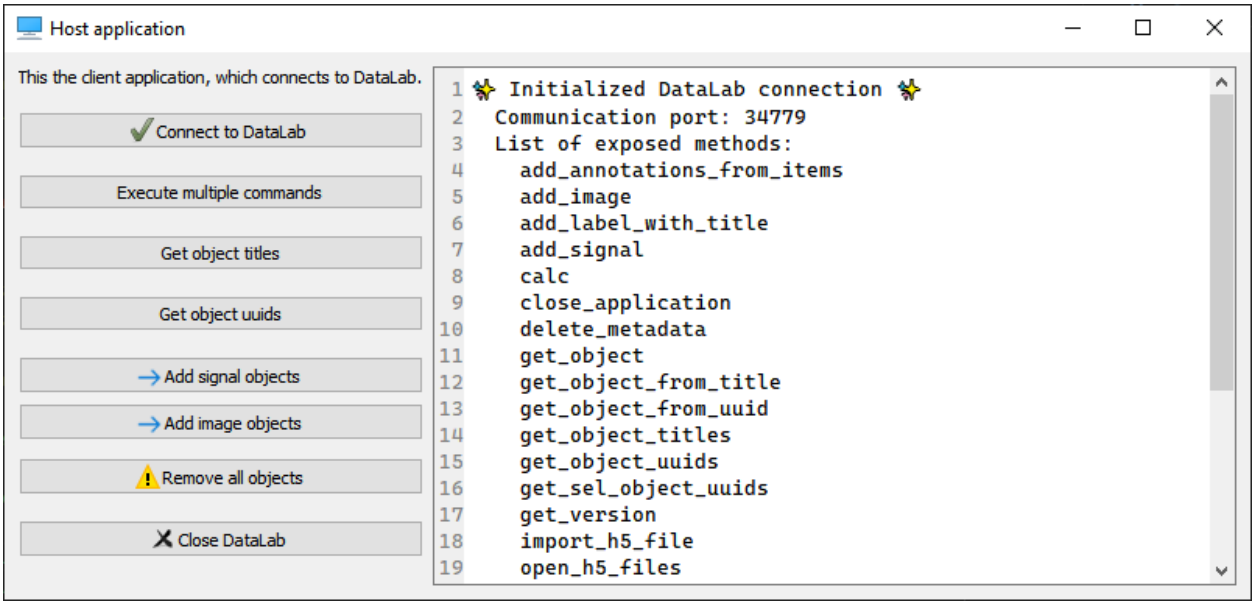


FIG. 71 – Capture d'écran du test de l'application cliente distante (`datalab.tests.features.control.remoteclient_app_test`)

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) :

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

Created on Fri May 12 12:28:56 2023

@author: p.raybaut
"""

# %% Importing necessary modules

# NumPy for numerical array computations:
import numpy as np

# DataLab remote control client:
from sigima.client import SimpleRemoteProxy as RemoteProxy

# %% Connecting to DataLab current session

proxy = RemoteProxy()
```

(suite sur la page suivante)

(suite de la page précédente)

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

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

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

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

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

proxy.calc("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.calc("fft")
```

Voici une réimplémentation de cette classe en Python 2.7 :

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

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

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

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


def array_to_rpcbinary(data):
    """Convert NumPy array to XML-RPC Binary object, with shape and dtype"""
    dbytes = io.BytesIO()
```

(suite sur la page suivante)

(suite de la page précédente)

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


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


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

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

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

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

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

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

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

(suite sur la page suivante)

(suite de la page précédente)

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

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

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

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

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

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

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

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

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

(suite sur la page suivante)

(suite de la page précédente)

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

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

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

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


def test_remote_client():
    """DataLab Remote Client test"""
    datalab = RemoteClient()
    datalab.connect()
    data = np.array([[3, 4, 5], [7, 8, 0]], dtype=np.uint16)
    datalab.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 `datalab.widgets.connection.ConnectionDialog`.

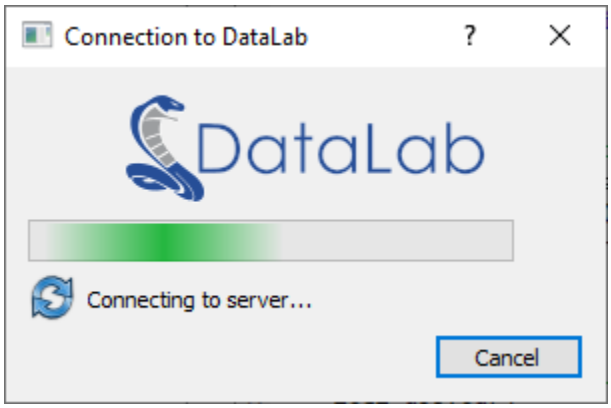


Fig. 72 – Capture d'écran de la boîte de dialogue de connexion (`datalab.widgets.connection.ConnectionDialog`)

Exemple d'utilisation :

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

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

# guitest: show,skip

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

from datalab.control.proxy import RemoteProxy
from datalab.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

class datalab.control.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 datalab.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"]
<sigima.objects.signal.SignalObj at 0x7f7f1c0b4a90>
>>> proxy[1]
<sigima.objects.signal.SignalObj at 0x7f7f1c0b4a90>
>>> proxy[1].data
array([1., 2., 3.])
```

set_port(*port : str | None = None*) → None

Set XML-RPC port to connect to.

Paramètres

port – XML-RPC port to connect to. If None, the port is automatically retrieved from DataLab configuration.

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

Try to connect to DataLab XML-RPC server.

Paramètres

- **port** – XML-RPC port to connect to. If not specified, the port is automatically retrieved from DataLab configuration.
- **timeout** – Maximum time to wait for connection in seconds. Defaults to 5.0. This is the total maximum wait time, not per retry.
- **retries** – Number of retries. Defaults to 10. This parameter is deprecated and will be removed in a future version (kept for backward compatibility).

Lève

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

disconnect() → None

Disconnect from DataLab XML-RPC server.

is_connected() → bool

Return True if connected to DataLab XML-RPC server.

get_method_list() → list[str]

Return list of available methods.

add_signal(*title : str*, *xdata : ndarray*, *ydata : ndarray*, *xunit : str = ''*, *yunit : str = ''*, *xlabel : str = ''*, *ylabel : str = ''*, *group_id : str = ''*, *set_current : bool = True*) → bool

Add signal data to DataLab.

Paramètres

- **title** – Signal title
- **xdata** – X data
- **ydata** – Y data
- **xunit** – X unit. Defaults to « «
- **yunit** – Y unit. Defaults to « «
- **xlabel** – X label. Defaults to « «
- **ylabel** – Y label. Defaults to « «
- **group_id** – group id in which to add the signal. Defaults to « «
- **set_current** – if True, set the added signal as current

Renvoie

True if signal was added successfully, False otherwise

Lève

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

add_image(*title : str*, *data : ndarray*, *xunit : str = ''*, *yunit : str = ''*, *zunit : str = ''*, *xlabel : str = ''*, *ylabel : str = ''*, *zlabel : str = ''*, *group_id : str = ''*, *set_current : bool = True*) → bool

Add image data to DataLab.

Paramètres

- **title** – Image title
- **data** – Image data
- **xunit** – X unit. Defaults to « «
- **yunit** – Y unit. Defaults to « «
- **zunit** – Z unit. Defaults to « «
- **xlabel** – X label. Defaults to « «
- **ylabel** – Y label. Defaults to « «
- **zlabel** – Z label. Defaults to « «
- **group_id** – group id in which to add the image. Defaults to « «
- **set_current** – if True, set the added image as current

Renvoie

True if image was added successfully, False otherwise

Lève

ValueError – Invalid data dtype

add_object(*obj : SignalObj | ImageObj*, *group_id : str = ''*, *set_current : bool = True*) → None

Add object to DataLab.

Paramètres

- **obj** – Signal or image object
- **group_id** – group id in which to add the object. Defaults to « «
- **set_current** – if True, set the added object as current

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

Call computation feature `name`

Note : This calls either the processor's `compute_<name>` method (if it exists), or the processor's `<name>` computation feature (if it is registered, using the `run_feature` method). It looks for the function in all

panels, starting with the current one.

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

Lève
ValueError – unknown function

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

Get object (signal/image) from index.

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

Renvoie
Object

Lève
KeyError – if object not found

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

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

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

Renvoie
List of plot item shapes

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

Add object annotations (annotation plot items).

Paramètres
- **items** – annotation plot items
- **refresh_plot** – refresh plot. Defaults to True.
- **panel** – panel name (valid values : « signal », « image »). If None, current panel is used.

add_group(*title : str, panel : str | None = None, select : bool = False*) → None

Add group to DataLab.

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

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

Add a label with object title on the associated plot

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

close_application() → None

Close DataLab application

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

Return a context manager to temporarily disable auto refresh.

Renvoie
Context manager

Exemple

```
>>> with proxy.context_no_refresh():
...     proxy.add_image("image1", data1)
...     proxy.calc("fft")
...     proxy.calc("wiener")
...     proxy.calc("ifft")
...     # Auto refresh is disabled during the above operations
```

delete_metadata(*refresh_plot : bool = True, keep_roi : bool = False*) → None

Delete metadata of selected objects

Paramètres
— **refresh_plot** – Refresh plot. Defaults to True.
— **keep_roi** – Keep ROI. Defaults to False.

get_current_panel() → str

Return current panel name.

Renvoie
« signal », « image », « macro »))

Type renvoyé
Panel name (valid values

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

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

Renvoie
groups titles, lists of inner objects uuids and titles

Type renvoyé
Tuple

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

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

Paramètres
panel – panel name (valid values : « signal », « image », « macro »). If None, current data panel is used (i.e. signal or image panel).

Renvoie
List of object titles

Lève
ValueError – if panel not found

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

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

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

Renvoie
List of object uuids

Lève
`ValueError` – if panel not found

`classmethod get_public_methods()` → list[str]

Return all public methods of the class, except itself.

Renvoie
List of public methods

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

Return selected objects uuids.

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

Renvoie
List of selected objects uuids.

`get_version()` → str

Return DataLab public version.

Renvoie
DataLab version

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

Open DataLab HDF5 browser to Import HDF5 file.

Paramètres
— `filename` – HDF5 file name
— `reset_all` – Reset all application data. Defaults to None.

`import_macro_from_file(filename : str)` → None

Import macro from file

Paramètres
`filename` – Filename.

`load_from_directory(path : str)` → None

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

Paramètres
`path` – directory path

`load_from_files(filenames : list[str])` → None

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

Paramètres
`filenames` – list of file names

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

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

Paramètres
— `h5files` – List of HDF5 files to open. Defaults to None.
— `import_all` – Import all objects from HDF5 files. Defaults to None.
— `reset_all` – Reset all application data. Defaults to None.

`raise_window()` → None

Raise DataLab window

`reset_all()` → None

Reset all application data

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

Run macro.

Paramètres
number_or_title – Macro number, or macro title. Defaults to None (current macro).

Lève
ValueError – if macro not found

save_to_h5_file(*filename : str*) → None

Save to a DataLab HDF5 file.

Paramètres
filename – HDF5 file name

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

Select groups in current panel.

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

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

Select objects in current panel.

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

set_current_panel(*panel : str*) → None

Switch to panel.

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

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

Stop macro.

Paramètres
number_or_title – Macro number, or macro title. Defaults to None (current macro).

Lève
ValueError – if macro not found

toggle_auto_refresh(*state : bool*) → None

Toggle auto refresh state.

Paramètres
state – Auto refresh state

toggle_show_titles(*state : bool*) → None

Toggle show titles state.

Paramètres
state – Show titles state

2.5.5 Modèle de données interne

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

- `sigima.objects.SignalObj`, qui représente un objet signal, et
- `sigima.objects.ImageObj`, qui représente un objet image.

Ces classes sont définies dans le paquet `sigima.objects`.

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 `sigima.params`.

Voir aussi :

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

2.5.6 Journaux de bord

Malgré des efforts considérables en matière de tests (tests unitaires, couverture de tests…), 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)

Note : Le visualiseur de journaux de bord est accessible depuis le menu « ? » dans la fenêtre principale.

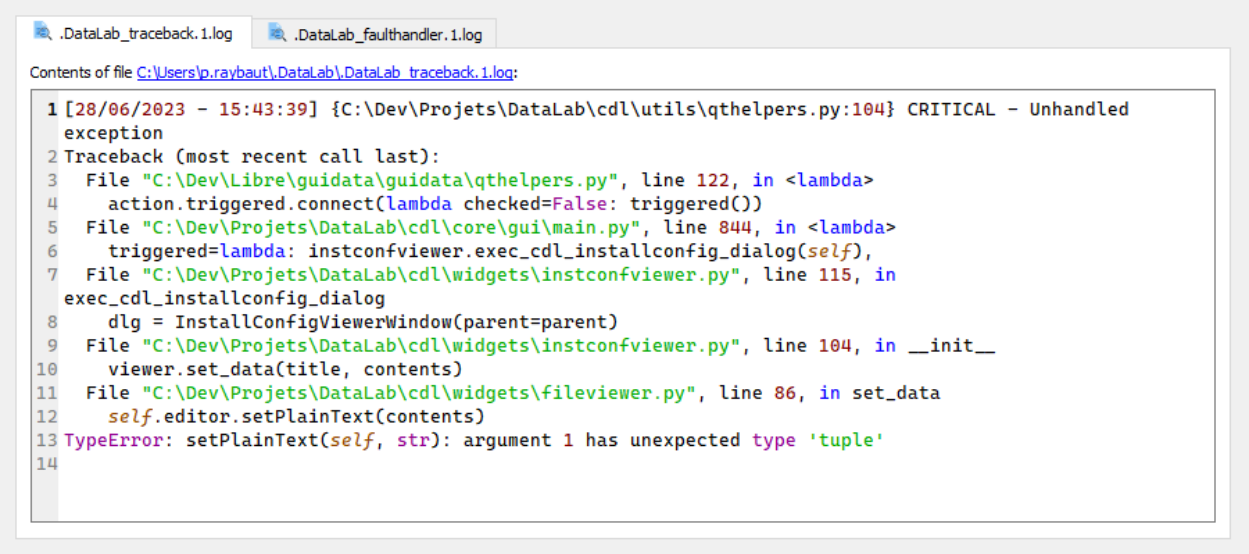


Fig. 73 – Journaux de bord DataLab (voir le menu « ? »)

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*).

2.5.7 Installation et configuration

La boîte de dialogue **Installation et configuration** est un outil de diagnostic disponible dans l'interface graphique de DataLab. Elle fournit un aperçu structuré de l'installation actuelle, des paramètres spécifiques à l'utilisateur et des plugins et fonctionnalités d'E/S disponibles.

Cet outil est principalement conçu pour le dépannage et la vérification — que vous soyez un utilisateur essayant de comprendre comment DataLab est configuré sur votre système, ou un développeur validant l'intégration de votre plugin.

Pour ouvrir la boîte de dialogue, allez dans le menu « ? » dans la fenêtre principale de DataLab et sélectionnez **Installation et configuration**. Cela ouvrira une boîte de dialogue qui affiche toutes les informations pertinentes sur votre installation de DataLab.

La fenêtre de dialogue est divisée en **trois onglets**, chacun couvrant un aspect spécifique de l'environnement.

1. Installation et configuration

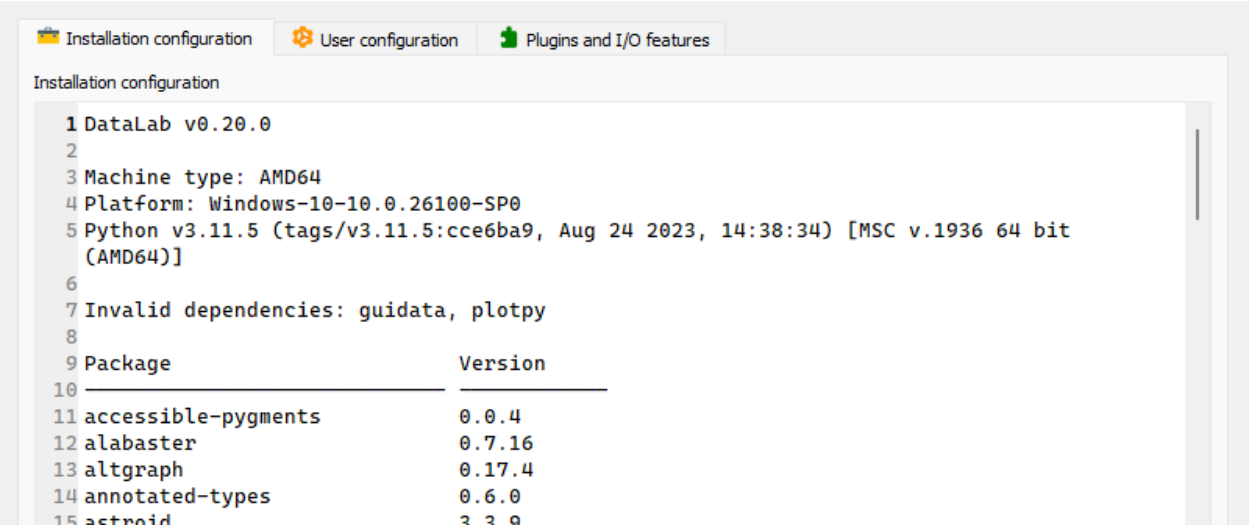


Fig. 74 – Installation et configuration (voir menu « ? »), onglet 1

Cet onglet affiche des informations au niveau système sur l'installation de DataLab, y compris :

- Version de l'application ;
- Système d'exploitation et plateforme ;
- Si DataLab fonctionne en mode gelé (autonome) ou en tant que package Python standard ;
- Chemins internes utilisés pour le stockage de données et la découverte de plugins ;
- Détails de l'interpréteur Python (version, architecture, chemins) ;
- Emplacement du package *datalab*.

Cette section est particulièrement utile pour déboguer les problèmes liés aux chemins ou confirmer le contexte d'installation (par exemple, autonome ou installé via pip).

2. Configuration utilisateur

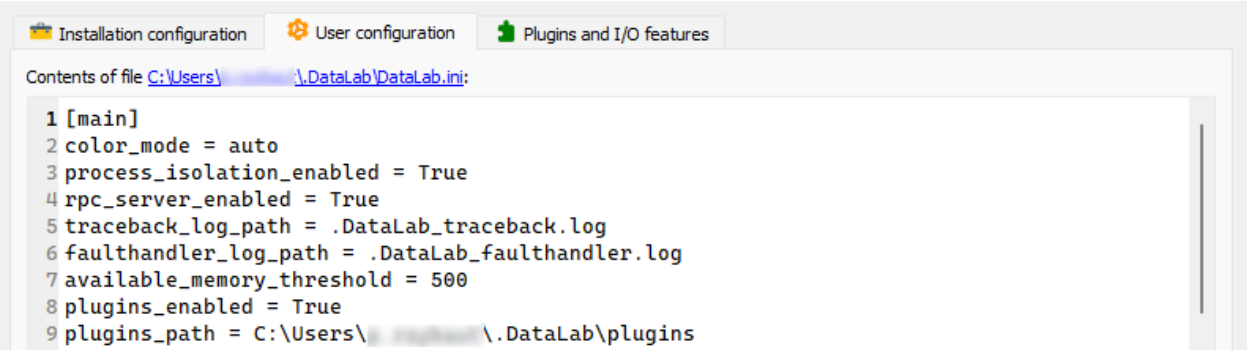


FIG. 75 – Installation et configuration (voir menu « ? »), onglet 2

Cet onglet répertorie les détails de configuration spécifiques à l'utilisateur, à travers le contenu du fichier de configuration de DataLab.

Il aide les utilisateurs à vérifier leurs paramètres personnalisés et garantit que DataLab lit les bons répertoires pour les plugins ou préférences définis par l'utilisateur.

3. Plugins et fonctionnalités d'E/S

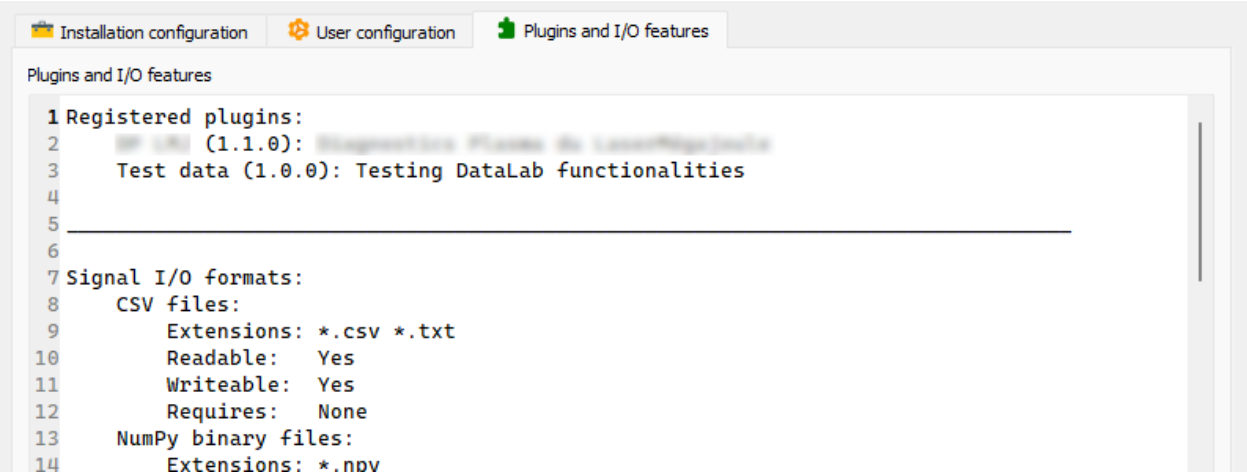


FIG. 76 – Installation et configuration (voir menu « ? »), onglet 3

Cet onglet répertorie tous les plugins et gestionnaires d'E/S actuellement disponibles dans l'application.

Cette vue est idéale pour vérifier que tous les plugins et fonctionnalités d'E/S attendus sont détectés, chargés et fonctionnels.

Conseils d'utilisation

- Vous pouvez copier du texte depuis n'importe quel onglet à des fins de diagnostic ou de support.
- Si un plugin ou une fonctionnalité d'E/S n'apparaît pas, vérifiez que son nom de fichier commence par *datalab_* et qu'il est situé dans un répertoire de plugins reconnu (voir l'onglet *Configuration utilisateur*).

Note : Signaler un comportement inattendu ou tout autre bug sur GitHub Issues sera grandement apprécié, surtout si le contenu de ce visualiseur est joint au rapport (ainsi que les fichiers journaux, voir *Journaux de bord*).

2.5.8 Fonctionnalités de la ligne de commande

Exécuter DataLab

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

```
$ datalab
```

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

```
$ datalab --help
usage: app.py [-h] [-b path] [-v] [--unattended] [--screenshot] [--delay DELAY] [--
↪xmlrpcport PORT]
              [--verbose {quiet,minimal,normal}]
              [h5]

Run DataLab

positional arguments:
h5                    HDF5 file names (separated by ';'), optionally with dataset name␣
↪(separated by ',')

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

Ouvrir un fichier HDF5 au démarrage

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

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

Ouvrir l'explorateur de fichiers HDF5 au démarrage

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

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

Mode démonstration de DataLab

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

```
$ datalab-demo
```

Exécuter les tests de validation technique

Note : Les tests de validation technique sont directement inclus dans les tests unitaires individuels et sont disséminés dans tout le code. Les fonctions de test incluant des tests de validation sont marquées avec le décorateur `@pytest.mark.validation`.

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

```
$ pytest -m validation
```

Voir aussi :

Voir la section *Vue d'ensemble et fonctionnalités communes* pour plus d'informations sur la stratégie de validation de DataLab.

Exécuter l'ensemble des tests

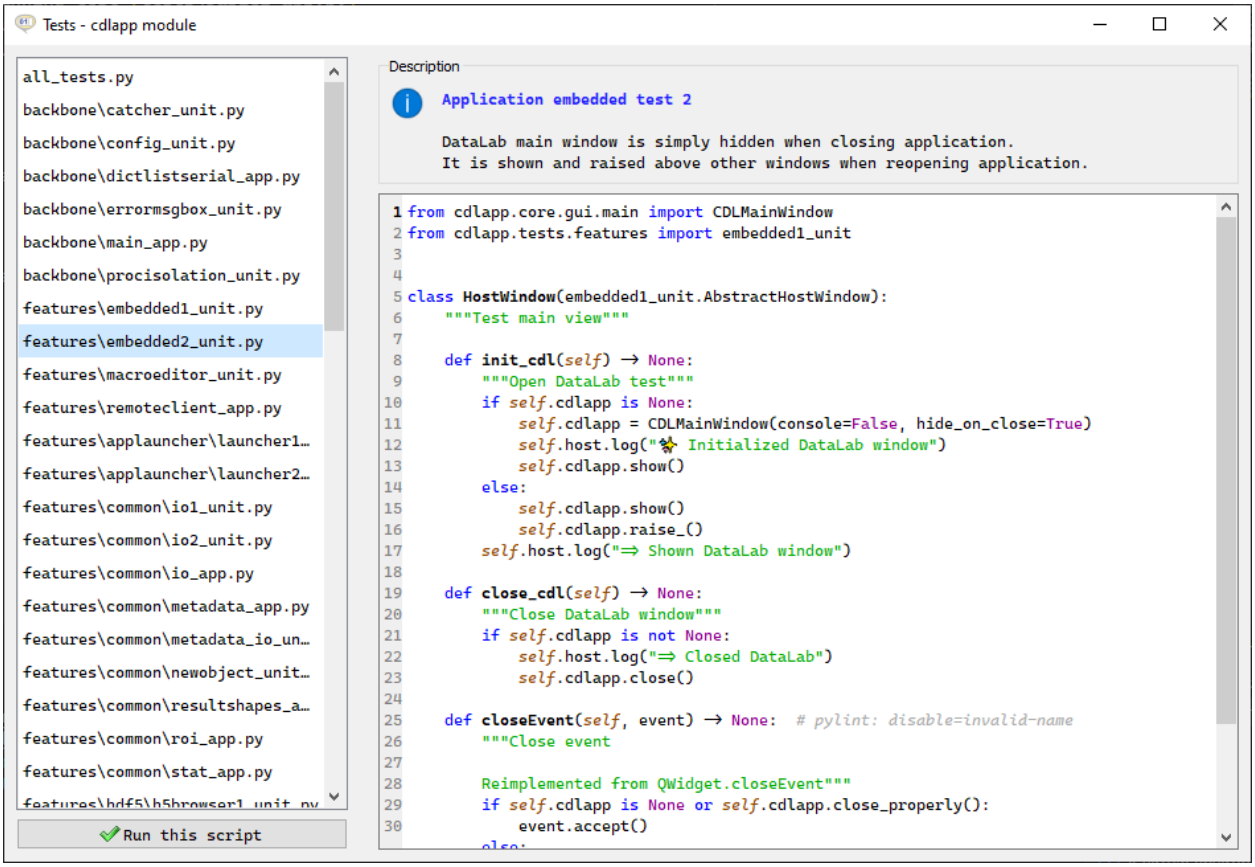
Pour exécuter l'ensemble des tests unitaires de DataLab, taper la commande suivante :

```
$ pytest
```

Exécuter les tests interactifs

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

```
$ datalab-tests
```



2.5.9 API

L'interface de programmation d'application (API) publique de DataLab offre un ensemble de fonctions pour accéder aux fonctionnalités de DataLab.

Note : Pour plus de détails sur le package *sigima*, veuillez vous référer à la documentation de Sigima. L'API de Sigima est particulièrement utile pour écrire des scripts pouvant être exécutés dans DataLab (voir *Macros*) ou pour développer des plugins (voir *Plugins*).

Proxy objects (datalab.control.proxy)

The *datalab.control.proxy* module provides a way to access DataLab features from a proxy class.

Remote proxy

The remote proxy is used when DataLab is started from a different process than the proxy. In this case, the proxy connects to DataLab XML-RPC server.

class datalab.control.proxy.**RemoteProxy**(*autoconnect : bool = True*)

DataLab remote proxy class.

This class provides access to DataLab features from a proxy class. This is the remote version of proxy, which is used when DataLab is started from a different process than the proxy.

Paramètres

autoconnect – Automatically connect to DataLab XML-RPC server.

Lève

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

Exemples

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

```
>>> from datalab.control.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
<sigima.objects.signal.SignalObj at 0x7f7f1c0b4a90>
>>> proxy[1]  # from number
<sigima.objects.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

Add object annotations (annotation plot items).

Paramètres

— **items** – annotation plot items
— **refresh_plot** – refresh plot. Defaults to True.
— **panel** – panel name (valid values : « signal », « image »). If None, current panel is used.

add_group(*title : str, panel : str | None = None, select : bool = False*) → None

Add group to DataLab.

Paramètres

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

add_image(*title : str*, *data : ndarray*, *xunit : str = ''*, *yunit : str = ''*, *zunit : str = ''*, *xlabel : str = ''*, *ylabel : str = ''*, *zlabel : str = ''*, *group_id : str = ''*, *set_current : bool = True*) → bool

Add image data to DataLab.

Paramètres
— **title** – Image title
— **data** – Image data
— **xunit** – X unit. Defaults to « «
— **yunit** – Y unit. Defaults to « «
— **zunit** – Z unit. Defaults to « «
— **xlabel** – X label. Defaults to « «
— **ylabel** – Y label. Defaults to « «
— **zlabel** – Z label. Defaults to « «
— **group_id** – group id in which to add the image. Defaults to « «
— **set_current** – if True, set the added image as current

Renvoie
True if image was added successfully, False otherwise

Lève
ValueError – Invalid data dtype

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

Add a label with object title on the associated plot

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

add_object(*obj : SignalObj | ImageObj*, *group_id : str = ''*, *set_current : bool = True*) → None

Add object to DataLab.

Paramètres
— **obj** – Signal or image object
— **group_id** – group id in which to add the object. Defaults to « «
— **set_current** – if True, set the added object as current

add_signal(*title : str*, *xdata : ndarray*, *ydata : ndarray*, *xunit : str = ''*, *yunit : str = ''*, *xlabel : str = ''*, *ylabel : str = ''*, *group_id : str = ''*, *set_current : bool = True*) → bool

Add signal data to DataLab.

Paramètres
— **title** – Signal title
— **xdata** – X data
— **ydata** – Y data
— **xunit** – X unit. Defaults to « «
— **yunit** – Y unit. Defaults to « «
— **xlabel** – X label. Defaults to « «
— **ylabel** – Y label. Defaults to « «
— **group_id** – group id in which to add the signal. Defaults to « «
— **set_current** – if True, set the added signal as current

Renvoie
True if signal was added successfully, False otherwise

Lève
— **ValueError** – Invalid xdata dtype

— **ValueError** – Invalid ydata dtype

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

Call computation feature `name`

Note : This calls either the processor's `compute_<name>` method (if it exists), or the processor's `<name>` computation feature (if it is registered, using the `run_feature` method). It looks for the function in all panels, starting with the current one.

Paramètres

— **name** – Compute function name

— **param** – Compute function parameter. Defaults to None.

Lève

ValueError – unknown function

close_application() → None

Close DataLab application

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

Try to connect to DataLab XML-RPC server.

Paramètres

— **port** – XML-RPC port to connect to. If not specified, the port is automatically retrieved from DataLab configuration.

— **timeout** – Maximum time to wait for connection in seconds. Defaults to 5.0. This is the total maximum wait time, not per retry.

— **retries** – Number of retries. Defaults to 10. This parameter is deprecated and will be removed in a future version (kept for backward compatibility).

Lève

— **ConnectionRefusedError** – Unable to connect to DataLab

— **ValueError** – Invalid timeout (must be >= 0.0)

— **ValueError** – Invalid number of retries (must be >= 1)

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

Return a context manager to temporarily disable auto refresh.

Renvoie

Context manager

Exemple

```
>>> with proxy.context_no_refresh():
...     proxy.add_image("image1", data1)
...     proxy.calc("fft")
...     proxy.calc("wiener")
...     proxy.calc("ifft")
...     # Auto refresh is disabled during the above operations
```

delete_metadata(*refresh_plot : bool = True*, *keep_roi : bool = False*) → None

Delete metadata of selected objects

Paramètres

— **refresh_plot** – Refresh plot. Defaults to True.

— **keep_roi** – Keep ROI. Defaults to False.

disconnect() → None

Disconnect from DataLab XML-RPC server.

get_current_panel() → str

Return current panel name.

Renvoie
« signal », « image », « macro »))

Type renvoyé
Panel name (valid values

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

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

Renvoie
groups titles, lists of inner objects uuids and titles

Type renvoyé
Tuple

get_method_list() → list[str]

Return list of available methods.

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

Get object (signal/image) from index.

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

Renvoie
Object

Lève
KeyError – if object not found

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

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

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

Renvoie
List of plot item shapes

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

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

Paramètres
panel – panel name (valid values : « signal », « image », « macro »). If None, current data panel is used (i.e. signal or image panel).

Renvoie
List of object titles

Lève
ValueError – if panel not found

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

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

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

Renvoie
List of object uuids

Lève
ValueError – if panel not found

classmethod get_public_methods() → list[str]

Return all public methods of the class, except itself.

Renvoie
List of public methods

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

Return selected objects uuids.

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

Renvoie
List of selected objects uuids.

get_version() → str

Return DataLab public version.

Renvoie
DataLab version

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

Open DataLab HDF5 browser to Import HDF5 file.

Paramètres
— **filename** – HDF5 file name
— **reset_all** – Reset all application data. Defaults to None.

import_macro_from_file(*filename : str*) → None

Import macro from file

Paramètres
filename – Filename.

is_connected() → bool

Return True if connected to DataLab XML-RPC server.

load_from_directory(*path : str*) → None

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

Paramètres
path – directory path

load_from_files(*filenames : list[str]*) → None

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

Paramètres
filenames – list of file names

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

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

Paramètres
— **h5files** – List of HDF5 files to open. Defaults to None.

— **import_all** – Import all objects from HDF5 files. Defaults to None.
— **reset_all** – Reset all application data. Defaults to None.

raise_window() → None

Raise DataLab window

reset_all() → None

Reset all application data

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

Run macro.

Paramètres
number_or_title – Macro number, or macro title. Defaults to None (current macro).

Lève
ValueError – if macro not found

save_to_h5_file(*filename : str*) → None

Save to a DataLab HDF5 file.

Paramètres
filename – HDF5 file name

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

Select groups in current panel.

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

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

Select objects in current panel.

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

set_current_panel(*panel : str*) → None

Switch to panel.

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

set_port(*port : str | None = None*) → None

Set XML-RPC port to connect to.

Paramètres
port – XML-RPC port to connect to. If None, the port is automatically retrieved from DataLab configuration.

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

Stop macro.

Paramètres
number_or_title – Macro number, or macro title. Defaults to None (current macro).

Lève
ValueError – if macro not found

toggle_auto_refresh(*state : bool*) → None

Toggle auto refresh state.

Paramètres
state – Auto refresh state

toggle_show_titles(*state : bool*) → None

Toggle show titles state.

Paramètres
state – Show titles state

Local proxy

The local proxy is used when DataLab is started from the same process as the proxy. In this case, the proxy is directly connected to DataLab main window instance. The typical use case is high-level scripting.

class datalab.control.proxy.**LocalProxy**(*datalab : DLMainWindow | ServerProxy | None = None*)

DataLab local proxy class.

This class provides access to DataLab features from a proxy class. This is the local version of proxy, which is used when DataLab is started from the same process as the proxy.

Paramètres
datalab (DLMainWindow) – DLMainWindow instance.

add_signal(*title : str, xdata : ndarray, ydata : ndarray, xunit : str = '', yunit : str = '', xlabel : str = '', ylabel : str = '', group_id : str = '', set_current : bool = True*) → bool

Add signal data to DataLab.

Paramètres
— **title** – Signal title
— **xdata** – X data
— **ydata** – Y data
— **xunit** – X unit. Defaults to « «
— **yunit** – Y unit. Defaults to « «
— **xlabel** – X label. Defaults to « «
— **ylabel** – Y label. Defaults to « «
— **group_id** – group id in which to add the signal. Defaults to « «
— **set_current** – if True, set the added signal as current

Renvoie
True if signal was added successfully, False otherwise

Lève
— **ValueError** – Invalid xdata dtype
— **ValueError** – Invalid ydata dtype

add_image(*title : str, data : ndarray, xunit : str = '', yunit : str = '', zunit : str = '', xlabel : str = '', ylabel : str = '', zlabel : str = '', group_id : str = '', set_current : bool = True*) → bool

Add image data to DataLab.

Paramètres
— **title** – Image title
— **data** – Image data
— **xunit** – X unit. Defaults to « «
— **yunit** – Y unit. Defaults to « «
— **zunit** – Z unit. Defaults to « «
— **xlabel** – X label. Defaults to « «
— **ylabel** – Y label. Defaults to « «

— **zlabel** – Z label. Defaults to « «
— **group_id** – group id in which to add the image. Defaults to « «
— **set_current** – if True, set the added image as current

Renvoie
True if image was added successfully, False otherwise

Lève
ValueError – Invalid data dtype

add_object(*obj : SignalObj | ImageObj*, *group_id : str = ''*, *set_current : bool = True*) → None

Add object to DataLab.

Paramètres
— **obj** – Signal or image object
— **group_id** – group id in which to add the object. Defaults to « «
— **set_current** – if True, set the added object as current

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

Call computation feature `name`

Note : This calls either the processor's `compute_<name>` method (if it exists), or the processor's `<name>` computation feature (if it is registered, using the `run_feature` method). It looks for the function in all panels, starting with the current one.

Paramètres
— **name** – Compute function name
— **param** – Compute function parameter. Defaults to None

Lève
ValueError – unknown function

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

Get object (signal/image) from index.

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

Renvoie
Object

Lève
KeyError – if object not found

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

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

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

Renvoie
List of plot item shapes

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

Add object annotations (annotation plot items).

Paramètres
— **itemsTrue** – annotation plot items

— **refresh_plotTrue** – refresh plot. Defaults to True
— **panel** – panel name (valid values : « signal », « image »). If None, current panel is used

add_group(*title : str*, *panel : str | None = None*, *select : bool = False*) → None

Add group to DataLab.

Paramètres
— **title** – Group title
— **panel** – Panel name (valid values : « signal », « image »). Defaults to None.
— **select** – Select the group after creation. Defaults to False.

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

Add a label with object title on the associated plot

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

close_application() → None

Close DataLab application

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

Return a context manager to temporarily disable auto refresh.

Renvoie
Context manager

Exemple

```
>>> with proxy.context_no_refresh():
...     proxy.add_image("image1", data1)
...     proxy.calc("fft")
...     proxy.calc("wiener")
...     proxy.calc("ifft")
...     # Auto refresh is disabled during the above operations
```

delete_metadata(*refresh_plot : bool = True*, *keep_roi : bool = False*) → None

Delete metadata of selected objects

Paramètres
— **refresh_plot** – Refresh plot. Defaults to True.
— **keep_roi** – Keep ROI. Defaults to False.

get_current_panel() → str

Return current panel name.

Renvoie
« signal », « image », « macro »))

Type renvoyé
Panel name (valid values

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

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

Renvoie
groups titles, lists of inner objects uuids and titles

Type renvoyé
Tuple

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

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

Paramètres
panel – panel name (valid values : « signal », « image », « macro »). If None, current data panel is used (i.e. signal or image panel).

Renvoie
List of object titles

Lève
ValueError – if panel not found

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

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

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

Renvoie
List of object uuids

Lève
ValueError – if panel not found

classmethod get_public_methods() → list[str]

Return all public methods of the class, except itself.

Renvoie
List of public methods

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

Return selected objects uuids.

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

Renvoie
List of selected objects uuids.

get_version() → str

Return DataLab public version.

Renvoie
DataLab version

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

Open DataLab HDF5 browser to Import HDF5 file.

Paramètres
— **filename** – HDF5 file name
— **reset_all** – Reset all application data. Defaults to None.

import_macro_from_file(*filename : str*) → None

Import macro from file

Paramètres
filename – Filename.

load_from_directory(*path : str*) → None

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

Paramètres
path – directory path

load_from_files(*filenames : list[str]*) → None

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

Paramètres
filenames – list of file names

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

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

Paramètres
— **h5files** – List of HDF5 files to open. Defaults to None.
— **import_all** – Import all objects from HDF5 files. Defaults to None.
— **reset_all** – Reset all application data. Defaults to None.

raise_window() → None

Raise DataLab window

reset_all() → None

Reset all application data

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

Run macro.

Paramètres
number_or_title – Macro number, or macro title. Defaults to None (current macro).

Lève
ValueError – if macro not found

save_to_h5_file(*filename : str*) → None

Save to a DataLab HDF5 file.

Paramètres
filename – HDF5 file name

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

Select groups in current panel.

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

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

Select objects in current panel.

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

set_current_panel(*panel : str*) → None

Switch to panel.

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

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

Stop macro.

Paramètres

number_or_title – Macro number, or macro title. Defaults to None (current macro).

Lève

ValueError – if macro not found

toggle_auto_refresh(*state : bool*) → None

Toggle auto refresh state.

Paramètres

state – Auto refresh state

toggle_show_titles(*state : bool*) → None

Toggle show titles state.

Paramètres

state – Show titles state

Proxy context manager

The proxy context manager is a convenient way to handle proxy creation and destruction. It is used as follows :

```
with proxy_context("local") as proxy:
    proxy.add_signal(...)
```

The proxy type can be « local » or « remote ». For remote proxy, the port can be specified as « remote :port ».

Note : The proxy context manager allows to use the proxy in various contexts (Python script, Jupyter notebook, etc.). It also allows to switch seamlessly between local and remote proxy, keeping the same code inside the context.

datalab.control.proxy.**proxy_context**(*what : str*) → Generator[*LocalProxy* | *RemoteProxy*, None, None]

Context manager handling DL proxy creation and destruction.

Paramètres

what – proxy type (« local » or « remote ») For remote proxy, the port can be specified as « remote :port »

Yields

proxy

LocalProxy if what == « local » RemoteProxy if what == « remote » or « remote :port »

Exemple

with proxy_context(« local ») as proxy :
proxy.add_signal(…)

Calling processor methods using proxy objects

All the proxy objects provide access to the DataLab computing methods exposed by the processor classes :

- `datalab.gui.processor.signal.SignalProcessor`
- `datalab.gui.processor.image.ImageProcessor`

To run a computation feature associated to a processor, you can use the `calc()` method of the proxy object :

```
# Call a method without parameter
proxy.calc("average")

# Call a method with parameters
p = sigima.params.MovingAverageParam.create(n=30)
proxy.calc("moving_average", p)
```

IHM

Le package *datalab.gui* contient des fonctionnalités liées à l'interface graphique (IHM) du projet DataLab. Ces fonctionnalités sont principalement spécifiques à DataLab et ne sont pas destinées à être utilisées de manière indépendante.

Le but de cette section de la documentation est de fournir une vue d'ensemble de l'architecture de l'IHM de DataLab et de décrire les principales fonctionnalités des modules contenus dans ce package. Elle n'a pas pour but de fournir une description détaillée des fonctionnalités de l'IHM, mais plutôt de fournir un point de départ pour le lecteur qui souhaite comprendre l'architecture interne de DataLab.

La fenêtre principale de DataLab est composée de plusieurs parties, chacune d'elles étant gérée par un module spécifique de ce package :

- Les **panneaux de signaux et d'images** : ces panneaux sont utilisés pour afficher les signaux et les images et pour fournir un ensemble d'outils pour les manipuler. Chaque panneau de données s'appuie sur un ensemble de modules pour gérer les fonctionnalités de l'IHM (*datalab.gui.actionhandler* et *datalab.gui.objectview*), le modèle de données (`datalab.gui.objectmodel`), la visualisation des données (*datalab.gui.plothandler*) et le traitement des données (*datalab.gui.processor*).
- Le **panneau des macros** : ce panneau est utilisé pour afficher et exécuter des macros. Il s'appuie sur le module `datalab.gui.macroeditor` pour gérer l'édition et l'exécution des macros.
- Les **widgets spécialisés** : ces widgets sont utilisés pour gérer des fonctionnalités spécifiques telles que l'édition des ROI (*datalab.gui.roieditor*), l'édition des profils d'intensité (`datalab.gui.profiledialog`), etc.

Sous-module	Périmètre
datalab.gui.main	Fenêtre principale et application DataLab
datalab.gui.panel	Panneaux de signaux, d'images et de macros
datalab.gui.actionhandler	Actions de l'application (menus, barres d'outils, menu contextuel)
datalab.gui.objectview	Widgets pour afficher les arbres d'objets (signal/image)
datalab.gui.plothandler	Items graphiques *PlotPy* pour représenter les signaux et les images
datalab.gui.roieditor	Éditeur de ROI
datalab.gui.processor	Processeur
datalab.gui.docks	Widgets de dock
datalab.gui.h5io	Entrée/sortie HDF5

Main window

The *datalab.gui.main* module provides the main window of the DataLab project.

class datalab.gui.main.**DLMainWindow**(*console=None*, *hide_on_close=False*)

DataLab main window

Paramètres
- **console** – enable internal console
- **hide_on_close** – True to hide window on close

static get_instance(*console=None*, *hide_on_close=False*)

Return singleton instance

static xmlrpc_server_started(*port*)

XML-RPC server has started, writing comm port in configuration file

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

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

Renvoie
Groups titles, lists of inner objects uuids and titles

get_object_titles(*panel : Literal['signal', 'image', 'macro'] | None = None*) → list[str]

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

Paramètres
panel – panel name. If None, current data panel is used (i.e. signal or image panel).

Renvoie
List of object titles

Lève
ValueError – if panel is unknown

get_object(*nb_id_title : int | str | None = None*, *panel : Literal['signal', 'image'] | None = None*) → SignalObj | ImageObj

Get object (signal/image) from index.

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

Renvoie
Object

Lève
- **KeyError** – if object not found
- **TypeError** – if index_id_title type is invalid

find_object_by_uuid(*uuid : str*) → SignalObj | ImageObj | ObjectGroup | None

Find an object by UUID, searching across all panels.

This method searches for an object in both signal and image panels, making it suitable for cross-panel operations (e.g., radial profile that takes an ImageObj and produces a SignalObj).

Difference from get_object() : - get_object() requires specifying a panel and accepts number/id/title - find_object_by_uuid() searches all panels automatically using only UUID

Paramètres
uuid – UUID of the object to find

Renvoie
The object if found in any panel, None otherwise

get_object_uuids(*panel : Literal['signal', 'image'] | None = None, group : int | str | None = None*) → list[str]

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

Paramètres
— **panel** – panel name. If None, current panel is used.
— **group** – Group number, or group id, or group title. Defaults to None (all groups).

Renvoie
List of object uuids

Lève
ValueError – if panel is unknown

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

Return selected objects uuids.

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

Renvoie
List of selected objects uuids.

add_group(*title : str, panel : Literal['signal', 'image'] | None = None, select : bool = False*) → None

Add group to DataLab.

Paramètres
— **title** – Group title
— **panel** – Panel name. Defaults to None.
— **select** – Select the group after creation. Defaults to False.

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

Select objects in current panel.

Paramètres
— **selection** – List of object numbers (1 to N) or uuids to select
— **panel** – panel name. If None, current panel is used. Defaults to None.

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

Select groups in current panel.

Paramètres
— **selection** – List of group numbers (1 to N), or list of group uuids, or None to select all groups. Defaults to None.
— **panel** – panel name. If None, current panel is used. Defaults to None.

delete_metadata(*refresh_plot : bool = True, keep_roi : bool = False*) → None

Delete metadata of selected objects

Paramètres
— **refresh_plot** – Refresh plot. Defaults to True.
— **keep_roi** – Keep ROI. Defaults to False.

get_object_shapes(*nb_id_title : int | str | None = None, panel : Literal['signal', 'image'] | None = None*) → list

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

Paramètres

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

Renvoie
List of plot item shapes

add_annotations_from_items(*items : list, refresh_plot : bool = True, panel : Literal['signal', 'image'] | None = None*) → None

Add object annotations (annotation plot items).

Paramètres
— **items** – annotation plot items
— **refresh_plot** – refresh plot. Defaults to True.
— **panel** – panel name. If None, current panel is used.

add_label_with_title(*title : str | None = None, panel : Literal['signal', 'image'] | None = None*) → None

Add a label with object title on the associated plot

Paramètres
— **title** – Label title. Defaults to None. If None, the title is the object title.
— **panel** – panel name. If None, current panel is used.

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

Run macro.

Paramètres
number – Number of the macro (starting at 1). Defaults to None (run current macro, or does nothing if there is no macro).

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

Stop macro.

Paramètres
number – Number of the macro (starting at 1). Defaults to None (stop current macro, or does nothing if there is no macro).

import_macro_from_file(*filename : str*) → None

Import macro from file

Paramètres
filename – Filename.

property panels: tuple[*AbstractPanel*, ...]

Return the tuple of implemented panels (signal, image)

Renvoie
Tuple of panels

confirm_memory_state() → bool

Check memory warning state and eventually show a warning dialog

Renvoie
True if memory state is ok

check_stable_release() → None

Check if this is a stable release

check_for_previous_crash() → None

Check for previous crash

check_for_v020_plugins() → None

Check for v0.20 plugins and warn user if any are found

execute_post_show_actions() → None

Execute post-show actions

take_screenshot(*name : str*) → None

Take main window screenshot

take_menu_screenshots() → None

Take menu screenshots

setup(*console : bool = False*) → None

Setup main window

Paramètres
console – True to setup console

set_process_isolation_enabled(*state : bool*) → None

Enable/disable process isolation

Paramètres
state – True to enable process isolation

get_current_panel() → str

Return current panel name

Renvoie
« signal », « image », « macro »)

Type renvoyé
Panel name (valid values

set_current_panel(*panel : Literal['signal', 'image', 'macro'] | BaseDataPanel*) → None

Switch to panel.

Paramètres
panel – panel name or panel instance

Lève
ValueError – unknown panel

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

Call computation feature `name`

Note : This calls either the processor's `compute_<name>` method (if it exists), or the processor's `<name>` computation feature (if it is registered, using the `run_feature` method). It looks for the function in all panels, starting with the current one.

Paramètres
— **name** – Compute function name
— **param** – Compute function parameter. Defaults to None.

Lève
ValueError – unknown function

has_objects() → bool

Return True if sig/ima panels have any object

set_modified(*state : bool = True*) → None

Set mainwindow modified state

repopulate_panel_trees() → None

Repopulate all panel trees

toggle_show_titles(*state : bool*) → None

Toggle show annotations option

Paramètres
state – state

handle_autorefresh_action(*state : bool*) → None

Handle auto-refresh action from UI (with confirmation dialog)

Paramètres
state – desired state

toggle_auto_refresh(*state : bool*) → None

Toggle auto refresh option

Paramètres
state – state

toggle_show_first_only(*state : bool*) → None

Toggle show first only option

Paramètres
state – state

reset_all() → None

Reset all application data

save_to_h5_file(*filename=None*) → None

Save to a DataLab HDF5 file

Paramètres
filename – HDF5 filename. If None, a file dialog is opened.

Lève
IOError – if filename is invalid or file cannot be saved.

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

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

Paramètres
— **h5files** – HDF5 filenames (optionally with dataset name, separated by « : »)
— **import_all** – Import all datasets from HDF5 files
— **reset_all** – Reset all application data before importing

browse_h5_files(*filenames : list[str], reset_all : bool*) → None

Browse HDF5 files

Paramètres
— **filenames** – HDF5 filenames
— **reset_all** – Reset all application data before importing

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

Import HDF5 file into DataLab

Paramètres
— **filename** – HDF5 filename (optionally with dataset name,
— **"** (*separated by*) – «)
— **reset_all** – Delete all DataLab signals/images before importing data

add_object(*obj : SignalObj | ImageObj*, *group_id : str = ''*, *set_current=True*) → None

Add object - signal or image

Paramètres

- **obj** – object to add (signal or image)
- **group_id** – group ID (optional)
- **set_current** – True to set the object as current object

load_from_files(*filenames : list[str]*) → None

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

Paramètres

filenames – list of filenames

load_from_directory(*path : str*) → None

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

Paramètres

path – directory path

get_version() → str

Return DataLab public version.

Renvoie

DataLab version

close_application() → None

Close DataLab application

raise_window() → None

Raise DataLab window

add_signal(*title : str*, *xdata : ndarray*, *ydata : ndarray*, *xunit : str = ''*, *yunit : str = ''*, *xlabel : str = ''*, *ylabel : str = ''*, *group_id : str = ''*, *set_current : bool = True*) → bool

Add signal data to DataLab.

Paramètres

- **title** – Signal title
- **xdata** – X data
- **ydata** – Y data
- **xunit** – X unit. Defaults to « «
- **yunit** – Y unit. Defaults to « «
- **xlabel** – X label. Defaults to « «
- **ylabel** – Y label. Defaults to « «
- **group_id** – group id in which to add the signal. Defaults to « «
- **set_current** – if True, set the added signal as current

Renvoie

True if signal was added successfully, False otherwise

Lève

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

add_image(*title : str*, *data : ndarray*, *xunit : str = ''*, *yunit : str = ''*, *zunit : str = ''*, *xlabel : str = ''*, *ylabel : str = ''*, *zlabel : str = ''*, *group_id : str = ''*, *set_current : bool = True*) → bool

Add image data to DataLab.

Paramètres

- **title** – Image title
- **data** – Image data
- **xunit** – X unit. Defaults to « «

— **yunit** – Y unit. Defaults to « «
— **zunit** – Z unit. Defaults to « «
— **xlabel** – X label. Defaults to « «
— **ylabel** – Y label. Defaults to « «
— **zlabel** – Z label. Defaults to « «
— **group_id** – group id in which to add the image. Defaults to « «
— **set_current** – if True, set the added image as current

Renvoie
True if image was added successfully, False otherwise

Lève
ValueError – Invalid data dtype

play_demo() → None

Play demo

show_tour() → None

Show tour

static test_segfault_error() → None

Generate errors (both fault and traceback)

show() → None

Reimplement QMainWindow method

close_properly() → bool

Close properly

Renvoie
True if closed properly, False otherwise

closeEvent(*event : QCloseEvent*) → None

Reimplement QMainWindow method

Panel

The *datalab.gui.panel* package provides the **panel objects** for signals and images.

Three types of panels are available :
— *datalab.gui.panel.signal.SignalPanel* : Signal panel
— *datalab.gui.panel.image.ImagePanel* : Image panel
— *datalab.gui.panel.macro.MacroPanel* : Macro panel

Signal and Image Panels are called **Data Panels** and are used to display and handle signals and images in the main window of DataLab.

Data Panels rely on the *datalab.gui.panel.base.ObjectProp* class (managing the object properties) and a set of modules to handle the GUI features :
— *datalab.gui.actionhandler* : Application actions (menus, toolbars, context menu)
— *datalab.gui.objectview* : Widgets to display object (signal/image) trees
— *datalab.gui.plothandler* : *PlotPy* items for representing signals and images
— *datalab.gui.processor* : Processor (computation)
— datalab.gui.panel.roieditor : ROI editor

The Macro Panel is used to display and run macros. It relies on the datalab.gui.macroeditor module to handle the macro edition and execution.

Base features

`datalab.gui.panel.base.is_plot_item_serializable(item : Any) → bool`

Return True if plot item is serializable

`datalab.gui.panel.base.is_hdf5_file(filename : str, check_content : bool = False) → bool`

Return True if filename has an HDF5 extension or is an HDF5 file.

Paramètres
- **filename** – Path to the file to check
- **check_content** – If True, also attempts to open the file to verify it's a valid HDF5 file. If False, only checks the file extension.

Renvoie
True if the file is (likely) an HDF5 file, False otherwise.

`class datalab.gui.panel.base.ProcessingReport(success : bool, obj_uuid : str | None = None, message : str | None = None)`

Report of processing operation

Paramètres
- **success** – True if processing succeeded
- **obj_uuid** – UUID of the processed object
- **message** – Optional message (error or info)

`class datalab.gui.panel.base.ObjectProp(panel : BaseDataPanel, objclass : SignalObj | ImageObj)`

Object handling panel properties

Paramètres
- **panel** – parent data panel
- **objclass** – class of the object handled by the panel (SignalObj or ImageObj)

`display_analysis_parameters(obj : SignalObj | ImageObj) → bool`

Set analysis parameter label.

Paramètres
obj – Signal or Image object

Renvoie
True if analysis parameters were found and displayed, False otherwise.

`display_processing_history(obj : SignalObj | ImageObj) → bool`

Display processing history.

Paramètres
obj – Signal or Image object

Renvoie
True if processing history was found and displayed, False otherwise.

`update_properties_from(obj : SignalObj | ImageObj | None = None) → None`

Update properties panel (properties, creation, processing) from object.

Paramètres
obj – Signal or Image object

`get_changed_properties() → dict[str, Any]`

Get dictionary of properties that have changed from original values.

Renvoie
Dictionary mapping property names to their new values, containing only the properties that were modified by the user.

update_original_values() → None

Update the stored original values to the current dataset values.

This should be called after applying changes to reset the baseline for detecting future changes.

setup_creation_tab(*obj : SignalObj | ImageObj*, *set_current : bool = False*) → bool

Setup the Creation tab with parameter editor for interactive object creation.

Paramètres
- **obj** – Signal or Image object
- **set_current** – If True, set the Creation tab as current after creation

Renvoie
True if Creation tab was set up, False otherwise

apply_creation_parameters() → None

Apply creation parameters : recreate object with updated parameters.

setup_processing_tab(*obj : SignalObj | ImageObj*, *reset_params : bool = True*, *set_current : bool = False*) → bool

Setup the Processing tab with parameter editor for re-processing.

Paramètres
- **obj** – Signal or Image object
- **reset_params** – If True, call update_from_obj() to reset parameters from source object. If False, use parameters as stored in metadata.
- **set_current** – If True, set the Processing tab as current after creation

Renvoie
True if Processing tab was set up, False otherwise

apply_processing_parameters(*obj : SignalObj | ImageObj | None = None*, *interactive : bool = True*) → *ProcessingReport*

Apply processing parameters : re-run processing with updated parameters.

Paramètres
- **obj** – Signal or Image object to reprocess. If None, uses the current object.
- **interactive** – If True, show progress and error messages in the UI.

Renvoie
ProcessingReport with success status, object UUID, and optional message.

class datalab.gui.panel.base.**AbstractPanelMeta**(*name*, *bases*, *namespace*, ***kwargs*)

Mixed metaclass to avoid conflicts

class datalab.gui.panel.base.**AbstractPanel**(*parent*)

Object defining DataLab panel interface, based on a vertical QSplitter widget

A panel handle an object list (objects are signals, images, macros…). Each object must implement `datalab.gui.ObjItf` interface

get_serializable_name(*obj : ObjItf*) → str

Return serializable name of object

serialize_object_to_hdf5(*obj : ObjItf*, *writer : NativeH5Writer*) → None

Serialize object to HDF5 file

deserialize_object_from_hdf5(*reader : NativeH5Reader*, *name : str*, *reset_all : bool = False*) → ObjItf

Deserialize object from a HDF5 file

Paramètres
- **reader** – HDF5 reader
- **name** – Object name in HDF5 file

— **reset_all** – If True, preserve original UUIDs (workspace reload). If False, regenerate UUIDs (importing objects).

abstract serialize_to_hdf5(*writer : NativeH5Writer*) → None

Serialize whole panel to a HDF5 file

abstract deserialize_from_hdf5(*reader : NativeH5Reader*, *reset_all : bool = False*) → None

Deserialize whole panel from a HDF5 file

Paramètres

— **reader** – HDF5 reader

— **reset_all** – If True, preserve original UUIDs (workspace reload). If False, regenerate UUIDs (importing objects).

abstract create_object() → ObjItf

Create and return object

abstract add_object(*obj : ObjItf*) → None

Add object to panel

abstract remove_all_objects()

Remove all objects

class datalab.gui.panel.base.**PasteMetadataParam**

Paste metadata parameters

keep_roi

Default : True.

Type

guidata.dataset.dataitems.BoolItem

keep_geometry

Default : False.

Type

guidata.dataset.dataitems.BoolItem

keep_tables

Default : False.

Type

guidata.dataset.dataitems.BoolItem

keep_other

Default : True.

Type

guidata.dataset.dataitems.BoolItem

classmethod create(*keep_roi : bool*, *keep_geometry : bool*, *keep_tables : bool*, *keep_other : bool*) → *datalab.gui.panel.base.PasteMetadataParam*

Returns a new instance of `PasteMetadataParam` with the fields set to the given values.

Paramètres

— **keep_roi** (*bool*) – Default : True.

— **keep_geometry** (*bool*) – Default : False.

— **keep_tables** (*bool*) – Default : False.

— **keep_other** (*bool*) – Default : True.

Renvoie

New instance of `PasteMetadataParam`.

class datalab.gui.panel.base.**NonModalInfoDialog**(*parent : QWidget*, *title : str*, *text : str*)

Non-modal information message box with selectable text.

This widget displays an information message in a message dialog box, allowing users to select and copy the text content.

class datalab.gui.panel.base.**SaveToDirectoryGUIParam**

Save to directory parameters

directory

Répertoire. Chaîne. Default : “”.

Type
guidata.dataset.dataitems.DirectoryItem

basename

Nom de base. Chaîne de format python. Voir la description pour plus de détails. Chaîne. Default : “{title}”.

Type
guidata.dataset.dataitems.StringItem

help

Aide. Default : None.

Type
guidata.dataset.dataitems.ButtonItem

extension

Default : None.

Type
guidata.dataset.dataitems.ChoiceItem

overwrite

Écraser les fichiers existants. Default : False.

Type
guidata.dataset.dataitems.BoolItem

preview

Aperçu. Chaîne, expr. Rég. : `^(?!invalid).\*`. Default : None.

Type
guidata.dataset.dataitems.TextItem

classmethod create(*directory : str*, *basename : str*, *help : type*, *extension : Any*, *overwrite : bool*, *preview : str*) → *datalab.gui.panel.base.SaveToDirectoryGUIParam*

Returns a new instance of `SaveToDirectoryGUIParam` with the fields set to the given values.

Paramètres

- **directory** (*str*) – Répertoire. Chaîne. Default : “”.
- **basename** (*str*) – Nom de base. Chaîne de format python. Voir la description pour plus de détails. Chaîne. Default : “{title}”.
- **help** (*type*) – Aide. Default : None.
- **extension** (*Any*) – Default : None.
- **overwrite** (*bool*) – Écraser les fichiers existants. Default : False.
- **preview** (*str*) – Aperçu. Chaîne, expr. Rég. : `^(?!invalid).\*`. Default : None.

Renvoie
New instance of `SaveToDirectoryGUIParam`.

on_button_click(*_item : ButtonItem*, *_value : None*, *parent : QWidget*) → None

Help button callback.

get_extension_choices(*_item=None*, *_value=None*)

Return list of available extensions for choice item.

build_filenames(*objs : list[TypeObj] | None = None*) → list[str]

Build filenames according to current parameters.

generate_filepath_obj_pairs(*objs : list[TypeObj]*) → Generator[tuple[str, TypeObj], None, None]

Iterate over (filepath, object) pairs to be saved.

update_preview(*_item=None*, *_value=None*) → None

Update preview.

class datalab.gui.panel.base.**AddMetadataParam**

Add metadata parameters

metadata_key

Clé de métadonnées. Le nom de la clé pour l'élément de métadonnées. Chaîne, non vide, expr. Rég. : `^[a-za-z_][a-za-z0-9_]\*$`. Default : "custom_key".

Type
guidata.dataset.dataitems.StringItem

value_pattern

Modèle de valeur. Chaîne de format python. Voir la description pour plus de détails. Chaîne. Default : "{index}".

Type
guidata.dataset.dataitems.StringItem

help

Aide. Default : None.

Type
guidata.dataset.dataitems.ButtonItem

conversion

Default : "string".

Type
guidata.dataset.dataitems.ChoiceItem

preview

Aperçu. Chaîne, expr. Rég. : `^(?!invalid).\*`. Default : "".

Type
guidata.dataset.dataitems.TextItem

classmethod create(*metadata_key : str*, *value_pattern : str*, *help : type*, *conversion : Any*, *preview : str*) → *datalab.gui.panel.base.AddMetadataParam*

Returns a new instance of `AddMetadataParam` with the fields set to the given values.

Paramètres

- **metadata_key** (*str*) – Clé de métadonnées. Le nom de la clé pour l'élément de métadonnées. Chaîne, non vide, expr. Rég. : `^[a-za-z_][a-za-z0-9_]\*$`. Default : "custom_key".
- **value_pattern** (*str*) – Modèle de valeur. Chaîne de format python. Voir la description pour plus de détails. Chaîne. Default : "{index}".
- **help** (*type*) – Aide. Default : None.
- **conversion** (*Any*) – Default : "string".
- **preview** (*str*) – Aperçu. Chaîne, expr. Rég. : `^(?!invalid).\*`. Default : "".

Renvoie
New instance of `AddMetadataParam`.

`on_help_button_click`(*_item : ButtonItem*, *_value : None*, *parent : QWidget*) → None

Help button callback.

`get_conversion_choices`(*_item=None*, *_value=None*)

Return list of available conversion choices.

`build_values`(*objs : list[TypeObj] | None = None*) → list[str | float | int | bool]

Build values according to current parameters.

Lève
`ValueError` – If a value cannot be converted to the target type.

`update_preview`(*_item=None*, *_value=None*) → None

Update preview.

class `datalab.gui.panel.base.BaseDataPanel`(*parent : QWidget*)

Object handling the item list, the selected item properties and plot

abstract static `get_roi_class`() → Type[TypeROI]

Return ROI class

abstract static `get_roieditor_class`() → Type[TypeROIEditor]

Return ROI editor class

`closeEvent`(*event*)

Reimplement QMainWindow method

`plot_item_parameters_changed`(*item : CurveItem | MaskedXYImageItem | LabelItem*) → None

Plot items changed : update metadata of all objects from plot items

`plot_item_moved`(*item : LabelItem*, *x0 : float*, *y0 : float*, *x1 : float*, *y1 : float*) → None

Plot item moved : update metadata of all objects from plot items

Paramètres
- `item` – Plot item
- `x0` – new x0 coordinate
- `y0` – new y0 coordinate
- `x1` – new x1 coordinate
- `y1` – new y1 coordinate

`serialize_object_to_hdf5`(*obj : TypeObj*, *writer : NativeH5Writer*) → None

Serialize object to HDF5 file

`serialize_to_hdf5`(*writer : NativeH5Writer*) → None

Serialize whole panel to a HDF5 file

`deserialize_from_hdf5`(*reader : NativeH5Reader*, *reset_all : bool = False*) → None

Deserialize whole panel from a HDF5 file

Paramètres
- `reader` – HDF5 reader
- `reset_all` – If True, preserve original UUIDs (workspace reload). If False, regenerate UUIDs (importing objects).

`create_object`() → TypeObj

Create object (signal or image)

Renvoie
SignalObj or ImageObj object

`add_object`(*obj : TypeObj*, *group_id : str | None = None*, *set_current : bool = True*) → None

Add object

Paramètres
- **obj** – SignalObj or ImageObj object
- **group_id** – group id to which the object belongs. If None or empty string, the object is added to the current group.
- **set_current** – if True, set the added object as current

`remove_all_objects`() → None

Remove all objects

`setup_panel`() → None

Setup panel

`refresh_plot`(*what : str*, *update_items : bool = True*, *force : bool = False*, *only_visible : bool = True*, *only_existing : bool = False*) → None

Refresh plot.

Paramètres
- **what** – string describing the objects to refresh. Valid values are « selected » (refresh the selected objects), « all » (refresh all objects), « existing » (refresh existing plot items), or an object uuid.
- **update_items** – if True, update the items. If False, only show the items (do not update them). Defaults to True.
- **force** – if True, force refresh even if auto refresh is disabled. Defaults to False.
- **only_visible** – if True, only refresh visible items. Defaults to True. Visible items are the ones that are not hidden by other items or the items except the first one if the option « Show first only » is enabled. This is useful for images, where the last image is the one that is shown. If False, all items are refreshed.
- **only_existing** – if True, only refresh existing items. Defaults to False. Existing items are the ones that have already been created and are associated to the object uuid. If False, create new items for the objects that do not have an item yet.

Lève
ValueError – if *what* is not a valid value

`manual_refresh`() → None

Manual refresh

`get_category_actions`(*category : ActionCategory*) → list[QAction]

Return actions for category

`get_context_menu`() → QMenu

Update and return context menu

`add_group`(*title : str*, *select : bool = False*) → ObjectGroup

Add group

Paramètres
- **title** – group title
- **select** – if True, select the group in the tree view. Defaults to False.

Renvoie
Created group object

`duplicate_object`() → None

Duplication signal/image object

copy_metadata() → None

Copy object metadata

paste_metadata(*param : PasteMetadataParam | None = None*) → None

Paste metadata to selected object(s)

add_metadata(*param : AddMetadataParam | None = None*) → None

Add metadata item to selected object(s)

Paramètres

param – Add metadata parameters

copy_roi() → None

Copy regions of interest

paste_roi() → None

Paste regions of interest

remove_object(*force : bool = False*) → None

Remove signal/image object

Paramètres

force – if True, remove object without confirmation. Defaults to False.

delete_all_objects() → None

Confirm before removing all objects

delete_metadata(*refresh_plot : bool = True, keep_roi : bool | None = None*) → None

Delete metadata of selected objects

Paramètres

— **refresh_plot** – Refresh plot. Defaults to True.

— **keep_roi** – Keep regions of interest, if any. Defaults to None (ask user).

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

Add object annotations (annotation plot items).

Paramètres

— **items** – annotation plot items

— **refresh_plot** – refresh plot. Defaults to True.

update_metadata_view_settings() → None

Update metadata view settings

copy_titles_to_clipboard() → None

Copy object titles to clipboard (for reproducibility)

new_group() → None

Create a new group

rename_selected_object_or_group(*new_name : str | None = None*) → None

Rename selected object or group

Paramètres

new_name – new name (default : None, i.e. ask user)

abstract get_newparam_from_current(*newparam : NewSignalParam | NewImageParam | None = None*) → NewSignalParam | NewImageParam | None

Get new object parameters from the current object.

Paramètres

newparam – new object parameters. If None, create a new one.

Renvoie
New object parameters

abstract new_object(*param : NewSignalParam | NewImageParam | None = None, edit : bool = False, add_to_panel : bool = True*) → TypeObj | None

Create a new object (signal/image).

Paramètres
- **param** – new object parameters
- **edit** – Open a dialog box to edit parameters (default : False). When False, the object is created with default parameters and creation parameters are stored in metadata for interactive editing.
- **add_to_panel** – Add object to panel (default : True)

Renvoie
New object

set_current_object_title(*title : str*) → None

Set current object title

load_from_directory(*directory : str | None = None*) → list[TypeObj]

Open objects from directory (signals or images, depending on the panel), add them to DataLab and return them. If the directory is not specified, ask the user to select a directory.

Paramètres
directory – directory name

Renvoie
list of new objects

load_from_files(*filenames : list[str] | None = None, create_group : bool = False, add_objects : bool = True, ignore_errors : bool = False*) → list[TypeObj]

Open objects from file (signals/images), add them to DataLab and return them.

Paramètres
- **filenames** – File names
- **create_group** – if True, create a new group if more than one object is loaded for a single file. Defaults to False : all objects are added to the current group.
- **add_objects** – if True, add objects to the panel. Defaults to True.
- **ignore_errors** – if True, ignore errors when loading files. Defaults to False.

Renvoie
list of new objects

save_to_files(*filenames : list[str] | str | None = None*) → None

Save selected objects to files (signal/image).

Paramètres
filenames – File names

save_to_directory(*param : SaveToDirectoryParam | None = None*) → None

Save signals or images to directory using a filename pattern.

Opens a dialog to select the output directory, the basename pattern and the extension.

Paramètres
param – parameters.

handle_dropped_files(*filenames : list[str] | None = None*) → None

Handle dropped files

Paramètres
filenames – File names

Renvoie
None

`exec_import_wizard()` → None

Execute import wizard

`import_metadata_from_file(filename : str | None = None)` → None

Import metadata from file (JSON).

Paramètres
`filename` – File name

`export_metadata_from_file(filename : str | None = None)` → None

Export metadata to file (JSON).

Paramètres
`filename` – File name

`copy_annotations()` → None

Copy annotations from selected object

`paste_annotations()` → None

Paste annotations to selected object(s)

`import_annotations_from_file(filename : str | None = None)` → None

Import annotations from file (JSON).

Paramètres
`filename` – File name

`export_annotations_from_file(filename : str | None = None)` → None

Export annotations to file (JSON).

Paramètres
`filename` – File name

`delete_annotations()` → None

Delete all annotations from selected object(s)

`import_roi_from_file(filename : str | None = None)` → None

Import regions of interest from file (JSON).

Paramètres
`filename` – File name

`export_roi_to_file(filename : str | None = None)` → None

Export regions of interest to file (JSON).

Paramètres
`filename` – File name

`selection_changed(update_items : bool = False)` → None

Object selection changed : update object properties, refresh plot and update object view.

Paramètres
`update_items` – Update plot items (default : False)

`properties_changed()` → None

The properties "Apply" button was clicked : update object properties, refresh plot and update object view.

`recompute_processing()` → None

Recompute/rerun selected objects or group with stored processing parameters.

This method handles both single objects and groups. For each object, it checks if it has 1-to-1 processing parameters that can be recomputed. Objects without recomputable parameters are skipped.

select_source_objects() → None

Select source objects associated with the selected object's processing.

This method retrieves the source object UUIDs from the selected object's processing parameters and selects them in the object view.

add_plot_items_to_dialog(*dlg : PlotDialog*, *oids : list[str]*) → None

Add plot items to dialog

Paramètres
- **dlg** – Dialog
- **oids** – Object IDs

open_separate_view(*oids : list[str] | None = None*, *edit_annotations : bool = False*) → PlotDialog | None

Open separate view for visualizing selected objects

Paramètres
- **oids** – Object IDs (default : None)
- **edit_annotations** – Edit annotations (default : False)

Renvoie
Instance of PlotDialog

view_images_side_by_side(*oids : list[str] | None = None*) → None

View selected images side-by-side in a grid layout

Paramètres
oids – Object IDs (default : None, uses selected objects)

get_dialog_size() → tuple[int, int]

Get dialog size (minimum and maximum)

create_new_dialog(*edit : bool = False*, *toolbar : bool = True*, *title : str | None = None*, *name : str | None = None*, *options : dict[str, Any] | None = None*) → PlotDialog | None

Create new pop-up signal/image plot dialog.

Paramètres
- **edit** – Edit mode
- **toolbar** – Show toolbar
- **title** – Dialog title
- **name** – Dialog object name
- **options** – Plot options

Renvoie
Plot dialog instance

get_roi_editor_output(*mode : Literal['apply', 'extract', 'define'] = 'apply'*) → tuple[TypeROI, bool] | None

Get ROI data (array) from specific dialog box.

Paramètres
mode – Mode of operation, either « apply » (define ROI, then apply it to selected objects), « extract » (define ROI, then extract data from it), or « define » (define ROI without applying or extracting).

Renvoie
A tuple containing the ROI object and a boolean indicating whether the dialog was accepted or not.

get_objects_with_dialog(*title : str*, *comment : str = ''*, *nb_objects : int = 1*, *parent : QW.QWidget | None = None*) → TypeObj | None

Get object with dialog box.

Paramètres

— **title** – Dialog title
— **comment** – Optional dialog comment
— **nb_objects** – Number of objects to select
— **parent** – Parent widget

Renvoie

Object(s) (signal(s) or image(s), or None if dialog was canceled)

show_results() → None

Show results

toggle_result_label_visibility(*state : bool*) → None

Toggle the visibility of the merged result label on the plot.

Paramètres

state – True to show the label, False to hide it

plot_results(*kind : str | None = None*, *xaxis : str | None = None*, *yaxis : str | None = None*) → None

Plot results

Paramètres

— **kind** – Plot kind. Either « one_curve_per_object » or « one_curve_per_title ». If None, show dialog to get parameters.
— **xaxis** – X axis column name. If None, show dialog to get parameters.
— **yaxis** – Y axis column name. If None, show dialog to get parameters.

delete_results() → None

Delete results

add_label_with_title(*title : str | None = None*, *ignore_msg : bool = True*) → None

Add a label with object title on the associated plot

Paramètres

— **title** – Label title. Defaults to None. If None, the title is the object title.
— **ignore_msg** – If True, do not show the information message. Defaults to True. If False, show a message box to inform the user that the label has been added as an annotation, and that it can be edited or removed using the annotation editing window.

Signal panel

class datalab.gui.panel.signal.**SignalPanel**(*parent : QW.QWidget*, *dockableplotwidget : DockablePlotWidget*, *panel_toolbar : QW.QToolBar*)

Object handling the item list, the selected item properties and plot, specialized for Signal objects

PARAMCLASS

Signal object

title

Titre du signal. Chaîne. Default : “Sans titre”.

Type

guidata.dataset.dataitems.StringItem

xydata

Données. Default : None.

Type

guidata.dataset.dataitems.FloatArrayItem

metadata

Métadonnées. Default : {}.

Type

guidata.dataset.dataitems.DictItem

annotations

Chaîne. Default : “”.

Type

guidata.dataset.dataitems.StringItem

xlabel

Titre. Chaîne. Default : “”.

Type

guidata.dataset.dataitems.StringItem

xunit

Unité. Chaîne. Default : “”.

Type

guidata.dataset.dataitems.StringItem

ylabel

Titre. Chaîne. Default : “”.

Type

guidata.dataset.dataitems.StringItem

yunit

Unité. Chaîne. Default : “”.

Type

guidata.dataset.dataitems.StringItem

autoscale

Default : True.

Type

guidata.dataset.dataitems.BoolItem

xscalelog

Default : False.

Type

guidata.dataset.dataitems.BoolItem

xscalemin

Borne inférieure. Flottant. Default : None.

Type

guidata.dataset.dataitems.FloatItem

xscalemax

Borne supérieure. Flottant. Default : None.

Type

guidata.dataset.dataitems.FloatItem

yscalelog

Default : False.

Type
guidata.dataset.dataitems.BoolItem

yscalemin

Borne inférieure. Flottant. Default : None.

Type
guidata.dataset.dataitems.FloatItem

yscalemax

Borne supérieure. Flottant. Default : None.

Type
guidata.dataset.dataitems.FloatItem

alias de `SignalObj`

classmethod `PARAMCLASS.create`(*title : str*, *xydata : numpy.ndarray*, *metadata : dict*, *annotations : str*, *xlabel : str*, *xunit : str*, *ylabel : str*, *yunit : str*, *autoscale : bool*, *xscalelog : bool*, *xscalemin : float*, *xscalemax : float*, *yscalelog : bool*, *yscalemin : float*, *yscalemax : float*) → sigima.objects.signal.object.SignalObj

Returns a new instance of `SignalObj` with the fields set to the given values.

Paramètres

- **title** (*str*) – Titre du signal. Chaîne. Default : "Sans titre".
- **xydata** (*numpy.ndarray*) – Données. Default : None.
- **metadata** (*dict*) – Métadonnées. Default : {}.
- **annotations** (*str*) – Chaîne. Default : "".
- **xlabel** (*str*) – Titre. Chaîne. Default : "".
- **xunit** (*str*) – Unité. Chaîne. Default : "".
- **ylabel** (*str*) – Titre. Chaîne. Default : "".
- **yunit** (*str*) – Unité. Chaîne. Default : "".
- **autoscale** (*bool*) – Default : True.
- **xscalelog** (*bool*) – Default : False.
- **xscalemin** (*float*) – Borne inférieure. Flottant. Default : None.
- **xscalemax** (*float*) – Borne supérieure. Flottant. Default : None.
- **yscalelog** (*bool*) – Default : False.
- **yscalemin** (*float*) – Borne inférieure. Flottant. Default : None.
- **yscalemax** (*float*) – Borne supérieure. Flottant. Default : None.

Renvoie
New instance of `SignalObj`.

IO_REGISTRY

alias de `SignalIORegistry`

static get_roi_class() → Type[SignalROI]

Return ROI class

static get_roieditor_class() → Type[*SignalROIEditor*]

Return ROI editor class

get_newparam_from_current(*newparam : NewSignalParam | None = None*, *title : str | None = None*) → NewSignalParam | None

Get new object parameters from the current object.

Paramètres

— **newparam** (*guidata.dataset.DataSet*) – new object parameters. If None, create a new one.
— **title** – new object title. If None, use the current object title, or the default title.

Renvoie
New object parameters

new_object(*param : NewSignalParam | None = None, edit : bool = False, add_to_panel : bool = True*) → SignalObj | None

Create a new object (signal).

Paramètres
— **param** (*guidata.dataset.DataSet*) – new object parameters
— **edit** (*bool*) – Open a dialog box to edit parameters (default : False). When False, the object is created with default parameters and creation parameters are stored in metadata for interactive editing.
— **add_to_panel** (*bool*) – Add the new object to the panel (default : True)

Renvoie
New object

toggle_anti_aliasing(*state : bool*) → None

Toggle anti-aliasing on/off

Paramètres
state – state of the anti-aliasing

reset_curve_styles() → None

Reset curve styles

Image panel

class datalab.gui.panel.image.**ImagePanel**(*parent : QW.QWidget, dockableplotwidget : DockablePlotWidget, panel_toolbar : QW.QToolBar*)

Object handling the item list, the selected item properties and plot, specialized for Image objects

PARAMCLASS

Image object

data

Données. Default : None.

Type
guidata.dataset.dataitems.FloatArrayItem

metadata

Métadonnées. Default : {}.

Type
guidata.dataset.dataitems.DictItem

annotations

Chaîne. Default : “”.

Type
guidata.dataset.dataitems.StringItem

is_uniform_coords

Default : True.

Type
guidata.dataset.dataitems.BoolItem

x0

X_0. Flottant. Default : 0.0.

Type
guidata.dataset.dataitems.FloatItem

y0

Y_0. Flottant. Default : 0.0.

Type
guidata.dataset.dataitems.FloatItem

dx

x. Flottant. Default : 1.0.

Type
guidata.dataset.dataitems.FloatItem

dy

y. Flottant. Default : 1.0.

Type
guidata.dataset.dataitems.FloatItem

xmin

X_{MIN}. Flottant. Default : None.

Type
guidata.dataset.dataitems.FloatItem

xmax

X_{MAX}. Flottant. Default : None.

Type
guidata.dataset.dataitems.FloatItem

ymin

Y_{MIN}. Flottant. Default : None.

Type
guidata.dataset.dataitems.FloatItem

ymax

Y_{MAX}. Flottant. Default : None.

Type
guidata.dataset.dataitems.FloatItem

xcoords

Coordonnées X. Default : array([], dtype=float64).

Type
guidata.dataset.dataitems.FloatArrayItem

ycoords

Coordonnées Y. Default : array([], dtype=float64).

Type
guidata.dataset.dataitems.FloatArrayItem

title

Titre de l'image. Chaîne. Default : "Sans titre".

Type

guidata.dataset.dataitems.StringItem

xlabel

Titre. Chaîne. Default : "".

Type

guidata.dataset.dataitems.StringItem

xunit

Unité. Chaîne. Default : "".

Type

guidata.dataset.dataitems.StringItem

ylabel

Titre. Chaîne. Default : "".

Type

guidata.dataset.dataitems.StringItem

yunit

Unité. Chaîne. Default : "".

Type

guidata.dataset.dataitems.StringItem

zlabel

Titre. Chaîne. Default : "".

Type

guidata.dataset.dataitems.StringItem

zunit

Unité. Chaîne. Default : "".

Type

guidata.dataset.dataitems.StringItem

autoscale

Default : True.

Type

guidata.dataset.dataitems.BoolItem

xscalelog

Default : False.

Type

guidata.dataset.dataitems.BoolItem

xscalemin

Borne inférieure. Flottant. Default : None.

Type

guidata.dataset.dataitems.FloatItem

xscalemax

Borne supérieure. Flottant. Default : None.

Type

guidata.dataset.dataitems.FloatItem

yscalelog

Default : False.

Type
guidata.dataset.dataitems.BoolItem

yscalemin

Borne inférieure. Flottant. Default : None.

Type
guidata.dataset.dataitems.FloatItem

yscalemax

Borne supérieure. Flottant. Default : None.

Type
guidata.dataset.dataitems.FloatItem

zscalemin

Borne inférieure. Flottant. Default : None.

Type
guidata.dataset.dataitems.FloatItem

zscalemax

Borne supérieure. Flottant. Default : None.

Type
guidata.dataset.dataitems.FloatItem

alias de `ImageObj`

classmethod PARAMCLASS.**create**(*data : numpy.ndarray, metadata : dict, annotations : str, is_uniform_coords : bool, x0 : float, y0 : float, dx : float, dy : float, xmin : float, xmax : float, ymin : float, ymax : float, xcoords : numpy.ndarray, ycoords : numpy.ndarray, title : str, xlabel : str, xunit : str, ylabel : str, yunit : str, zlabel : str, zunit : str, autoscale : bool, xscalelog : bool, xscalemin : float, xscalemax : float, yscalelog : bool, yscalemin : float, yscalemax : float, zscalemin : float, zscalemax : float*) → sigima.objects.image.object.ImageObj

Returns a new instance of `ImageObj` with the fields set to the given values.

Paramètres

- **data** (*numpy.ndarray*) – Données. Default : None.
- **metadata** (*dict*) – Métadonnées. Default : {}.
- **annotations** (*str*) – Chaîne. Default : “”.
- **is_uniform_coords** (*bool*) – Default : True.
- **x0** (*float*) – X_0. Flottant. Default : 0.0.
- **y0** (*float*) – Y_0. Flottant. Default : 0.0.
- **dx** (*float*) – x. Flottant. Default : 1.0.
- **dy** (*float*) – y. Flottant. Default : 1.0.
- **xmin** (*float*) – X_{MIN}. Flottant. Default : None.
- **xmax** (*float*) – X_{MAX}. Flottant. Default : None.
- **ymin** (*float*) – Y_{MIN}. Flottant. Default : None.
- **ymax** (*float*) – Y_{MAX}. Flottant. Default : None.
- **xcoords** (*numpy.ndarray*) – Coordonnées X. Default : array([], dtype=float64).
- **ycoords** (*numpy.ndarray*) – Coordonnées Y. Default : array([], dtype=float64).
- **title** (*str*) – Titre de l’image. Chaîne. Default : “Sans titre”.
- **xlabel** (*str*) – Titre. Chaîne. Default : “”.
- **xunit** (*str*) – Unité. Chaîne. Default : “”.

— **ylabel** (*str*) – Titre. Chaîne. Default : “”.
— **yunit** (*str*) – Unité. Chaîne. Default : “”.
— **zlabel** (*str*) – Titre. Chaîne. Default : “”.
— **zunit** (*str*) – Unité. Chaîne. Default : “”.
— **autoscale** (*bool*) – Default : True.
— **xscalelog** (*bool*) – Default : False.
— **xscalemin** (*float*) – Borne inférieure. Flottant. Default : None.
— **xscalemax** (*float*) – Borne supérieure. Flottant. Default : None.
— **yscalelog** (*bool*) – Default : False.
— **yscalemin** (*float*) – Borne inférieure. Flottant. Default : None.
— **yscalemax** (*float*) – Borne supérieure. Flottant. Default : None.
— **zscalemin** (*float*) – Borne inférieure. Flottant. Default : None.
— **zscalemax** (*float*) – Borne supérieure. Flottant. Default : None.

Renvoie
New instance of `ImageObj`.

IO_REGISTRY

alias de `ImageIORegistry`

static get_roi_class() → Type[ImageROI]

Return ROI class

static get_roieditor_class() → Type[*ImageROIEditor*]

Return ROI editor class

plot_lut_changed(*plot : BasePlot*) → None

The LUT of the plot has changed : updating image objects accordingly

Paramètres
plot – Plot object

get_newparam_from_current(*newparam : NewImageParam | None = None, title : str | None = None*) → NewImageParam | None

Get new object parameters from the current object.

Paramètres
— **newparam** (*guidata.dataset.DataSet*) – new object parameters. If None, create a new one.
— **title** – new object title. If None, use the current object title, or the default title.

Renvoie
New object parameters

new_object(*param : NewImageParam | None = None, edit : bool = False, add_to_panel : bool = True*) → ImageObj | None

Create a new object (image).

Paramètres
— **param** (*guidata.dataset.DataSet*) – new object parameters
— **edit** (*bool*) – Open a dialog box to edit parameters (default : False). When False, the object is created with default parameters and creation parameters are stored in metadata for interactive editing.
— **add_to_panel** (*bool*) – Add the object to the panel (default : True)

Renvoie
New object

toggle_show_contrast(*state : bool*) → None

Toggle show contrast option

Macro panel

class datalab.gui.panel.macro.**MacroTabs**(*parent=None*)

Macro tabwidget

Paramètres
parent (*QWidget*) – Parent widget

clear() → None

Override Qt method

contextMenuEvent(*event*)

Override Qt method

add_tab(*macro : Macro*) → int

Add tab

Paramètres
macro – Macro object

Renvoie
Number of the tab (starting at 1)

Type renvoyé
int

remove_tab(*number : int*) → None

Remove tab

Paramètres
number – Number of the tab (starting at 1)

get_widget(*number : int*) → CodeEditor

Return macro editor widget at number

Paramètres
number – Number of the tab (starting at 1)

Renvoie
Macro editor widget

set_current_number(*number : int*) → None

Set current tab number

Paramètres
number – Number of the tab (starting at 1)

get_current_number() → int

Return current tab number

Renvoie
Number of the tab (starting at 1)

Type renvoyé
int

set_tab_title(*number : int*, *name : str*) → None

Set tab title

Paramètres
— **number** – Number of the tab (starting at 1)
— **name** – Macro name

class datalab.gui.panel.macro.**MacroPanel**(*parent : QMainWindow*)

Macro Panel widget

Paramètres
parent (*QWidget*) – Parent widget

update_color_mode() → None

Update color mode according to the current theme

get_serializable_name(*obj : Macro*) → str

Return serializable name of object

serialize_to_hdf5(*writer : NativeH5Writer*) → None

Serialize whole panel to a HDF5 file

Paramètres
writer – HDF5 writer

deserialize_from_hdf5(*reader : NativeH5Reader*, *reset_all : bool = False*) → None

Deserialize whole panel from a HDF5 file

Paramètres
— **reader** – HDF5 reader
— **reset_all** – If True, preserve original UUIDs (workspace reload). If False, regenerate UUIDs (importing objects).

create_object() → Macro

Create object.

Renvoie
Macro object

add_object(*obj : Macro*) → None

Add object.

Paramètres
obj – Macro object

remove_all_objects() → None

Remove all objects

setup_actions() → None

Setup macro menu actions

get_macro(*number_or_title : int | str | None = None*) → Macro | None

Return macro at number (if number is None, return current macro)

Paramètres
number – Number of the macro (starting at 1) or title of the macro. Defaults to None (current macro).

Renvoie
Macro object or None (if not found)

get_number_from_title(*title : str*) → int | None

Return macro number from title

Paramètres
title – Title of the macro

Renvoie
Number of the macro (starting at 1) or None (if not found)

get_number_from_macro(*macro : Macro*) → int | None

Return macro number from macro object

Paramètres
macro – Macro object

Renvoie
Number of the macro (starting at 1) or None (if not found)

get_macro_titles() → list[str]

Return list of macro titles

macro_contents_changed() → None

One of the macro contents has changed

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

Run current macro

Paramètres
number – Number of the macro (starting at 1). Defaults to None (run current macro, or does nothing if there is no macro).

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

Stop current macro

Paramètres
number – Number of the macro (starting at 1). Defaults to None (run current macro, or does nothing if there is no macro).

macro_state_changed(*orig_macro : Macro, state : bool*) → None

Macro state has changed (True : started, False : stopped)

Paramètres
— **orig_macro** – Macro object
— **state** – State of the macro

add_macro() → Macro

Add macro, optionally with name

Renvoie
Macro object

macro_name_changed(*name : str*) → None

Macro name has been changed

Paramètres
name – New name of the macro

rename_macro(*number : int | None = None, title : str | None = None*) → None

Rename macro

Paramètres
— **number** – Number of the macro (starting at 1). Defaults to None.
— **title** – Title of the macro. Defaults to None.

export_macro_to_file(*number_or_title : int | str | None = None, filename : str | None = None*) → None

Export macro to file

Paramètres
— **number_or_title** – Number of the macro (starting at 1) or title of the macro. Defaults to None.
— **filename** – Filename. Defaults to None.

Lève
`ValueError` – If title is not found

`import_macro_from_file`(*filename : str | None = None*) → int

Import macro from file

Paramètres
`filename` – Filename. Defaults to None.

Renvoie
Number of the macro (starting at 1)

`remove_macro`(*number_or_title : int | str | None = None*) → None

Remove macro

Paramètres
`number_or_title` – Number of the macro (starting at 1) or title of the macro. Defaults to None.

Action handler

The `datalab.gui.actionhandler` module handles all application actions (menus, toolbars, context menu). These actions point to DataLab panels, processors, objecthandler, …

Utility classes

class datalab.gui.actionhandler.**SelectCond**

Signal or image select conditions

static always(*selected_groups : list[ObjectGroup], selected_objects : list[SignalObj | ImageObj]*) → bool

Always true

static exactly_one(*selected_groups : list[ObjectGroup], selected_objects : list[SignalObj | ImageObj]*) → bool

Exactly one signal or image is selected

static exactly_one_group(*selected_groups : list[ObjectGroup], selected_objects : list[SignalObj | ImageObj]*) → bool

Exactly one group is selected

static exactly_one_group_or_one_object(*selected_groups : list[ObjectGroup], selected_objects : list[SignalObj | ImageObj]*) → bool

Exactly one group or one signal or image is selected

static at_least_one_group_or_one_object(*sel_groups : list[ObjectGroup], sel_objects : list[SignalObj | ImageObj]*) → bool

At least one group or one signal or image is selected

static at_least_one(*sel_groups : list[ObjectGroup], sel_objects : list[SignalObj | ImageObj]*) → bool

At least one signal or image is selected

static at_least_two(*sel_groups : list[ObjectGroup], sel_objects : list[SignalObj | ImageObj]*) → bool

At least two signals or images are selected

static with_roi(*selected_groups : list[ObjectGroup], selected_objects : list[SignalObj | ImageObj]*) → bool

At least one signal or image has a ROI

static exactly_one_with_roi(*selected_groups : list[ObjectGroup], selected_objects : list[SignalObj | ImageObj]*) → bool

Exactly one signal or image has a ROI

static exactly_one_with_annotations(*selected_groups : list[ObjectGroup], selected_objects : list[SignalObj | ImageObj]*) → bool

Exactly one signal or image has annotations

static with_annotations(*selected_groups : list[ObjectGroup], selected_objects : list[SignalObj | ImageObj]*) → bool

At least one signal or image has annotations

static with_results(*selected_groups : list[ObjectGroup], selected_objects : list[SignalObj | ImageObj]*) → bool

At least one signal or image has results

class datalab.gui.actionhandler.**ActionCategory**(*value*)

Action categories

Handler classes

class datalab.gui.actionhandler.**SignalActionHandler**(*panel : SignalPanel | ImagePanel, panel_toolbar : QW.QToolBar, view_toolbar : QW.QToolBar*)

Object handling signal panel GUI interactions : actions, menus, …

create_first_actions()

Create actions that are added to the menus in the first place

create_last_actions()

Create actions that are added to the menus in the end

action_for(*function_or_name : Callable | str, position : int | None = None, separator : bool = False, context_menu_pos : int | None = None, context_menu_sep : bool = False, toolbar_pos : int | None = None, toolbar_sep : bool = False, toolbar_category : ActionCategory | None = None*) → QW.QAction

Create action for a feature.

Paramètres

- **function_or_name** – function or name of the feature
- **position** – add action to menu at this position. Defaults to None.
- **separator** – add separator before action in menu
- **context_menu_pos** – add action to context menu at this position.
- **context_menu_pos** – add action to context menu at this position. Defaults to None.
- **context_menu_sep** – add separator before action in context menu (or after if context_menu_pos is positive). Defaults to False.
- **toolbar_pos** – add action to toolbar at this position. Defaults to None.
- **toolbar_sep** – add separator before action in toolbar (or after if toolbar_pos is positive). Defaults to False.
- **toolbar_category** – toolbar category. Defaults to None. If toolbar_pos is not None, this specifies the category of the toolbar. If None, defaults to ActionCategory.VIEW_TOOLBAR if the current category is ActionCategory.VIEW, else to ActionCategory.PANEL_TOOLBAR.

Renvoie

New action

add_action(*action : QW.QAction*, *select_condition : Callable | None = None*) → None

Add action to list of actions.

Paramètres

- **action** – action to add
- **select_condition** – condition to enable action. Defaults to None. If None, action is enabled if at least one object is selected.

add_to_action_list(*action : QW.QAction*, *category : ActionCategory | None = None*, *pos : int | None = None*, *sep : bool = False*) → None

Add action to list of actions.

Paramètres

- **action** – action to add
- **category** – action category. Defaults to None. If None, action is added to the current category.
- **pos** – add action to menu at this position. Defaults to None. If None, action is added at the end of the list.
- **sep** – add separator before action in menu (or after if pos is positive). Defaults to False.

create_all_actions()

Create all actions

has_annotations_in_clipboard(*selected_groups : list[ObjectGroup]*, *selected_objects : list[SignalObj | ImageObj]*) → bool

Check if annotations clipboard is not empty

has_metadata_in_clipboard(*selected_groups : list[ObjectGroup]*, *selected_objects : list[SignalObj | ImageObj]*) → bool

Check if metadata clipboard is not empty

new_action(*title : str*, *position : int | None = None*, *separator : bool = False*, *triggered : Callable | None = None*, *toggled : Callable | None = None*, *shortcut : QW.QShortcut | None = None*, *icon_name : str | None = None*, *tip : str | None = None*, *select_condition : Callable | str | None = None*, *context_menu_pos : int | None = None*, *context_menu_sep : bool = False*, *toolbar_pos : int | None = None*, *toolbar_sep : bool = False*, *toolbar_category : ActionCategory | None = None*) → QW.QAction

Create new action and add it to list of actions.

Paramètres

- **title** – action title
- **position** – add action to menu at this position. Defaults to None.
- **separator** – add separator before action in menu (or after if pos is positive). Defaults to False.
- **triggered** – triggered callback. Defaults to None.
- **toggled** – toggled callback. Defaults to None.
- **shortcut** – shortcut. Defaults to None.
- **icon_name** – icon name. Defaults to None.
- **tip** – tooltip. Defaults to None.
- **select_condition** – selection condition. Defaults to None. If str, must be the name of a method of SelectCond, i.e. one of « always », « exactly_one », « exactly_one_group », « at_least_one_group_or_one_object », « at_least_one », « at_least_two », « with_roi ».
- **context_menu_pos** – add action to context menu at this position. Defaults to None.
- **context_menu_sep** – add separator before action in context menu (or after if context_menu_pos is positive). Defaults to False.
- **toolbar_pos** – add action to toolbar at this position. Defaults to None.
- **toolbar_sep** – add separator before action in toolbar (or after if toolbar_pos is positive). Defaults to False.

— **toolbar_category** – toolbar category. Defaults to None. If toolbar_pos is not None, this specifies the category of the toolbar. If None, defaults to ActionCategory.VIEW_TOOLBAR if the current category is ActionCategory.VIEW, else to ActionCategory.PANEL_TOOLBAR.

Renvoie
New action

new_category(*category : ActionCategory*) → Generator[None, None, None]

Context manager for creating a new menu.

Paramètres
category – Action category

Yields
None

new_menu(*title : str, icon_name : str | None = None, store_ref : str | None = None*) → Generator[None, None, None]

Context manager for creating a new menu.

Paramètres
— **title** – Menu title
— **icon_name** – Menu icon name. Defaults to None.
— **store_ref** – Optional attribute name to store menu reference. Defaults to None.

Yields
None

property object_suffix: str

Object suffix (e.g. « sig » for signal, « ima » for image)

populate_results_delete_submenu() → None

Populate the Results Delete submenu dynamically based on current selection

populate_roi_remove_submenu() → None

Populate the ROI Remove submenu dynamically based on current selection

selected_objects_changed(*selected_groups : list[ObjectGroup], selected_objects : list[SignalObj | ImageObj]*) → None

Update actions based on selected objects.

Paramètres
— **selected_groups** – selected groups
— **selected_objects** – selected objects

class datalab.gui.actionhandler.**ImageActionHandler**(*panel : SignalPanel | ImagePanel, panel_toolbar : QW.QToolBar, view_toolbar : QW.QToolBar*)

Object handling image panel GUI interactions : actions, menus, …

create_first_actions()

Create actions that are added to the menus in the first place

create_last_actions()

Create actions that are added to the menus in the end

action_for(*function_or_name : Callable | str, position : int | None = None, separator : bool = False, context_menu_pos : int | None = None, context_menu_sep : bool = False, toolbar_pos : int | None = None, toolbar_sep : bool = False, toolbar_category : ActionCategory | None = None*) → QW.QAction

Create action for a feature.

Paramètres

- **function_or_name** – function or name of the feature
- **position** – add action to menu at this position. Defaults to None.
- **separator** – add separator before action in menu
- **context_menu_pos** – add action to context menu at this position.
- **context_menu_pos** – add action to context menu at this position. Defaults to None.
- **context_menu_sep** – add separator before action in context menu (or after if context_menu_pos is positive). Defaults to False.
- **toolbar_pos** – add action to toolbar at this position. Defaults to None.
- **toolbar_sep** – add separator before action in toolbar (or after if toolbar_pos is positive). Defaults to False.
- **toolbar_category** – toolbar category. Defaults to None. If toolbar_pos is not None, this specifies the category of the toolbar. If None, defaults to ActionCategory.VIEW_TOOLBAR if the current category is ActionCategory.VIEW, else to ActionCategory.PANEL_TOOLBAR.

Renvoie

New action

add_action(*action : QW.QAction*, *select_condition : Callable | None = None*) → None

Add action to list of actions.

Paramètres

- **action** – action to add
- **select_condition** – condition to enable action. Defaults to None. If None, action is enabled if at least one object is selected.

add_to_action_list(*action : QW.QAction*, *category : ActionCategory | None = None*, *pos : int | None = None*, *sep : bool = False*) → None

Add action to list of actions.

Paramètres

- **action** – action to add
- **category** – action category. Defaults to None. If None, action is added to the current category.
- **pos** – add action to menu at this position. Defaults to None. If None, action is added at the end of the list.
- **sep** – add separator before action in menu (or after if pos is positive). Defaults to False.

create_all_actions()

Create all actions

has_annotations_in_clipboard(*selected_groups : list[ObjectGroup]*, *selected_objects : list[SignalObj | ImageObj]*) → bool

Check if annotations clipboard is not empty

has_metadata_in_clipboard(*selected_groups : list[ObjectGroup]*, *selected_objects : list[SignalObj | ImageObj]*) → bool

Check if metadata clipboard is not empty

new_action(*title : str*, *position : int | None = None*, *separator : bool = False*, *triggered : Callable | None = None*, *toggled : Callable | None = None*, *shortcut : QW.QShortcut | None = None*, *icon_name : str | None = None*, *tip : str | None = None*, *select_condition : Callable | str | None = None*, *context_menu_pos : int | None = None*, *context_menu_sep : bool = False*, *toolbar_pos : int | None = None*, *toolbar_sep : bool = False*, *toolbar_category : ActionCategory | None = None*) → QW.QAction

Create new action and add it to list of actions.

Paramètres

— **title** – action title
— **position** – add action to menu at this position. Defaults to None.
— **separator** – add separator before action in menu (or after if pos is positive). Defaults to False.
— **triggered** – triggered callback. Defaults to None.
— **toggled** – toggled callback. Defaults to None.
— **shortcut** – shortcut. Defaults to None.
— **icon_name** – icon name. Defaults to None.
— **tip** – tooltip. Defaults to None.
— **select_condition** – selection condition. Defaults to None. If str, must be the name of a method of SelectCond, i.e. one of « always », « exactly_one », « exactly_one_group », « at_least_one_group_or_one_object », « at_least_one », « at_least_two », « with_roi ».
— **context_menu_pos** – add action to context menu at this position. Defaults to None.
— **context_menu_sep** – add separator before action in context menu (or after if context_menu_pos is positive). Defaults to False.
— **toolbar_pos** – add action to toolbar at this position. Defaults to None.
— **toolbar_sep** – add separator before action in toolbar (or after if toolbar_pos is positive). Defaults to False.
— **toolbar_category** – toolbar category. Defaults to None. If toolbar_pos is not None, this specifies the category of the toolbar. If None, defaults to ActionCategory.VIEW_TOOLBAR if the current category is ActionCategory.VIEW, else to ActionCategory.PANEL_TOOLBAR.

Renvoie
New action

new_category(*category : ActionCategory*) → Generator[None, None, None]

Context manager for creating a new menu.

Paramètres
category – Action category

Yields
None

new_menu(*title : str*, *icon_name : str | None = None*, *store_ref : str | None = None*) → Generator[None, None, None]

Context manager for creating a new menu.

Paramètres
— **title** – Menu title
— **icon_name** – Menu icon name. Defaults to None.
— **store_ref** – Optional attribute name to store menu reference. Defaults to None.

Yields
None

property object_suffix: str

Object suffix (e.g. « sig » for signal, « ima » for image)

populate_results_delete_submenu() → None

Populate the Results Delete submenu dynamically based on current selection

populate_roi_remove_submenu() → None

Populate the ROI Remove submenu dynamically based on current selection

selected_objects_changed(*selected_groups : list[ObjectGroup]*, *selected_objects : list[SignalObj | ImageObj]*) → None

Update actions based on selected objects.

Paramètres

— **selected_groups** – selected groups
— **selected_objects** – selected objects

Object view

The *datalab.gui.objectview* module provides widgets to display object (signal/image) trees.

Note : This module provides tree widgets to display signals, images and groups. It is important to note that, by design, the user can only select either individual signals/images or groups, but not both at the same time. This is an important design choice, as it allows to simplify the user experience, and to avoid potential confusion between the two types of selection.

Simple object tree

class datalab.gui.objectview.**SimpleObjectTree**(*parent : QW.QWidget*, *objmodel : ObjectModel*)

Base object handling panel list widget, object (sig/ima) lists

initialize_from(*sobjlist :* SimpleObjectTree) → None

Init from another SimpleObjectList, without making copies of objects

iter_items(*item : QW.QTreeWidgetItem* | *None* = *None*) → Iterator[QW.QTreeWidgetItem]

Recursively iterate over all items

get_item_from_id(*item_id*) → QTreeWidgetItem

Return QTreeWidgetItem from id (stored in item's data)

get_current_item_id(*object_only : bool* = *False*) → str | None

Return current item id

set_current_item_id(*uuid : str*, *extend : bool* = *False*) → None

Set current item by id

get_current_group_id() → str

Return current group ID

get_sel_group_items() → list[QTreeWidgetItem]

Return selected group items

get_sel_group_uuids() → list[str]

Return selected group uuids

get_sel_object_items() → list[QTreeWidgetItem]

Return selected object items

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

Return selected objects uuids.

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

Renvoie
List of selected objects uuids.

get_sel_objects(*include_groups : bool = False*) → list[SignalObj | ImageObj]

Return selected objects.

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

get_sel_groups() → list[ObjectGroup]

Return selected groups

populate_tree() → None

Populate tree with objects

update_tree() → None

Update tree

add_group_item(*group : ObjectGroup*) → None

Add group item

add_object_item(*obj : SignalObj | ImageObj, group_id : str, set_current : bool = True*) → None

Add item

update_item(*uuid : str*) → None

Update item

remove_item(*oid : str, refresh : bool = True*) → None

Remove item

item_double_clicked(*item : QTreeWidgetItem*) → None

Item was double-clicked : open a pop-up plot dialog

contextMenuEvent(*event : QContextMenuEvent*) → None

Override Qt method

Get object dialog

class datalab.gui.objectview.**GetObjectsDialog**(*parent : QW.QWidget, panel : BaseDataPanel, title : str, comment : str = '', nb_objects : int = 1, minimum_size : tuple[int, int] | None = None*)

Dialog box showing groups and objects (signals or images) to select one, or more.

Paramètres

- **parent** – parent widget
- **panel** – data panel
- **title** – dialog title
- **comment** – optional dialog comment
- **nb_objects** – number of objects to select (default : 1)
- **minimum_size** – minimum size (width, height)

get_selected_objects() → list[SignalObj | ImageObj]

Return selected objects

Object view

class datalab.gui.objectview.**ObjectView**(*parent :* BaseDataPanel, *objmodel : ObjectModel*)

Object handling panel list widget, object (sig/ima) lists

paintEvent(*event*)

Reimplement Qt method

dragEnterEvent(*event : QDragEnterEvent*) → None

Reimplement Qt method

dragLeaveEvent(*event : QDragLeaveEvent*) → None

Reimplement Qt method

dragMoveEvent(*event : QDragMoveEvent*) → None

Reimplement Qt method

get_all_group_uuids() → list[str]

Return all group uuids, in a list ordered by group position in the tree

get_all_object_uuids() → dict[str, list[str]]

Return all object uuids, in a dictionary that maps group uuids to the list of object uuids in each group, in the correct order

dropEvent(*event : QDropEvent*) → None

Reimplement Qt method

get_current_object() → SignalObj | ImageObj | None

Return current object

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

Set current object

item_selection_changed() → None

Refreshing the selection of objects and groups, emitting the SIG_SELECTION_CHANGED signal which triggers the update of the object properties panel, the plot items and the actions of the toolbar and menu bar.

This method is called when the user selects or deselects items in the tree. It is also called when the user clicks on an item that was already selected.

This method emits the SIG_SELECTION_CHANGED signal.

select_objects(*selection : list[SignalObj | ImageObj | int | str]*) → None

Select multiple objects

Paramètres

selection – list of objects, object numbers (1 to N) or object uuids

select_groups(*groups : list[ObjectGroup | int | str] | None = None*) → None

Select multiple groups

Paramètres

groups – list of groups, group numbers (1 to N), group names or None (select all groups). Defaults to None.

move_up()

Move selected objects/groups up

move_down()

Move selected objects/groups down

Plot handler

The `datalab.gui.plothandler` module provides plot handlers for signal and image panels, that is, classes handling *PlotPy* plot items for representing signals and images.

Signal plot handler

class datalab.gui.plothandler.**SignalPlotHandler**(*panel : BaseDataPanel*, *plotwidget : PlotWidget*)

Object handling signal plot items, plot dialogs, plot options

toggle_anti_aliasing(*state : bool*) → None

Toggle anti-aliasing

Paramètres
state – if True, enable anti-aliasing

get_plot_options() → PlotOptions

Return standard signal/image plot options

refresh_plot(*what : str*, *update_items : bool = True*, *force : bool = False*, *only_visible : bool = True*, *only_existing : bool = False*) → None

Refresh plot and configure datetime axis if needed.

This override adds automatic datetime axis configuration when at least one of the displayed signals has a datetime X-axis.

Paramètres
— **what** – string describing the objects to refresh
— **update_items** – if True, update the items
— **force** – if True, force refresh even if auto refresh is disabled
— **only_visible** – if True, only refresh visible items
— **only_existing** – if True, only refresh existing items

add_shapes(*oid : str*, *do_autoscale : bool = False*) → None

Add geometric shape items associated to computed results and annotations, for the object with the given uuid

cleanup_dataview() → None

Clean up data view

clear() → None

Clear plot items

get(*key : str*, *default : TypePlotItem | None = None*) → TypePlotItem | None

Return item associated to object uuid. If the key is not found, default is returned if given, otherwise None is returned.

get_existing_oids() → list[str]

Get existing object uuids.

Renvoie
List of object uuids that have a plot item associated to them.

get_obj_from_item(*item : TypePlotItem*) → TypeObj | None

Return object associated to plot item

Paramètres
item – plot item

Renvoie
Object associated to plot item

reduce_shown_oids(*oids : list[str]*) → list[str]

Reduce the number of shown objects to visible items only. The base implementation is to show only the first selected item if the option « Show first only » is enabled.

Paramètres
oids – list of object uuids

Renvoie
Reduced list of object uuids

refresh_all_shape_items() → None

Refresh all geometric shapes to apply new style parameters.

This method is called when shape/marker visualization settings are changed in the Settings dialog. It removes and recreates all shape items to apply the new parameters.

remove_all_shape_items() → None

Remove all geometric shapes associated to result items

remove_item(*oid : str*) → None

Remove plot item associated to object uuid

set_auto_refresh(*auto_refresh : bool*) → None

Set auto refresh mode.

Paramètres
auto_refresh – if True, refresh plot items automatically

set_show_first_only(*show_first_only : bool*) → None

Set show first only mode.

Paramètres
show_first_only – if True, show only the first selected item

toggle_result_label_visibility(*show : bool*) → None

Toggle the visibility of all merged result labels on the plot.

Paramètres
show – True to show the labels, False to hide them

update_resultproperty_from_plot_item(*item : LabelItem*) → None

Update result properties from merged label plot item.

When the merged results label is moved, update the stored position in the merged result adapter.

Paramètres
item – Merged results label item

Image plot handler

class datalab.gui.plothandler.**ImagePlotHandler**(*panel :* BaseDataPanel, *plotwidget : PlotWidget*)

Object handling image plot items, plot dialogs, plot options

reduce_shown_oids(*oids : list[str]*) → list[str]

Reduce the number of shown objects to visible items only. The base implementation is to show only the first selected item if the option « Show first only » is enabled.

Paramètres
oids – list of object uuids

Renvoie
Reduced list of object uuids

`refresh_plot`(*what : str*, *update_items : bool = True*, *force : bool = False*, *only_visible : bool = True*, *only_existing : bool = False*) → None

Refresh plot.

Paramètres
- **what** – string describing the objects to refresh. Valid values are « selected » (refresh the selected objects), « all » (refresh all objects), « existing » (refresh existing plot items), or an object uuid.
- **update_items** – if True, update the items. If False, only show the items (do not update them). Defaults to True.
- **force** – if True, force refresh even if auto refresh is disabled. Defaults to False.
- **only_visible** – if True, only refresh visible items. Defaults to True. Visible items are the ones that are not hidden by other items or the items except the first one if the option « Show first only » is enabled. This is useful for images, where the last image is the one that is shown. If False, all items are refreshed.
- **only_existing** – if True, only refresh existing items. Defaults to False. Existing items are the ones that have already been created and are associated to the object uuid. If False, create new items for the objects that do not have an item yet.

Lève
ValueError – if *what* is not a valid value

`cleanup_dataview`() → None

Clean up data view

`get_plot_options`() → PlotOptions

Return standard signal/image plot options

`add_shapes`(*oid : str*, *do_autoscale : bool = False*) → None

Add geometric shape items associated to computed results and annotations, for the object with the given uuid

`clear`() → None

Clear plot items

`get`(*key : str*, *default : TypePlotItem | None = None*) → TypePlotItem | None

Return item associated to object uuid. If the key is not found, default is returned if given, otherwise None is returned.

`get_existing_oids`() → list[str]

Get existing object uuids.

Renvoie
List of object uuids that have a plot item associated to them.

`get_obj_from_item`(*item : TypePlotItem*) → TypeObj | None

Return object associated to plot item

Paramètres
item – plot item

Renvoie
Object associated to plot item

`refresh_all_shape_items`() → None

Refresh all geometric shapes to apply new style parameters.

This method is called when shape/marker visualization settings are changed in the Settings dialog. It removes and recreates all shape items to apply the new parameters.

remove_all_shape_items() → None

Remove all geometric shapes associated to result items

remove_item(*oid : str*) → None

Remove plot item associated to object uuid

set_auto_refresh(*auto_refresh : bool*) → None

Set auto refresh mode.

Paramètres

auto_refresh – if True, refresh plot items automatically

set_show_first_only(*show_first_only : bool*) → None

Set show first only mode.

Paramètres

show_first_only – if True, show only the first selected item

toggle_result_label_visibility(*show : bool*) → None

Toggle the visibility of all merged result labels on the plot.

Paramètres

show – True to show the labels, False to hide them

update_resultproperty_from_plot_item(*item : LabelItem*) → None

Update result properties from merged label plot item.

When the merged results label is moved, update the stored position in the merged result adapter.

Paramètres

item – Merged results label item

ROI editor

The `datalab.gui.roieditor` module provides the ROI editor widgets for signals and images.

Signal ROI editor

class datalab.gui.roieditor.**SignalROIEditor**(*parent : QW.QWidget | None, obj : SignalObj | ImageObj, mode : Literal['apply', 'extract', 'define'] = 'apply', item : TypePlotItem | None = None, options : PlotOptions | dict[str, Any] | None = None, size : tuple[int, int] | None = None*)

Signal ROI Editor

Paramètres

- **parent** – Parent plot dialog
- **obj** – Object to edit (`sigima.objects.SignalObj` or `sigima.objects.ImageObj`)
- **mode** – Mode of operation, either « apply » (define ROI, then apply it to selected objects), « extract » (define ROI, then extract data from it), or « define » (define ROI without applying or extracting).
- **item** – Optional plot item to add to the plot (if None, a new item is created from the object)

get_obj_roi_class() → type[SignalROI]

Get object ROI class

add_tools_to_plot_dialog() → None

Add tools to plot dialog

manually_add_roi() → None

Manually add segment ROI

create_coordinate_based_roi_actions() → list[QAction]

Create coordinate-based ROI actions

Image ROI editor

class datalab.gui.roieditor.**ImageROIEditor**(*parent : QW.QWidget | None, obj : SignalObj | ImageObj, mode : Literal['apply', 'extract', 'define'] = 'apply', item : TypePlotItem | None = None, options : PlotOptions | dict[str, Any] | None = None, size : tuple[int, int] | None = None*)

Image ROI Editor

Paramètres

- **parent** – Parent plot dialog
- **obj** – Object to edit (`sigima.objects.SignalObj` or `sigima.objects.ImageObj`)
- **mode** – Mode of operation, either « apply » (define ROI, then apply it to selected objects), « extract » (define ROI, then extract data from it), or « define » (define ROI without applying or extracting).
- **item** – Optional plot item to add to the plot (if None, a new item is created from the object)

get_obj_roi_class() → type[ImageROI]

Get object ROI class

add_tools_to_plot_dialog() → None

Add tools to plot dialog

manually_add_roi(*roi_type : Literal['rectangle', 'circle', 'polygon']*) → None

Manually add image ROI

create_coordinate_based_roi_actions() → list[QAction]

Create coordinate-based ROI actions

setup_items() → None

Setup items

Processor

The *datalab.gui.processor* package provides the **processor objects** for signals and images.

Processor objects are the bridge between the computation modules (in `sigima.proc`) and the GUI modules (in *datalab.gui*). They are used to call the computation functions and to update the GUI from inside the data panel objects.

When implementing a processing feature in DataLab, the steps are usually the following :

- Add an action in the *datalab.gui.actionhandler* module to trigger the processing feature from the GUI (e.g. a menu item or a toolbar button).
- Implement the computation function in the `sigima.proc` module (that would eventually call the algorithm from the `sigima.tools` module).
- Implement the processor object method in this package to call the computation function and eventually update the GUI.

The processor objects are organized in submodules according to their purpose :

- `datalab.gui.processor.base` : Common processing features
- `datalab.gui.processor.signal` : Signal processing features
- `datalab.gui.processor.image` : Image processing features

Generic processing types

To support consistent processing workflows, the `BaseProcessor` class defines five generic processing methods, each corresponding to a fundamental input/output pattern. These methods are tightly integrated with the GUI logic : the input objects are taken from the current selection in the active panel (Signal or Image), and the output objects are automatically appended to the same panel.

Descriptions :

- `compute_1_to_1` : Applies an independent transformation to each selected object.
- `compute_1_to_0` : Runs an analysis or measurement producing metadata or scalar data.
- `compute_1_to_n` : Produces multiple output objects from a single input (e.g. ROI extraction).
- `compute_n_to_1` : Aggregates multiple objects into one (e.g. sum, average) ; supports pairwise mode.
- `compute_2_to_1` : Applies a binary operation with a second operand (object or constant) ; supports pairwise mode.

Method name	Signature	Multi-selection behavior
`compute_1_to_1`	1 object 1 object	k k
`compute_1_to_0`	1 object no object	k 0
`compute_1_to_n`	1 object n objects	k k·n
`compute_n_to_1`	n objects 1 object	n 1 n n (pairwise mode)
`compute_2_to_1`	1 object + 1 operand 1 object	k + 1 k n + n n (pairwise mode)

These methods are for internal or advanced use (e.g. plugin or macro authors) and will evolve without backward compatibility guarantees.

Future developments (such as a visual pipeline editor) may require generalizing this model to support additional sources and destinations beyond the current panel-based selection/output logic.

Docks

The *`datalab.gui.docks`* module provides the dockable widgets for the DataLab main window.

Plot widget

class `datalab.gui.docks.`**`DataLabPlotWidget`**(*plot_type : PlotType*)

DataLab PlotWidget

This class is a subclass of *plotpy.plot.PlotWidget* that provides a customized widget for DataLab, with a specific set of tools and a customized appearance.

Paramètres

`plot_type` – Plot type

`register_tools`() → None

Register the plotting tools according to the plot type

Dockable plot widget

class datalab.gui.docks.**DockablePlotWidget**(*parent : QWidget*, *plot_type : PlotType*)

Docked plotting widget

Paramètres
- **parent** – Parent widget
- **plot_type** – Plot type

setup_layout() → None

Setup layout

update_toolbar_position() → None

Update toolbar position

setup_plotwidget() → None

Setup plotting widget

update_color_mode() → None

Update plot widget styles according to application color mode

get_plot() → BasePlot

Return plot instance

update_watermark(*plot : BasePlot*) → None

Update watermark visibility

visibility_changed(*enable : bool*) → None

DockWidget visibility has changed

HDF5 I/O

The *datalab.gui.h5io* module provides the HDF5 file open/save into/from DataLab data model/main window.

class datalab.gui.h5io.**H5InputOutput**(*mainwindow : DLMainWindow*)

Object handling HDF5 file open/save into/from DataLab data model/main window

Paramètres
- **mainwindow** – Main window

save_file(*filename : str*) → None

Save all signals and images from DataLab model into a HDF5 file

open_file(*filename : str*, *import_all : bool*, *reset_all : bool*) → None

Open HDF5 file

import_files(*filenames : list[str]*, *import_all : bool*, *reset_all : bool*) → None

Import HDF5 files

import_dataset_from_file(*filename : str*, *dsetname : str*) → None

Import dataset from HDF5 file

CHAPITRE 3

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.

3.1 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

3.2 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

3.3 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

3.4 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.

3.4.1 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.

3.4.2 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 :
- ruff pour le formatage du code et l'analyse statique du code.
- pylint pour l'analyse statique du code.

ruff

Si vous utilisez Visual Studio Code, les paramètres du projet formateront automatiquement votre code avec *ruff* à l'enregistrement (vous pouvez également exécuter la tâche « Run Ruff » pour exécuter *ruff* sur le projet).

Pour exécuter *ruff*, exécutez la commande suivante :

```
ruff check
```

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.

3.4.3 Workflow Git

Ce document décrit le workflow Git utilisé dans le projet DataLab, basé sur une branche `main`, une branche `develop` et des branches spécifiques aux fonctionnalités. Il définit également comment les correctifs de bogues sont gérés.

Note : Ce workflow est une version simplifiée du Gitflow Workflow. Il a été adapté pour répondre aux besoins du projet DataLab à l'étape actuelle de développement. À l'avenir, nous pourrions envisager d'adopter un workflow plus complexe, par exemple en ajoutant des branches de publication.

Modèle de branches

Branches principales

- `main` : Représente la version stable et prête pour la production du projet.
- `develop` : Utilisé pour le développement continu et l'intégration de nouvelles fonctionnalités.

Branches de fonctionnalités

- `feature/feature_name` : Utilisé pour le développement de nouvelles fonctionnalités.
  - Créé à partir de `develop`.
  - Fusionné de nouveau dans `develop` une fois terminé.
  - Supprimé après la fusion.

Branches de correction d'anomalies

- `fix/xxx` : Utilisé pour les corrections générales d'anomalies qui ne sont pas urgentes.
  - Créé à partir de `develop`.
  - Fusionné de nouveau dans `develop` une fois terminé.
  - Supprimé après la fusion.
- `hotfix/xxx` : Utilisé pour les corrections urgentes critiques pour la production.
  - Créé à partir de `main`.
  - Fusionné de nouveau dans `main`.
  - La correction est ensuite sélectionnée dans `develop`.
  - Supprimé après la fusion.

Note : Les correctifs urgents (correctifs de haute priorité) seront intégrés dans la prochaine version de maintenance (X.Y.Z -> Z+1), tandis que les correctifs (correctifs de basse priorité) seront intégrés dans la prochaine version de fonctionnalité (X.Y -> Y+1).

Branches de documentation

Lors de la rédaction de documentation qui n'est pas liée au code source (par exemple, les supports de formation, les guides utilisateur), les branches doivent être nommées en utilisant le préfixe `doc/`.

Exemples :
- `doc/training-materials`
- `doc/user-guide`

Cette convention de nommage améliore la clarté en séparant clairement les efforts de documentation du développement lié au code (fonctionnalités, corrections, etc.).

Workflow pour les nouvelles fonctionnalités

1. Créer une nouvelle branche de fonctionnalité à partir de `develop` :

```
git checkout develop
git checkout -b develop/feature_name
```

2. Développer la fonctionnalité et valider les modifications.
3. Fusionner la branche de fonctionnalité de nouveau dans `develop` :

```
git checkout develop
git merge --no-ff develop/feature_name
```

4. Supprimer la branche de fonctionnalité :

```
git branch -d develop/feature_name
```

Avertissement : Ne pas laisser les branches de fonctionnalités non fusionnées trop longtemps. Les réorganiser régulièrement sur `develop` pour minimiser les conflits.

Workflow pour les corrections d'anomalies régulières

1. Créer une branche de correction d'anomalie à partir de `develop` :

```
git checkout develop
git checkout -b fix/bug_description
```

2. Appliquer la correction et valider les modifications.
3. Fusionner la branche de correction de nouveau dans `develop` :

```
git checkout develop
git merge --no-ff fix/bug_description
```

4. Supprimer la branche de correction :

```
git branch -d fix/bug_description
```

Avertissement : Ne pas créer une branche `fix/xxx` à partir d'une branche `develop/feature_name`. Toujours créer une branche à partir de `develop` pour garantir que les corrections sont correctement propagées.

```
# Incorrect:
git checkout develop/feature_name
git checkout -b fix/wrong_branch
# Correct:
git checkout develop
git checkout -b fix/correct_branch
```

Workflow pour les correctifs urgents

1. Créer une branche de correctif urgent à partir de `main` :

   ```
   git checkout main
   git checkout -b hotfix/critical_bug
   ```

2. Appliquer la correction et valider les modifications.
3. Fusionner le correctif de nouveau dans `main` :

   ```
   git checkout main
   git merge --no-ff hotfix/critical_bug
   ```

4. Faire un cherry-pick du correctif dans `develop` :

   ```
   git checkout develop
   git cherry-pick <commit_hash>
   ```

5. Supprimer la branche de correctif urgent :

   ```
   git branch -d hotfix/critical_bug
   ```

Avertissement : Ne pas fusionner `fix/xxx` ou `hotfix/xxx` directement dans `main` sans suivre le workflow. Assurez-vous que les correctifs urgents ont fait l'objet de cherry-pick dans `develop` pour éviter de perdre des correctifs dans les futures versions.

Bonnes pratiques

— Réorganiser régulièrement les branches de fonctionnalités sur `develop` pour rester à jour :

```
git checkout develop/feature_name
git rebase develop
```

— Éviter les branches de longue durée pour minimiser les conflits de fusion.
— S'assurer que les corrections d'anomalies dans `main` sont **font toujours l'objet d'un cherry-pick** dans `develop`.
— Différencier clairement entre `fix/xxx` (correctifs non urgents) et `hotfix/xxx` (correctifs critiques pour la production).

Conclusion

Ce workflow garantit un processus de développement structuré mais flexible tout en maintenant `main` stable et `develop` toujours à jour avec les dernières modifications.

Il garantit également que les corrections d'anomalies sont correctement gérées et propagées entre les branches.

3.4.4 A propos des dépendances

Les dépendances sont une partie importante de tout projet logiciel, et elles peuvent affecter considérablement le processus de développement. Dans cette section, nous allons discuter des dépendances utilisées dans notre projet, comment les gérer et les meilleures pratiques pour les maintenir à jour.

Gérer les dépendances

Les dépendances du projet DataLab sont définies dans le fichier *pyproject.toml*, comme recommandé officiellement par l'autorité de packaging Python (voir Writing your pyproject.toml).

« Le fichier *pyproject.toml* contient de nombreuses sections, mais les plus pertinentes pour les dépendances sont :

- *[project.dependencies]* : Cette section liste les dépendances directes de l'application. Ce sont les paquets dont l'application a besoin pour fonctionner.
- *[project.optional-dependencies]* : Cette section liste les dépendances optionnelles qui peuvent être installées pour activer des fonctionnalités supplémentaires. Ces dépendances ne sont pas nécessaires au fonctionnement de base de l'application, mais peuvent améliorer ses capacités.

Parmi les dépendances optionnelles, nous avons les groupes suivants :

- *qt* : interface utilisateur graphique basée sur Qt (actuellement, PyQt5).
- *opencv* : tâches de vision par ordinateur nécessitant OpenCV.
- *dev* : développement et tests (linters, formateurs, etc.).
- *doc* : construction de la documentation (Sphinx, etc.).
- *test* : exécution des tests (pytest, etc.).

Déployer les dépendances

En production

En production, nous recommandons d'utiliser le gestionnaire de paquets qui convient le mieux à votre environnement et avec lequel vous êtes le plus à l'aise. Tous les gestionnaires de paquets (par exemple, *pip*, *conda*) utilisent directement ou indirectement les informations du fichier *pyproject.toml* pour installer les dépendances.

Voir *Installation* pour plus d'informations sur la façon d'installer DataLab et ses dépendances.

En développement

En développement, vous pouvez également utiliser le fichier de texte des exigences pour faciliter l'installation des dépendances dans un environnement virtuel ou un conteneur.

Le fichier *requirements.txt* liste toutes les dépendances nécessaires au projet, y compris les dépendances directes et optionnelles. C'est la liste exacte des dépendances définies dans le fichier *pyproject.toml*, mais formatée pour être utilisée avec *pip* ou d'autres gestionnaires de paquets qui prennent en charge les fichiers de dépendances.

Note : Le fichier de dépendances est généré à partir du fichier *pyproject.toml* à l'aide d'un outil fourni par le package *guidata*.

```
python -m guidata.utils.genreqs txt  # to generate requirements.txt
```

Ajouter de nouvelles dépendances

Lors de l'ajout de nouvelles dépendances au projet, veuillez suivre ces règles :

1. Si c'est une dépendance directe, c'est-à-dire un paquet dont l'application a besoin pour fonctionner, ajoutez-le à la section *[project.dependencies]* dans le fichier *pyproject.toml*, et spécifiez la plage de versions compatible avec l'application, afin d'éviter les changements incompatibles.
2. Si c'est une dépendance optionnelle, c'est-à-dire un paquet qui peut être installé pour activer des fonctionnalités supplémentaires, ajoutez-le à la section appropriée dans la section *[project.optional-dependencies]* du fichier *pyproject.toml*.

— Pour les dépendances *dev*, sauf si c'est absolument nécessaire, ne spécifiez pas de plage de versions, car cela peut limiter la capacité à utiliser la dernière version du paquet.
— Pour les autres dépendances optionnelles, spécifiez la plage de versions compatible avec l'application, afin d'éviter les changements incompatibles.

Note : En d'autres termes, sauf pour les dépendances *dev*, spécifiez toujours une plage de versions compatible avec l'application.

3.4.5 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
gettext	Coloration syntaxique Gettext
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
Insert GUID	Insérer un GUID
XML Tools	Outils XML

Environnement Python

DataLab nécessite les éléments suivants :

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

Installation de tous les paquets requis :

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

Voir Installation pour plus de détails sur les versions de référence de Python et Qt.

If you are contributing to DataLab's processing features, you will need to work on both DataLab's main repository and sigima's repository. To install `sigima` in editable mode, use the following command :

```
pip install -e ../sigima  # Adjust the path to your local sigima repository
```

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
3.9	OK	3.9.10.0
3.10	OK	3.10.11.1
3.11	OK	3.11.5.0
3.12	OK	3.12.3.0
3.13	OK	3.13.2.0

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 « datalab\data\tests » (données publiques)

Un second dossier %DATALAB_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/ou `tasks.json` (exemples) :

```
@REM Folder containing additional working test data
set DATALAB_DATA=C:\Dev\Projets\DATALAB_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).

Windows installer

The Windows installer is built using WiX Toolset V4.0.5. Using the WiX Toolset requires .NET SDK V6.0 minimum.

You may install .NET SDK using `winget` :

```
winget install Microsoft.DotNet.SDK.8
```

Once .NET SDK is installed, the WiX Toolset can be installed and configured using the following commands :

```
dotnet tool install --global wix --version 4.0.5
wix extension add WixToolset.UI.wixext/4.0.5
```

First, you need to generate the installer script from a generic template :

```
python wix/makewxs.py
```

Building the installer is done using the following command :

```
wix build .\wix\DataLab.wxs -ext WixToolset.UI.wixext
```

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

3.4.6 Feuille de route

Ce document décrit les plans de développement actuels et futurs de DataLab :

- Il contient des informations sur les *sources de financement* et les *jalons futurs*.
- Il fournit également un résumé des *jalons passés* pour donner un contexte à l'évolution du projet.

Financement

> En tant que projet open-source, DataLab s'appuie sur le soutien de diverses organisations et subventions pour financer son développement et sa maintenance. La feuille de route du projet est façonnée par les besoins et les priorités de ses utilisateurs, ainsi que par la disponibilité des ressources. Le travail financé est priorisé et programmé en conséquence, tandis que les contributions de la communauté sont les bienvenues et encouragées.

Depuis le début du projet en 2023, le développement de DataLab a été financé par les subventions et organisations suivantes :

Financement	Description
	CEA - Commissariat à l'énergie atomique et aux énergies alternatives :• DataLab a été initialement créé pour analyser les données de la structure Laser Mégajoule (LMJ) du CEA• Interfacé avec le système de contrôle du LMJ, DataLab est utilisé pour traiter et visualiser les signaux et images acquis avec des diagnostics plasmas (appareils tels que caméras, numériseurs, spectromètres, etc.)• Il est également utilisé pour des activités de R&D autour de l'installation LMJ (métrologie, analyse de données, etc.)• Le CEA est le principal investisseur dans DataLab et le principal contributeur au projet : les ingénieurs du CEA participent activement à la feuille de route
	CODRA, société d'ingénierie en informatique et éditeur de logiciels, soutient le projet open source DataLab depuis sa création :• Gestion de projet open source et communication (réseaux sociaux, site web, etc.)• Conférences et événements : SciPy 2024, PyData Paris 2024, Open Source Experience 2024, etc.• Documentation : tutoriels, vidéos, et bien plus
	La Fondation NLnet, dans le cadre du NGI0 Commons Fund soutenu par la Commission européenne, a financé la refonte de l'architecture de base de DataLab - une refonte majeure prévue pour la version 2.0 (décembre 2025) :• L'objectif est de découpler le modèle de données, le calcul et l'E/S de l'interface utilisateur• Cela permettra à DataLab d'être utilisé comme bibliothèque dans d'autres logiciels

Jalons futurs

> Les tâches suivantes sont des objectifs à long terme pour DataLab. Elles ne sont pas programmées pour une version spécifique et peuvent évoluer au fil du temps en fonction des besoins des utilisateurs et de l'orientation du projet.

Le tableau suivant résume les évolutions futures et les plans de maintenance de DataLab qui sont détaillés dans les sections ci-dessous.

Type de jalon	Description
Evolutions futures	• Prise en charge de l'acquisition de données• Interface Web• Prise en charge des séries temporelles• Connecteurs de base de données• Plugin Jupyter pour l'analyse de données interactive• Plugin Spyder pour l'analyse de données interactive• Interface de noyau Jupyter pour DataLab
Maintenance	• Passer à gRPC pour le contrôle à distance• Abandonner la prise en charge de Qt5 et migrer vers Qt6
Autres tâches	• Créer un modèle de plugin DataLab

Prise en charge de l'acquisition de données

Ajouter la prise en charge de l'acquisition de données serait une avancée majeure pour faire de DataLab non seulement une plateforme de traitement des signaux et des images, mais aussi un **outil polyvalent pour les flux de travail expérimentaux en temps réel**. L'idée serait de permettre aux utilisateurs d'acquérir des données directement à partir de divers appareils matériels (caméras, numériseurs, spectromètres, etc.) **au sein même de DataLab**.

Une telle fonctionnalité permettrait une **intégration transparente entre l'acquisition et l'analyse**, permettant aux utilisateurs de traiter et de visualiser les données immédiatement après leur capture, sans avoir à changer d'outil ou à exporter/importer des fichiers.

Bien qu'aucun travail de conception formel n'ait encore été établi, une approche viable pourrait être de :

- Introduire une **nouvelle famille de plugins dédiée à l'acquisition de données**, suivant l'architecture modulaire de DataLab ;
- Définir une **API générique pour les plugins d'acquisition**, garantissant flexibilité et compatibilité entre les types de dispositifs ;
- **Tirer parti des solutions existantes** telles que PyMoDAQ, un cadre d'acquisition de données basé sur Python déjà compatible avec un large éventail d'instruments de laboratoire.

Une **collaboration potentielle avec l'équipe de développement de PyMoDAQ** pourrait être envisagée, pour bénéficier de leur écosystème et éviter de dupliquer les efforts. Cela aiderait également à favoriser l'interopérabilité et à promouvoir des normes ouvertes dans la communauté d'instrumentation scientifique Python.

Interface Web

À mesure que l'architecture modulaire de DataLab évolue, une étape naturelle consiste à fournir une **interface Web** pour compléter l'application de bureau existante.

Une interface Web permettrait aux utilisateurs de :
- Exécuter DataLab à distance (par exemple depuis un serveur) et y accéder via un navigateur ;
- Effectuer des tâches de traitement et de visualisation sans avoir besoin d'un environnement Python local ;
- Faciliter l'"**analyse de données collaborative**, le partage de sessions ou de résultats avec des collègues ;
- Intégrer JupyterHub, des tableaux de bord ou des outils de gestion de laboratoire pour une utilisation centralisée.

Cette interface pourrait être construite sur la base de la bibliothèque Sigima (qui a été récemment créée à partir de l'externalisation des fonctionnalités de traitement de DataLab), exposant ses fonctionnalités via une interface Web - en utilisant éventuellement des outils tels que **JupyterLab extensions**, **Panel** ou **Dash**, en fonction de la pile choisie.

Bien que toujours exploratoire, cette direction augmenterait l'"**accessibilité et la portabilité** de DataLab, en particulier dans les environnements académiques et industriels.

> L'interface Web ne doit pas remplacer l'application de bureau, mais plutôt la compléter en fournissant un mode d'accès différent. La philosophie de DataLab est de rester une **application locale** qui ne nécessite pas de serveur pour fonctionner, garantissant la confidentialité et la sécurité des données. Plus précisément, l'interface Web permettra aux utilisateurs d'exécuter DataLab sur un serveur et d'y accéder à distance - généralement sur un réseau local, mais ce ne sera pas une solution basée sur le cloud ou une première étape vers une application entièrement basée sur le cloud, ce qui irait à l'encontre de la philosophie du projet.

Prise en charge des séries temporelles

DataLab se concentre actuellement sur le traitement générique des signaux et des images, mais de nombreux cas d'utilisation - en particulier dans l'instrumentation scientifique et la physique expérimentale - impliquent des **données de séries temporelles**.

Ajouter une prise en charge dédiée des séries temporelles faciliterait :
- Prendre en charge des signaux avec des horodatages associés ou un échantillonnage non uniforme ;
- Effectuer un traitement et une visualisation sensibles au temps (alignement d'événements, filtrage basé sur le temps, rééchantillonnage) ;
- Intégrer des systèmes externes générant des données indexées dans le temps.

Cette fonctionnalité est tracée par l'"Issue #27, où les cas d'utilisation potentiels et les considérations de conception sont discutés.

Introduire une gestion robuste des séries temporelles élargirait l'applicabilité de DataLab dans des domaines tels que l'enregistrement de données, le contrôle lent et la surveillance en temps réel.

Connecteurs de base de données

Actuellement, DataLab fonctionne principalement sur des fichiers et des structures de données en mémoire. L'ajout de **connecteurs aux bases de données** étendrait considérablement ses capacités, en particulier pour les utilisateurs traitant de grands ensembles de données structurées ou historiques.

Cette fonctionnalité permettrait à DataLab de :
- **Interroger et charger des données** à partir de bases de données SQL ou NoSQL (PostgreSQL, SQLite, MongoDB, etc.) ;
- Prendre en charge des flux de travail pilotés par les métadonnées, où les expériences ou ensembles de données sont référencés à partir d'une base de données ;
- **Stocker les résultats d'analyse** ou les annotations dans une base de données pour la traçabilité et la reproductibilité ;

— Faciliter l'intégration dans les systèmes de gestion de l'information de laboratoire (LIMS) ou les infrastructures de données d'entreprise.

Le design pourrait inclure :

— Un **système basé sur des plugins** pour prendre en charge divers backends de base de données ;
— Une interface de configuration simple pour les paramètres de connexion et la gestion des requêtes ;
— Intégration avec pandas ou SQLAlchemy pour un échange de données flexible.

Cette évolution aiderait à combler le fossé entre **l'acquisition de données, l'analyse et le stockage à long terme**, permettant des flux de travail scientifiques plus robustes.

Plugin Jupyter pour l'analyse de données interactive

Bien que DataLab puisse déjà être contrôlé à distance depuis un notebook Jupyter - grâce à ses capacités de contrôle à distance existantes (voir *Contrôle à distance*) - l'expérience utilisateur pourrait être considérablement améliorée en développant un **plugin Jupyter dédié**.

Ce plugin offrirait une **intégration plus étroite** entre Jupyter et DataLab, offrant les fonctionnalités suivantes :

— Utiliser DataLab comme un **noyau Jupyter**, permettant un accès direct à ses capacités de traitement depuis un notebook ;
— **Afficher les résultats numériques de DataLab dans Jupyter**, et vice versa - par exemple, importer des résultats calculés dans un notebook Jupyter dans l'interface DataLab ;
— Autoriser la **manipulation interactive des données** : utiliser DataLab pour des opérations efficaces sur les signaux et les images, et Jupyter pour des routines d'analyse de données personnalisées ou faites maison ;
— Créer un pont entre les flux de travail de script et d'interface graphique, rendant DataLab plus attrayant pour les utilisateurs scientifiques familiers avec les environnements Jupyter.

Techniquement, ce plugin pourrait s'appuyer sur l'"**interface de noyau Jupyter** pour exposer les capacités de DataLab dans un contexte de programmation interactive.

Une telle intégration renforcerait le rôle de DataLab dans l'écosystème scientifique Python et faciliterait les flux de travail d'analyse pilotés par des notebooks reproductibles.

Plugin Spyder pour l'analyse de données interactive

Un plugin pour l'"IDE Spyder répondrait aux **mêmes cas d'utilisation que le plugin Jupyter**, offrant une intégration transparente entre DataLab et l'environnement interactif de Spyder.

Ce plugin permettrait aux utilisateurs de :

— Interagir avec DataLab directement depuis Spyder ;
— Visualiser ou envoyer des données entre l'interface DataLab et la console Spyder ;
— Utiliser les capacités de traitement de DataLab en temps réel tout en scriptant des flux de travail d'analyse personnalisés dans Spyder.

Comme pour l'intégration Jupyter, ce plugin pourrait être mis en œuvre en s'appuyant sur l'"**interface de noyau Jupyter** (voir *Contrôle à distance*), que Spyder prend déjà en charge en interne.

Un tel plugin améliorerait l'utilisabilité de DataLab pour les scientifiques et les ingénieurs qui préfèrent l'environnement de développement intégré de Spyder pour l'analyse exploratoire.

Interface de noyau Jupyter pour DataLab

Mettre en œuvre une interface de **noyau Jupyter** native pour DataLab fournirait un moyen plus intégré de l'utiliser à partir d'autres environnements tels que **Jupyter notebooks, Spyder ou Visual Studio Code**.

Cette interface permettrait :
- Contrôle direct de DataLab à partir d'outils tiers qui prennent en charge les noyaux Jupyter ;
- **Échange de données bidirectionnel** : par exemple, afficher les résultats de DataLab dans un notebook, ou visualiser les données générées par Jupyter dans l'interface DataLab ;
- Intégration plus étroite dans les flux de travail de script et les IDE au-delà d'un simple contrôle à distance.

Cependant, sur la base d'explorations initiales, la mise en œuvre d'un noyau Jupyter complet peut être **non triviale et potentiellement chronophage**. Étant donné que le contrôle à distance permet déjà la communication entre DataLab et Jupyter (voir *Contrôle à distance*), la valeur ajoutée d'une intégration complète du noyau doit être soigneusement évaluée par rapport à sa complexité et à son coût de maintenance.

Cette option reste ouverte à la discussion en fonction de la demande des utilisateurs et des ressources de développement.

Plan de maintenance

2026 : Passer à gRPC pour le contrôle à distance

DataLab s'appuie actuellement sur XML-RPC pour le contrôle à distance, ce qui peut devenir une limitation pour des cas d'utilisation plus avancés ou haute performance. Si le besoin se fait sentir pour un protocole de communication plus **efficace, robuste et extensible**, un passage à **gRPC** est envisagé.

Cette amélioration est tracée par l'"Issue #18.

2025 : Abandonner la prise en charge de Qt5 et migrer vers Qt6

Avec la **fin de vie de Qt5 prévue pour mi-2025**, DataLab migrera entièrement vers **Qt6**. Cette transition devrait être **simple**, grâce à :
- L'utilisation de la couche d'abstraction `qtpy` ;
- La bibliothèque `PlotPyStack` est déjà compatible avec Qt6.

Ce changement garantira la compatibilité avec les futures versions de Qt et les environnements Python modernes.

Autres tâches

Créer un modèle de plugin DataLab

Pour encourager les contributions de la communauté et faciliter le développement d'extensions, un **modèle pour créer des plugins DataLab** est prévu.

Ce modèle permettrait :
- Fournir une structure prête à l'emploi avec les meilleures pratiques ;
- Aider les nouveaux développeurs à comprendre rapidement l'architecture des plugins ;
- Promouvoir la cohérence et la modularité entre les plugins tiers.

Cette tâche est tracée par l'"Issue #26.

Jalons passés

De la version 0.9 à 1.0, DataLab a connu un développement et des améliorations significatifs. Le projet a évolué d'un simple outil d'analyse de données à une plateforme puissante pour le traitement et la visualisation des signaux et des images.

Ces améliorations ont été rendues possibles grâce au soutien des organisations suivantes (voir *Financement* pour plus de détails) :

- CEA : Commissariat à l'énergie atomique et aux énergies alternatives
- CODRA : société d'ingénierie en informatique et éditeur de logiciels
- NLnet : Fondation NLnet, dans le cadre du NGI0 Commons Fund

Le tableau suivant résume les principaux jalons passés de DataLab, y compris les dates de publication et une brève description des fonctionnalités ou améliorations introduites dans chaque version.

Jalon	Description
1.020	• Édition interactive des objets : modifier les paramètres de création/de traitement et recalculer les objets• Nouvelle création d'objets : générateurs de signaux/images paramétriques, bruit de Poisson, objets à valeurs complexes• Traitement amélioré : fonctionnalités d'impulsion, filtres idéaux, rééchantillonnage 2D, convolution, projections• ROI et annotations : copier/coller, importer/exporter, création de grille, logique ROI inverse• Visualisation : étiquettes de résultats fusionnées, prise en charge des dates et heures, optimisations des performances• Formats de fichiers : FT-Lab (.sig, .ima), fichiers texte coordonnés, prise en charge améliorée de HDF5• Coordonnées non uniformes et étalonnage polynomial• Améliorations de l'API publique pour la programmabilité
0.202	• Images ANDOR SIF : ajouter la prise en charge des images de fond• Éditeur de tableau : nouvelles fonctionnalités de copie, d'exportation et de collage• Analyse de Fourier : nouvelle fonctionnalité de remplissage nul pour les signaux et les images• Éditeur de ROI : l'outil de sélection de ROI est désormais actif par défaut• Traitement du signal : mode X-Y, fonctionnalités de recherche d'abscisse/ordonnée, largeur complète à y=…• API publique : ajouter les arguments `group_id` et `set_current` aux méthodes d'ajout
0.192	• Ouvrir tous les signaux ou images d'un dossier (récursivement)• Éditeur de ROI : ajouter des options pour créer des ROI à partir de coordonnées
0.182	• Nouveau mode d'opérande par paires (l'opération est effectuée sur chaque paire de signaux/images)• Nouvelle fonctionnalité de ROI polygonal• Prise en charge de Windows 7 SP1 à Windows 11 avec un seul installateur
0.172	• Introduire la prise en charge des ROI dans toutes les fonctionnalités de traitement• Ajouter des opérations arithmétiques sur les signaux et les images• Nouveau package Ubuntu pour une installation native sur Linux• Nouveau package Conda pour toutes les plateformes (Windows, Linux, MacOS)
0.162	• Nouveau processus de validation pour les fonctionnalités de signal et d'image• Ajouter la prise en charge des images binaires (déverrouillage de nouvelles fonctionnalités de traitement)
0.152	• Nouvel installateur MSI pour la version autonome sur Windows• Ajouter la prise en charge de fichiers texte/CSV volumineux (> 1 Go)• Ajouter une fonctionnalité de sous-échantillonnage automatique
0.142	• Navigateur HDF5 : prise en charge de plusieurs fichiers, informations détaillées sur les groupes et les attributs• Nouveau package Debian pour une installation native sur Linux
0.122	• Ajouter une visite et une fonctionnalité de démonstration• Ajouter des tutoriels à la documentation• Ajouter un assistant d'importation de fichiers texte
0.112	• Ajouter des fonctionnalités pour réorganiser les signaux et les images (par exemple, glisser-déposer)• Ajouter des fonctionnalités de convolution 1D, d'interpolation, de rééchantillonnage et de désajustement
0.102 12	• Développer un plugin DataLab très simple pour démontrer le système de plugins• Permettre de désactiver le rafraîchissement automatique lors de plusieurs étapes de traitement• Sérialiser les styles de courbe et d'image dans des fichiers HDF5• Améliorer la lisibilité des courbes
0.920 11	• Exécuter des calculs dans un processus séparé• Ajouter un système de plugin : API pour les extensions tierces• Ajouter un système de macro-commande• Ajouter un serveur XML-RPC pour permettre le contrôle à distance de DataLab

CHAPITRE 4

Versions

Voir la page de la feuille de route de DataLab pour les jalons futurs et passés.

4.1 DataLab Version 1.0.1

Cette version majeure représente une étape importante pour DataLab avec de nombreuses améliorations dans tous les domaines. Les changements sont organisés par catégorie pour une navigation plus facile.

> **Note** : La version 1.0.1 est une mise à jour corrective qui résout les limitations de l'installateur Windows dans la version 1.0.0, qui ne permettait pas d'installer DataLab V1.0 alors que les versions majeures antérieures (V0.20 et antérieures) étaient déjà installées, malgré la prise en charge de la cohabitation déjà prévue dans l'application. Cette version ne contient aucune nouvelle fonctionnalité ou modification fonctionnelle.

4.1.1 Interface utilisateur et flux de travail

Réorganisation des menus :

- **Nouveau menu « Création »** : La fonctionnalité de création d'objets a été séparée du menu « Fichier » et placée entre les menus « Fichier » et « Édition » pour une organisation plus claire
  - Toutes les actions « Fichier > Nouveau […] » ont été déplacées vers le menu « Création »
  - **Note de migration** : Les actions de création de signaux/images se trouvent désormais dans le nouveau menu « Création »
- **Nouveau menu « ROI »** : Menu dédié à la gestion des régions d'intérêt, placé entre les menus « Édition » et « Opérations »
- **Nouveau sous-menu « Annotations »** : Regroupement des opérations d'annotation dans le menu « Édition »
- **Nouveau sous-menu « Métadonnées »** : Regroupement de toutes les opérations de métadonnées dans le menu « Édition »

Édition interactive d'objets :

- **Création interactive d'objets** : Les paramètres de création peuvent être modifiés après la création de l'objet via le nouvel onglet « Création » dans le panneau des propriétés
  - Appliquer les modifications sans créer de nouveaux objets, en préservant le traitement ultérieur

  - Disponible pour les générateurs paramétriques (Gaussien, sinusoïdal, etc.)
- **Traitement interactif 1-vers-1** : Les paramètres de traitement peuvent être ajustés et réappliqués après le traitement initial
  - Mise à jour sur place des objets résultats avec les paramètres modifiés
  - Uniquement pour les opérations paramétriques (filtres, morphologie, etc.)
- **Fonction de recalcul** : Nouvelle action « Recalculer » (Ctrl+R) pour retraiter rapidement les objets avec les paramètres stockés
  - Fonctionne avec un ou plusieurs objets
  - Met à jour automatiquement les résultats lorsque les données sources changent
- **Mise à jour automatique de l'analyse ROI** : Les résultats d'analyse sont automatiquement recalculés lorsque le ROI est modifié
  - Fonctionne pour toutes les opérations d'analyse (statistiques, centroïde, etc.)
  - Recalcul silencieux en arrière-plan pour un retour immédiat
- **Sélectionner les objets source** : Naviguer vers les objets source utilisés dans le traitement via une nouvelle action du menu « Édition »
  - Gère tous les modèles de traitement (1-vers-1, 2-vers-1, n-vers-1)
  - Affiche des messages informatifs si les sources n'existent plus
- **Historique de traitement** : Nouvel onglet « Historique » affichant la lignée de traitement des objets
  - Affiche la chaîne de traitement complète depuis la création jusqu'à l'état actuel
  - Texte sélectionnable à des fins de documentation

Édition des propriétés de plusieurs objets :

- Appliquer les modifications de propriétés à plusieurs objets sélectionnés simultanément
- Seules les propriétés modifiées sont appliquées, préservant les paramètres individuels inchangés
- Cas d'utilisation typique : modification des limites de la LUT (bornes de l'échelle Z) ou de la carte des couleurs de plusieurs images à la fois, ou ajustement des propriétés d'affichage d'un groupe de signaux.

Améliorations de la taille des boîtes de dialogue :

- Les boîtes de dialogue de traitement se redimensionnent désormais intelligemment en fonction de la taille de la fenêtre principale
- Ne dépasse jamais les dimensions de la fenêtre principale pour une meilleure expérience utilisateur

Indicateur de la console interne :

- L'indicateur de la barre d'état affiche l'état de la console lorsqu'elle est masquée
- Devient rouge en cas d'erreurs/avertissements pour alerter les utilisateurs
- Cliquer pour ouvrir la console

4.1.2 Nouvelles fonctionnalités de création d'objets

Générateurs de signaux paramétriques :

- Chirp linéaire, fonction logistique, fonction de Planck
- Générer des signaux avec bruit de Poisson

Générateurs d'images paramétriques :

- **Damier** : Motif de calibration avec taille de carrés et valeurs claires/sombres configurables
- **Réseau sinusoïdal** : Test de réponse en fréquence avec fréquences spatiales X/Y indépendantes
- **Motif en anneau** : Anneaux circulaires concentriques pour l'analyse radiale et les tests PSF. Période d'anneau, largeur, position centrale et plage d'amplitude réglables.
- **Étoile de Siemens** : Test de résolution avec rayons radiaux et rayon configurable
- **Sinc 2D** : Fonction sinus cardinal pour la modélisation de la PSF et la simulation de motifs de diffraction. Les paramètres incluent l'amplitude, la position centrale, le facteur d'échelle (sigma) et le décalage DC.
- **Rampe 2D** : Nouveau générateur d'image de rampe
- **Bruit de Poisson** : Générer des images avec distribution de bruit de Poisson

Création de signaux/images à partir d'opérations :

- Créer un signal/image à valeurs complexes à partir des parties réelle et imaginaire
- Créer un signal/image à valeurs complexes à partir de la magnitude et de la phase (ceci clôture Issue #216)
- Extraire l'information de phase (argument) des signaux ou images complexes

4.1.3 Traitement et analyse de données

Traitement du signal :

- **Ajustement de courbes amélioré** : Estimation des paramètres considérablement améliorée pour tous les types de courbes (Gaussienne, Lorentzienne, Voigt, exponentielle, sinusoïdale, Planckienne, pics asymétriques, CDF)
  - Estimations initiales des paramètres plus intelligentes pour une convergence robuste
  - **Prise en charge des paramètres verrouillés** (nécessite PlotPy v2.8.0) :
- **Vérification de la compatibilité des tableaux X pour les calculs multi-signaux** :
  - Nouveau paramètre de configuration : Demander à l'utilisateur ou interpoler automatiquement
  - Empêche les résultats inattendus dus à des tableaux de signaux incompatibles
- **Améliorations du zero padding** : Prise en charge de l'ajout de zéros au début et à la fin, stratégie par défaut désormais « Prochaine puissance de 2 »
- **Filtre idéal dans le domaine fréquentiel** : « Filtre en brique » pour les signaux (ceci clôture Issue #215)
- **Bande passante à -3dB** : Améliorée pour prendre en charge la bande passante
- **Extraction des caractéristiques des impulsions** : Nouvelle fonctionnalité complète d'analyse des impulsions pour les signaux en escalier et carrés :
  - Reconnaissance automatique de la forme et détection de la polarité
  - Mesure l'amplitude, le temps de montée/descente, la FWHM, les paramètres temporels, les plages de référence
- **Filtres dans le domaine fréquentiel** : Déconvolution et filtre Gaussien (ceci clôture Issue #189, Issue #205)
- **Transformations de coordonnées** : Conversion en coordonnées cartésiennes/polaires
- **Mode X-Y** : Simule le mode X-Y d'un oscilloscope (trace un signal par rapport à un autre)
- **Opérations de recherche** : Première abscisse à y=…, ordonnée à x=…, largeur totale à y=…
- **Opération `1/x`** : Opération réciproque avec gestion des NaN pour les dénominateurs nuls

Traitement d'images :

- **Rééchantillonnage 2D** : Rééchantillonner les images vers de nouvelles grilles de coordonnées avec plusieurs méthodes d'interpolation (ceci clôture Issue #208)
- **Convolution** : Opération de convolution 2D
- **Effacer une zone** : Effacer les zones d'image définies par ROI (ceci clôture Issue #204)
- **Projections horizontales/verticales** : Somme des pixels le long des axes (ceci clôture Issue #209)
- **Calcul du centroïde amélioré** : Plus précis dans les cas difficiles (images tronquées/asymétriques) (voir Issue #251)
- **Retourner en diagonale** : Nouvelle transformation géométrique
- **Opération `1/x`** : Opération réciproque pour les images

Opérations inter-panneaux :

- **Conversion signaux vers image** : Combiner plusieurs signaux en images 2D
  - Deux modes d'orientation : en lignes (spectrogrammes) ou en colonnes (affichages en cascade)
  - Normalisation optionnelle (Z-score, Min-Max, Maximum)
  - Cas d'utilisation typiques : cartes de chaleur, spectrogrammes, visualisation de données multi-canaux

Fonctionnalités communes :

- **Écart-type** : Calculer sur plusieurs signaux/images (ceci clôture Issue #196)
- **Ajouter du bruit** : Ajouter du bruit gaussien, de Poisson ou uniforme (ceci clôture Issue #201)
- **Ajouter des métadonnées** : Ajouter des métadonnées personnalisées avec prise en charge des motifs (`{title}`, `{index}`, etc.) et conversion de type

4.1.4 Gestion des ROI et annotations

Fonctionnalités ROI :

- **Opérations de presse-papiers ROI** : Copier/coller les ROI entre objets
- **Importer/exporter des ROI** : Enregistrer/charger les ROI au format JSON
- **Suppression individuelle de ROI** : Supprimer les ROI sélectivement via le sous-menu « Supprimer »
- **Édition des titres de ROI** : Définir les titres pendant la création interactive et dans la boîte de dialogue de confirmation
- **Créer une grille de ROI** : Générer une grille de ROI avec lignes, colonnes et espacement configurables
  - Importer/exporter les configurations de grille
  - Aperçu avant création
- **Logique ROI inverse** : Sélectionner la zone en dehors des formes définies (images uniquement)
- **Création de ROI par coordonnées** : Saisie manuelle des coordonnées pour les ROI rectangulaires et circulaires
- **Édition de ROI multi-objets** : Éditer les ROI sur plusieurs objets simultanément

Fonctionnalités d'annotation :

- **Coller les annotations** : Coller les annotations du presse-papiers sur l'objet (ou les objets) sélectionné(s)
- **Éditer les annotations** : Boîte de dialogue d'éditeur interactif avec outils PlotPy
- **Exporter les annotations** : Enregistrer les annotations dans des fichiers JSON .dlabann
- **Supprimer les annotations** : Supprimer toutes les annotations de l'objet (ou des objets) sélectionné(s)
- **Annotations indépendantes des ROI** : Peuvent coexister sur le même objet

Détection avec création de ROI :

- Les 7 algorithmes de détection prennent désormais en charge la création automatique de ROI :
  - Détection de pics, forme de contour, détection de blobs (DOG/DOH/LOG/OpenCV), cercle de Hough
- **Choix de géométrie de ROI** : ROI rectangulaires ou circulaires
- **Détection de pics 2D** : Option pour choisir la géométrie de ROI (ceci clôture les exigences associées)

4.1.5 Visualisation et affichage

Performances et limites d'affichage :

- **Optimisations des performances pour les grands ensembles de résultats** : Ajout de limites d'affichage configurables pour éviter le gel de l'interface utilisateur avec de grands résultats d'analyse
  - `max_shapes_to_draw` (par défaut : 1 000), `max_cells_in_label` (par défaut : 100), `max_cols_in_label` (par défaut : 15)
  - Paramètres documentés avec implications sur les performances
- **Rendu de contours plus rapide** : Amélioration des performances de plus de 5x pour l'affichage des contours
- **Optimisation des performances de rendu des signaux** : Ajout d'un plafonnement intelligent de l'épaisseur de trait pour les grands ensembles de données
  - Nouveau paramètre : « Seuil de performance de l'épaisseur de trait » (par défaut : 1 000 points) dans les paramètres de visualisation des signaux
  - Empêche un ralentissement de 10x dû à la limitation du moteur raster de Qt

Visualisation des résultats :

- **Étiquettes de résultats fusionnées** : Tous les résultats d'analyse pour un signal ou une image sont désormais affichés dans une seule étiquette consolidée en lecture seule
  - Réduit l'encombrement visuel, mises à jour automatiques, résultats divisés horizontalement
- **Contrôle de visibilité de l'étiquette de résultats** : Basculer la visibilité via la case à cocher du panneau de propriétés
  - Visibilité par défaut configurable dans les paramètres
- **Création de groupe de résultats** : Toutes les opérations de tracé des résultats organisent désormais automatiquement les résultats dans un groupe « Résultats » dédié
- **Titres de résultats complets** : Les titres des signaux de résultats incluent désormais les identifiants des objets source
- **Suppression individuelle de résultats** : Supprimer les résultats sélectivement via le menu Analyse

- **Styles de formes/marqueurs personnalisables** : Quatre nouveaux boutons de configuration de style dans les paramètres
  - Fonctionnalités communes aux signaux et aux images :
  - Éditeur interactif avec options complètes
  - Configuration persistante avec rafraîchissement au changement

Fonctionnalité d'extraction de profil d'intensité :

- Cliquer directement sur les tracés de profils X/Y pour changer la direction d'extraction
- Pas besoin d'ouvrir la boîte de dialogue des paramètres pour les changements de direction
- Améliore l'efficacité du flux de travail (ceci clôture Issue #156)

Prise en charge des signaux DateTime :

- Détection automatique des dates dans les fichiers CSV
- **Formatage de l'axe datetime** : Horodatages lisibles sur l'axe X
- **Formats configurables** : Formats séparés pour les unités standard et sub-secondes
- Prend en charge diverses unités de temps (secondes, millisecondes, microsecondes, minutes, heures)
- Ceci clôture Issue #258

Paramètres :

- **Marges d'auto-échelle** : Marges configurables (0-50%) pour les tracés de signaux/images
- **Verrouiller le rapport d'aspect de l'image** : Option pour un rapport d'aspect 1 :1 (par défaut : utiliser la taille de pixel physique) (ceci clôture Issue #244)
- **Afficher la console en cas d'erreur** : Comportement configurable (par défaut : désactivé)

Paramètres d'étendue de l'image :

- Nouvelle boîte de groupe « Étendue » affichant les Xmin, Xmax, Ymin, Ymax calculés
- Calculé automatiquement à partir de l'origine, de l'espacement des pixels et des dimensions

4.1.6 Import/Export et gestion des fichiers

Prise en charge de nouveaux formats de fichiers :

- **Signaux et images FT-Lab** : Formats binaires CEA (.sig, .ima) (ceci clôture Issue #211)
- **Fichiers texte coordonnés** : Images à valeurs réelles et complexes avec images d'erreur (similaire au format Matris)
  - Gestion automatique des NaN, métadonnées avec unités et étiquettes

Prise en charge HDF5 améliorée :

- **Extensions de fichiers personnalisées** : Détection intelligente HDF5 par contenu (pas seulement l'extension)
  - Détection basée sur l'extension pour les boîtes de dialogue, basée sur le contenu pour le glisser-déposer
  - Option « Tous les fichiers (*) » dans les boîtes de dialogue de fichiers
- **Améliorations du navigateur HDF5** : Vue arboresc ente réduite par défaut pour une meilleure navigation
- **Options de nettoyage de l'espace de travail** : Comportement configurable avec option « Ignorer » (ceci clôture Issue #146).

Améliorations des fichiers texte :

- Amélioration de la gestion des délimiteurs dans la lecture CSV (signaux) pour mieux prendre en charge les fichiers avec divers séparateurs d'espaces.
- **Séparateur décimal local** : Prise en charge de la virgule comme séparateur décimal (ceci clôture Issue #124)
- **Tolérance aux erreurs d'encodage** : Ignorer les erreurs pour les fichiers avec des caractères spéciaux
- **Détection d'en-tête** : Détection et saut automatiques des en-têtes de données

Autres fonctionnalités d'E/S :

- **Enregistrer dans un répertoire** : Nouvelle fonctionnalité (ceci clôture Issue #227)
- **Ouvrir depuis un répertoire** : Ouvrir récursivement plusieurs fichiers avec prise en charge du glisser-déposer de dossiers
- **Ordre d'importation des fichiers** : Correction de l'ordre incohérent des fichiers entre les plateformes lors de l'importation d'images/signaux

4.1.7 Fonctionnalités avancées

Prise en charge des coordonnées non uniformes :

- Les images prennent désormais en charge l'espacement de pixels non uniforme
- Nouvelle fonctionnalité « Définir les coordonnées uniformes » dans le menu Traitement > Transformation des axes :
- Prend désormais en charge la calibration polynomiale jusqu'à l'ordre cubique pour tous les axes (X, Y, Z)
  - Pour les axes X et Y : Crée des coordonnées non uniformes pour représenter les transformations polynomiales
- Le format HDF5 préserve les informations de coordonnées non uniformes

Gestion des groupes :

- **ID courts spécifiques au panneau** : Préfixe `gs` pour les signaux, préfixe `gi` pour les images
  - Évite l'ambiguïté dans les opérations inter-panneaux
- Numérotation de groupe fixe pour les nouveaux groupes

Réinitialisation de la configuration de DataLab :

- **Dossiers spécifiques à la version** : Coexistence des versions majeures (`.DataLab_v1`, `.DataLab_v2`, etc.)
  - Permet à v0.x et v1.x de s'exécuter simultanément

API publique (locale ou distante) :

- **Amélioration de la gestion des métadonnées en ajoutant le contexte du nom de la fonction aux résultats** :
- Méthodes `add_group`, `add_signal`, `add_image` avec arguments `group_id` et `set_current`
- `get_object_uuids` avec filtre `group` optionnel
- Plusieurs améliorations de l'API pour une meilleure programmabilité

Infrastructure de traitement (pour les développeurs) :

- Ajout d'un nouveau décorateur `@computation_function` pour marquer et enregistrer les fonctions de calcul
- Renommé les fonctions de calcul pour supprimer le préfixe redondant « compute_ » (par exemple, `compute_fft fft`)
- Méthodes `BaseProcessor` refactorisées avec des noms clairs :
  - `compute_1_to_1`, `compute_1_to_0`, `compute_1_to_n`, `compute_n_to_1`, `compute_2_to_1`
- **Aucune compatibilité ascendante n'est maintenue** pour ces modifications internes (ceci clôture Issue #180)

4.1.8 Corrections de bugs :

Corrections de performances :

- **Passage entre images avec de nombreux résultats** : Amélioration spectaculaire (66s → <1ms) lors de la navigation entre des images avec des centaines de formes
- **Robustesse du nettoyage des graphiques** : Correction d'erreurs occasionnelles lors de la suppression des résultats d'analyse des graphiques
- **Correction critique** : Les éléments d'étiquette de résultat sont désormais correctement exclus du nettoyage pour éviter une suppression accidentelle

Gestion inter-panneaux et des groupes :

- **Groupes de calculs inter-panneaux** : Correction de l'organisation incohérente des groupes pour les opérations image vers signal
- **Ordre d'importation des fichiers** : Correction de l'ordre incohérent des fichiers entre les plateformes lors de l'importation d'images/signaux

Mises à jour de l'état des actions :

- Correction des états d'activation des actions ne se mettant pas à jour après les opérations d'annotation/métadonnées
- L'interface utilisateur reflète désormais immédiatement l'état actuel de l'objet

Gestion des résultats :

- Correction de la suppression des étiquettes de résultats pour supprimer définitivement les métadonnées associées
- Correction du nettoyage des métadonnées de paramètres de résultats en double

- **Corrections des coordonnées des résultats** : Correction des résultats décalés sur les images avec des ROI et une origine décalée (Issue #106)
- **Indices d'extraction de profil** : Correction des indices erronés avec ROI (Issue #107)

Corrections liées aux ROI :

- Correction de l'extraction de ROI multi-image ne sauvegardant pas la ROI dans le premier objet (Issue #120)
- Correction de l'AttributeError lors de l'extraction de plusieurs ROI sur une seule image avec plusieurs sélections (Issue #121)
- Correction des problèmes de rafraîchissement du masque (Issue #122, Issue #123)
- Correction de l'éditeur de ROI sur plusieurs signaux/images (Issue #135)
- Correction de l'effacement des ROI n'affectant que la première image (Issue #160)
- Correction de l'éditeur de ROI affichant la première au lieu de la dernière image (Issue #158)

Assistant d'importation de texte :

- Correction de la préservation des titres et unités définis par l'utilisateur (Issue #239)
- Correction de la préservation des types de données (Issue #240)
- Correction de la prise en charge du séparateur décimal virgule (Issue #186)
- Correction du problème de délimiteur final (Issue #238)

Ajustement amélioré pour tous les types de courbes :

- Amélioration de l'estimation initiale de la fréquence pour l'ajustement de courbe sinusoïdale.
- Correction du formatage d'affichage des paramètres pour les valeurs extrêmes
- Correction de la gestion des exceptions de calcul FWHM
- Correction du changement inattendu du style de marqueur de courbe (Issue #184)
- Correction du plantage brutal sur un signal nul avec l'outil de statistiques de courbe (Issue #233)

Gestion des images :

- Correction du rapport d'aspect ne se mettant pas à jour lors du changement d'images
- Correction des problèmes de dépaquetage des formes (Issue #246, Issue #247)
- Correction des colormaps non stockées dans les métadonnées (problème PlotPy v2.6.3+) (Issue #138)
- Correction du calcul de l'amplitude pour les types de données non entiers

Traitement du signal :

- Correction du calcul de l'axe x pour ifft1d lorsque shift=False (Issue #241)
- Correction du plantage du médian mobile sur Linux avec le mode miroir (Issue #117) - bug SciPy
- Correction du spectre de magnitude avec échelle logarithmique (Issue #169)
- Correction du mode d'opération par paires pour les fonctions asymétriques (Issue #157)

Analyse et résultats :

- Correction de la gestion des valeurs NaN dans les statistiques et la normalisation (Issue #141, Issue #152, Issue #153)
- Correction des résultats d'analyse conservés de l'original après le traitement (Issue #136)
- Correction des résultats en double lorsqu'aucune ROI n'est définie
- Correction du mode « Une courbe par titre de résultat » ignorant les ROI (Issue #132)
- Ajout de la validation des résultats pour les résultats de type tableau

Graphique et visualisation :

- Correction des tracés de profils ne se rafraîchissant pas lors du déplacement/redimensionnement (Issue #172) - correction PlotPy
- Correction de l'affichage du profil moyen vide en dehors de la zone de l'image (Issue #168) - correction PlotPy
- Correction de la ValueError d'extraction de profil moyen avec un rectangle surdimensionné (Issue #144)
- Désactivation de l'onglet générique « Axes » dans les boîtes de dialogue des paramètres

Autres corrections :

- Correction de la méthode proxy `add_object` ne prenant pas en charge les métadonnées (Issue #111)
- Correction des appels de méthode RemoteClient sans arguments optionnels (Issue #113)
- Correction du KeyError lors de la suppression d'un groupe après l'ouverture d'un HDF5 (Issue #116)
- Correction du KeyError dans « Afficher dans une nouvelle fenêtre » avec plusieurs images après l'ouverture d'un HDF5 (Issue #159)
- Correction de l'affichage des titres d'objets longs (Issue #128)
- Correction des titres de noms de fichiers affichant des chemins relatifs (Issue #165)
- Correction des noms de groupe inattendus dans « Ouvrir depuis un répertoire » (Issue #177)

- Correction de l'attente d'un groupe par dossier (Issue #163)
- Correction des fichiers non pris en charge dans le chargement récursif (Issue #164)

Fonctionnalités supprimées :

- Suppression de l'option « Utiliser la plage de LUT de l'image de référence » :
  - **Stratégie de migration** : Utilisez plutôt le nouvel éditeur de propriétés de sélection multiple :

4.1.9 Corrections de sécurité :

- **Correction de vulnérabilité de dépendance** : Correction de la vulnérabilité CVE-2023-4863 dans opencv-python-headless
  - Mise à jour de l'exigence minimale de 4.5.4.60 à 4.8.1.78
  - Voir l'"avis de sécurité DataLab pour plus de détails.

4.1.10 Autres changements :

- Dépendance `plotpy` mise à jour vers V2.8.0
- Dépendance `guidata` mise à jour vers V3.13.0
- Utilisation du nouvel utilitaire de traduction `guidata` basé sur `babel`
- Python 3.13 désormais pris en charge (via scikit-image V0.25)

4.2 DataLab Version 0.20.0

Nouvelles fonctionnalités et améliorations :

- Images ANDOR SIF :
  - Ajout de la prise en charge des images de fond dans les fichiers ANDOR SIF
  - Ceci clôture le ticket Issue #178 - Ajout de la prise en charge des fichiers ANDOR SIF avec image de fond
- Éditeur de tableau (résultats, données de signal et d'image, …)
  - Nouvelle fonctionnalité « Copier tout » dans la boîte de dialogue de l'éditeur de tableau, pour copier toutes les données dans le presse-papiers, y compris les en-têtes de ligne et de colonne
  - Nouvelle fonctionnalité « Exporter » dans la boîte de dialogue de l'éditeur de tableau, pour exporter les données dans un fichier CSV, y compris les en-têtes de ligne et de colonne
  - Nouvelle fonctionnalité « Coller » dans la boîte de dialogue de l'éditeur de tableau, pour coller les données du presse-papiers dans l'éditeur de tableau (cette fonctionnalité n'est pas disponible pour les données en lecture seule, telles que les résultats d'analyse)
  - Les fonctionnalités ci-dessus nécessitent guidata v3.9.0 ou une version ultérieure
  - Ceci clôture les tickets Issue #174, Issue #175 et Issue #176
- Fonctionnalités d'analyse de Fourier (menu « Traitement ») :
  - Nouvelle fonctionnalité « Complément de zéros »
  - Implémentation pour les signaux :
    - Choisissez une stratégie de complément de zéros (prochain multiple de 2, double la longueur, triple la longueur, longueur personnalisée)
    - Ou définissez manuellement la longueur de complément de zéros (si « Longueur personnalisée » est sélectionnée)
  - Implémentation pour les images :
    - Choisissez une stratégie de complément de zéros (prochain multiple de 2, prochain multiple de 64, longueur personnalisée)
    - Ou définissez manuellement les longueurs de ligne et de colonne de complément de zéros (si « Longueur personnalisée » est sélectionnée)
    - Définir la position du complément de zéros (en bas à droite, centré)
  - Ceci corrige l'"Issue #170 - Analyse de Fourier : ajouter la fonctionnalité de complément de zéros pour les signaux et les images

- Éditeur de région d'intérêt (ROI) :
  - Cela concerne la fonctionnalité « Éditer les régions d'intérêt » pour les signaux et les images
  - Nouveau comportement :
    - Signaux : l'outil de sélection de la plage ROI est désormais actif par défaut, et l'utilisateur peut sélectionner immédiatement la plage du signal à utiliser comme ROI
    - Images : l'outil de sélection de la ROI rectangulaire est désormais actif par défaut, et l'utilisateur peut sélectionner immédiatement la ROI rectangulaire à utiliser comme ROI
    - Ceci clôture le ticket Issue #154 - Éditeur de ROI : activer l'outil de sélection de ROI par défaut, afin que l'utilisateur puisse sélectionner immédiatement la zone à utiliser comme ROI
  - Ajout de l'outil « Sélectionner » à la barre d'outils de l'éditeur, pour permettre à l'utilisateur de basculer facilement entre les outils « Sélectionner » et « Dessiner » sans avoir à utiliser la barre d'outils de tracé en haut de la fenêtre
- Fonctionnalités de traitement du signal (menu « Traitement ») :
  - Nouvelle fonctionnalité « Mode X-Y » : cette fonctionnalité simule le comportement du mode X-Y d'un oscilloscope, c'est-à-dire qu'elle permet de tracer un signal en fonction d'un autre signal (par exemple X en fonction de Y)
  - Nouvelles fonctionnalités de recherche d'abscisse et d'ordonnée :
    - Fonctionnalité « Abscisse à y=… » : cette fonctionnalité permet de trouver la première valeur d'abscisse d'un signal à une valeur y donnée (par exemple, la valeur d'abscisse d'un signal à y=0)
    - Fonctionnalité « Ordonnée à x=… » : cette fonctionnalité permet de trouver la valeur d'ordonnée d'un signal à une valeur x donnée (par exemple, la valeur d'ordonnée d'un signal à x=0)
    - Chaque fonctionnalité a sa propre boîte de dialogue, qui permet de définir la valeur y ou x à utiliser pour la recherche avec un curseur ou une zone de texte
    - Ceci clôture les tickets Issue #125 et Issue #126
  - Nouvelle fonctionnalité de calcul de largeur à un y donné :
    - La fonctionnalité « Largeur à y=… » permet de calculer la largeur d'un signal à une valeur y donnée
    - Une boîte de dialogue spécifique permet de définir la valeur y à utiliser avec un curseur ou une zone de texte
    - Ceci clôture le ticket Issue #127
- API publique (locale ou distante) :
  - Ajout des arguments `group_id` et `set_current` aux méthodes `add_signal`, `add_image` et `add_object` :
    - Cela concerne les classes `LocalProxy`, `AbstractDLControl`, `RemoteClient`, `RemoteServer` et `DLMainWindow`
    - L'argument `group_id` permet de spécifier l'identifiant du groupe où le signal ou l'image doit être ajouté (s'il n'est pas spécifié, le signal ou l'image est ajouté au groupe actuel)
    - L'argument `set_current` permet de spécifier si le signal ou l'image doit être défini comme actuel après avoir été ajouté (la valeur par défaut est `True`)
    - Ceci corrige l'"Issue #151 - API publique : ajouter un mot-clé `group_id` à `add_signal` et `add_image`

4.3 DataLab Version 0.19.2

Corrections de bugs :

- Correction de l'"Issue #172 - Profils d'image : lors du déplacement/redimensionnement de l'image, les graphiques de profil ne sont pas actualisés (corrigé dans PlotPy v2.7.4)
- Correction de l'"Issue #173 - Spectre de phase : ajouter l'unité (degré) et la référence de fonction (`numpy.angle`) à la documentation
- Correction de l'"Issue #177 - Fonctionnalité « Ouvrir depuis le répertoire » : nom de groupe inattendu (un groupe nommé « . » est créé au lieu du nom du dossier racine)
- Correction de l'"Issue #169 - Analyse de signal / Fourier : la fonction de spectre de magnitude ne fonctionne pas comme prévu avec l'échelle logarithmique activée
- Correction de l'"Issue #168 - Visualisation du profil moyen : le profil vide est affiché lorsque la zone rectangulaire cible est en dehors de la zone de l'image (ceci a été corrigé en amont, dans PlotPy v2.7.4, et nécessite donc la

dernière version de PlotPy)

4.4 DataLab Version 0.19.1

Corrections de bugs :

- Mode d'opération pairwise :
  - Correction d'un comportement inattendu lors de l'utilisation du mode d'opération pairwise avec des fonctions qui prennent un seul second opérande (par exemple pour les images : différence, division, opérations arithmétiques et correction de champ plat)
  - Si un seul ensemble d'opérandes était sélectionné dans un seul groupe, un message d'avertissement était affiché « En mode pairwise, vous devez sélectionner des objets dans au moins deux groupes. « , ce qui est correct pour les fonctions qui sont symétriques (par exemple addition, multiplication, etc.), mais pas pour les fonctions qui ne le sont pas (par exemple différence, division, etc.).
  - Ceci est maintenant corrigé : le message d'avertissement n'est affiché que pour les fonctions qui sont symétriques (par exemple addition, multiplication, etc.).
  - Ceci corrige l'"Issue #157 - Mode d'opération pairwise : comportement inattendu avec des fonctions qui prennent un seul second opérande
- Correction de l'"Issue #152 - Ignorer les valeurs `nan` pour la normalisation d'image, la correction de champ plat, la correction de décalage et le calcul du centroïde
- Correction de l'"Issue #153 - Ignorer les valeurs `nan` pour la normalisation du signal et les calculs de statistiques (à la fois le résultat d'analyse et l'outil interactif)
- Correction de l'"Issue #158 - Lors de l'édition de la ROI d'une liste d'images, la première image de la sélection est affichée (au lieu de la dernière comme dans le panneau d'image)
- Correction de l'"Issue #159 - Lors de la sélection de plusieurs images juste après l'ouverture d'un fichier HDF5, la fonctionnalité « Afficher dans une nouvelle fenêtre » ne fonctionne pas (exception `KeyError`)
- Correction de l'"Issue #160 - Lors de la sélection de plusieurs images et de l'effacement de la ROI dans l'éditeur de ROI, seule la première image est affectée
- Correction de l'"Issue #161 - Rafraîchir les items d'image uniquement si nécessaire (lors de l'édition de la ROI, du collage/de la suppression de métadonnées)
- Correction de l'"Issue #162 - Afficher dans une nouvelle fenêtre : lors de l'affichage de plusieurs images, le panneau de liste d'items doit être visible
- Ceci corrige l'"Issue #163 - Ouvrir depuis le répertoire : un groupe par dossier attendu lors du chargement de plusieurs fichiers
- Ceci corrige l'"Issue #164 - Ouvrir depuis le répertoire : les fichiers non pris en charge doivent être ignorés lors du chargement de fichiers de manière récursive, pour éviter les boîtes de dialogue d'avertissement
- Correction de l'"Issue #165 - Lors de l'ouverture d'un fichier, le titre par défaut du signal/image doit être défini sur le nom du fichier, au lieu du chemin relatif au nom du fichier

4.5 DataLab Version 0.19.0

Nouvelles fonctionnalités et améliorations :

- Fonctionnalités d'opération sur les images (menu « Opérations ») :
  - Renommage du sous-menu « Rotation » en « Symétrie ou rotation »
  - Nouvelle fonctionnalité « Symétrie diagonale »
- Fonctionnalités de traitement du signal (menu « Traitement ») :
  - Nouvelle fonctionnalité « Convertir en coordonnées cartésiennes »
  - Nouvelle fonctionnalité « Convertir en coordonnées polaires »
- Fonctionnalités d'analyse du signal (menu « Analyse ») :
  - Renommage de « Valeurs X au min/max » en « Abscisse du minimum et du maximum »
  - Nouvelle fonctionnalité « Abscisse à y=… »
- Nouvelle fonctionnalité « Ouvrir depuis le répertoire » :

    - Cette fonctionnalité permet d'ouvrir plusieurs fichiers d'un répertoire à la fois, de manière récursive (seuls les fichiers avec les extensions prises en charge par le panneau actuel sont ouverts)
    - Ajout de l'action « Ouvrir depuis le répertoire » au menu « Fichier » pour les panneaux Signal et Image
    - Ajout de la prise en charge des dossiers lors du glisser-déposer de fichiers dans les panneaux Signal et Image
- Ajout de l'opération `1/x` au menu « Opérations » pour les panneaux Signal et Image :
    - Cette fonctionnalité repose sur la fonction `numpy.reciprocal`, et gère le cas où le dénominateur est zéro en interceptant les avertissements et en remplaçant les valeurs `np.inf` par des valeurs `np.nan`
    - Ajout de la méthode `compute_inverse` pour les processeurs d'image et de signal
    - Ceci corrige l'"Issue #143 - Nouvelle fonctionnalité : `1/x` pour les signaux et les images
- API publique (locale ou distante) :
    - Ajout de la méthode `add_group` avec les arguments `title` et `select` pour créer un nouveau groupe dans un panneau de données (par exemple, panneau Signal ou Image) et éventuellement le sélectionner après sa création :
        - Cette méthode a été ajoutée aux classes suivantes : `AbstractDLControl`, `BaseDataPanel` et `RemoteClient`
        - Cette fonctionnalité clôture les tickets suivants :
            - Issue #131 - `BaseDataPanel.add_group` : ajout de l'argument `select`
            - Issue #47 - Proxy distant / API publique : ajout de la méthode `add_group`
    - `AbstractDLControl.get_object_uuids` : ajout d'un argument optionnel `group` (identifiant, titre ou numéro du groupe) pour éventuellement filtrer les objets par groupe (ceci clôture le ticket Issue #130)
- Lors de l'ouverture d'un fichier HDF5, la boîte de dialogue de confirmation demandant si l'espace de travail actuel doit être effacé a une nouvelle réponse possible « Ignorer » :
    - Choisir « Ignorer » empêchera la boîte de dialogue de confirmation d'être affichée à nouveau, et choisira le paramètre actuel (c'est-à-dire effacer ou non l'espace de travail) pour toutes les ouvertures de fichiers suivantes
    - Ajout d'une nouvelle option « Effacer l'espace de travail avant de charger le fichier HDF5 » dans la boîte de dialogue « Paramètres », pour permettre à l'utilisateur de changer le paramètre actuel (c'est-à-dire effacer ou non l'espace de travail) pour toutes les ouvertures de fichiers suivantes
    - Ajout d'une nouvelle option « Demander avant d'effacer l'espace de travail » dans la boîte de dialogue « Paramètres », pour permettre à l'utilisateur de désactiver ou de réactiver la boîte de dialogue de confirmation demandant si l'espace de travail actuel doit être effacé lors de l'ouverture d'un fichier HDF5
    - Ceci clôture le ticket Issue #146 - Demander avant d'effacer l'espace de travail lors de l'ouverture d'un fichier HDF5 : ajouter l'option « Ignorer » pour empêcher la boîte de dialogue d'être affichée à nouveau
- Renommage du titre des objets et des groupes :
    - Suppression de la fonctionnalité « Renommer le groupe » du menu « Édition » et du menu contextuel
    - Ajout de la fonctionnalité « Renommer l'objet » au menu « Édition » et au menu contextuel, avec le raccourci F2, pour renommer le titre de l'objet ou du groupe sélectionné
    - Ceci corrige l'"Issue #148 - Renommer le titre du signal/image/groupe en appuyant sur F2
- Éditeur de région d'intérêt :
    - Regroupement des actions graphiques (nouvelle ROI rectangulaire, nouvelle ROI circulaire, nouvelle ROI polygonale) dans un seul menu « ROI graphique »
    - Ajout d'un nouveau menu « ROI par coordonnées » pour créer une ROI en utilisant la saisie manuelle des coordonnées :
        - Pour les signaux, la ROI est définie par les coordonnées de début et de fin
        - Pour les images :
            - La ROI rectangulaire est définie par les coordonnées du coin supérieur gauche et du coin inférieur droit
            - La ROI circulaire est définie par les coordonnées du centre et du rayon
            - La ROI polygonale n'est pas encore prise en charge
        - Ceci clôture le ticket Issue #145 - Éditeur de ROI : ajouter la saisie manuelle des coordonnées

Corrections de bugs :

- Correction de l'"Issue #141 - Analyse d'image : masque des valeurs `nan` lors du calcul des statistiques, par exemple
- Correction de l'"Issue #144 - Extraction du profil moyen : `ValueError` lorsque le rectangle de sélection est plus

grand que l'image

4.6 DataLab Version 0.18.2

Informations générales :

- Python 3.13 est désormais pris en charge, depuis la disponibilité de scikit-image V0.25 (voir Issue #104 - Python 3.13 : `KeyError: 'area_bbox'`)

Améliorations :

- Ajout de la nouvelle option « Conserver les résultats après le calcul » dans la section « Traitement » :
  - Avant ce changement, lors de l'application d'une fonction de traitement (par exemple un filtre, un seuil, etc.) sur un signal ou une image, les résultats de l'analyse étaient supprimés de l'objet
  - Cette nouvelle option permet de conserver les résultats de l'analyse après l'application d'une fonction de traitement sur un signal ou une image. Même si les résultats de l'analyse ne sont pas mis à jour, ils peuvent être pertinents dans certains cas d'utilisation (par exemple lors de l'utilisation de la fonction de détection de pics 2D sur une image, puis de l'application d'un filtre sur l'image, ou de la sommation de deux images, etc.)

Corrections de bugs :

- Correction de l'"Issue #138 - Les cartes de couleurs d'image n'étaient plus stockées dans les métadonnées (et sérialisées dans les fichiers HDF5) depuis PlotPy v2.6.3 (ce commit, en particulier : PlotPyStack/PlotPy@a37af8a)
- Correction de l'"Issue #137 - Opérations arithmétiques et interpolation de signal : la boîte de dialogue avec les paramètres n'est pas affichée
- Correction de l'"Issue #136 - Lors du traitement d'un signal ou d'une image, le résultat de l'analyse est conservé à partir de l'objet d'origine
  - Avant cette correction, lors du traitement d'un signal ou d'une image (par exemple lors de l'application d'un filtre, d'un seuil, etc.), le résultat de l'analyse était conservé à partir de l'objet d'origine, et n'était pas mis à jour avec les nouvelles données. Ainsi, le résultat de l'analyse n'était plus pertinent, et induisait l'utilisateur en erreur.
  - Ceci est maintenant corrigé : le résultat de l'analyse est maintenant supprimé lors du traitement d'un signal ou d'une image. Cependant, il n'est pas recalculé automatiquement, car il n'y a aucun moyen de savoir quel résultat de l'analyse doit être recalculé (par exemple, si l'utilisateur a appliqué un filtre, le FWHM doit-il être recalculé ?) - de plus, l'implémentation actuelle des fonctionnalités d'analyse ne permet pas de recalculer automatiquement les résultats de l'analyse lorsque les données sont modifiées. L'utilisateur doit recalculer manuellement les résultats de l'analyse si nécessaire.
- Correction de l'"Issue #132 - Tracer les résultats de l'analyse : le mode « Une courbe par titre de résultat » ignore les ROI
  - Avant cette correction, le mode « Une courbe par titre de résultat » ignorait les ROI, et traçait le résultat sélectionné pour tous les objets (signaux ou images) sans tenir compte de la ROI définie sur les objets
  - Ceci est maintenant corrigé : le mode « Une courbe par titre de résultat » prend désormais en compte la ROI définie sur les objets, et trace le résultat sélectionné pour chaque objet (signal ou image) et pour chaque ROI définie sur l'objet
- Correction de l'"Issue #128 - Prise en charge des longs titres d'objet dans les panneaux Signal et Image
- Correction de l'"Issue #133 - Supprimer des résultats d'analyse spécifiques du presse-papiers de métadonnées lors de l'opération de copie
- Correction de l'"Issue #135 - Permettre de modifier la ROI sur plusieurs signaux ou images à la fois
  - Avant cette correction, l'éditeur de ROI était désactivé lorsque plusieurs signaux ou images étaient sélectionnés
  - Ceci est maintenant corrigé : l'éditeur de ROI est maintenant activé lorsque plusieurs signaux ou images sont sélectionnés, et la ROI est appliquée à tous les signaux ou images sélectionnés (seule la ROI du premier signal ou image sélectionné est prise en compte)
  - Ce nouveau comportement est cohérent avec la fonctionnalité d'extraction de ROI, qui permet d'extraire la ROI sur plusieurs signaux ou images à la fois, en fonction de la ROI définie sur le premier signal ou image

sélectionné
- Fonctionnalité de ROI d'image :
  - Correction de l'"Issue #120 - Extraction de ROI sur plusieurs images : la ROI définie ne doit pas être enregistrée dans le premier objet sélectionné. Le choix de conception est de ne pas enregistrer la ROI définie ni dans le premier ni dans aucun des objets sélectionnés : la ROI est uniquement utilisée pour l'extraction, et n'est pas enregistrée dans aucun objet
  - Correction de l'"Issue #121 - `AttributeError` lors de l'extraction de plusieurs ROIs sur une seule image, si plus d'une image est sélectionnée
  - Correction de l'"Issue #122 - Les masques d'image ne sont pas actualisés lors de la suppression des métadonnées sauf pour l'image active
  - Correction de l'"Issue #123 - Les masques d'image ne sont pas actualisés lors du collage de métadonnées sur plusieurs images, sauf pour la dernière image
- Fichiers texte et CSV :
  - Améliorer la lecture des fichiers texte en détectant les en-têtes de données (en utilisant une liste d'en-têtes typiques des instruments scientifiques) et en permettant de sauter l'en-tête lors de la lecture du fichier
  - Ignorer les erreurs d'encodage lors de la lecture des fichiers dans la fonctionnalité d'ouverture et l'assistant d'importation, permettant ainsi de lire des fichiers avec des caractères spéciaux sans lever d'exception
  - Correction de l'"Issue #124 - Fichiers texte : prise en charge du séparateur décimal local (différent de `.`)
- Fonctionnalités d'analyse de signal : correction des résultats en double lorsqu'aucune ROI n'est définie
- Correction de l'"Issue #113 - L'appel à `RemoteClient.open_h5_files` (et `import_h5_file`) échoue sans passer les arguments facultatifs
- Correction de l'"Issue #116 - Exception `KeyError` lors de la tentative de suppression d'un groupe après l'ouverture d'un fichier HDF5

4.7 DataLab Version 0.18.1

Améliorations :
- Le calcul de la FWHM lève désormais une exception lorsqu'il y a moins de deux points trouvés avec la méthode du zéro-crossing
- Amélioration de la validation des résultats de type tableau en vérifiant le type de données du résultat

Corrections de bugs :
- Correction de l'"Issue #106 - Analyse : résultats décalés de coordonnées sur des images avec des ROIs et une origine décalée
- Correction de l'"Issue #107 - Mauvais indices lors de l'extraction d'un profil à partir d'une image avec une ROI
- Correction de l'"Issue #111 - La méthode `add_object` du proxy ne prend pas en charge les métadonnées du signal/image (par exemple la ROI)
- Plugin de données de test / « Créer une image gaussienne bruitée 2D » : correction du calcul de l'amplitude dans `datalab.tests.data.create_2d_random` pour les types de données non entiers

Documentation :
- Correction des séparateurs de chemin dans la documentation du répertoire des plugins
- Corrigé les descriptions des zones gauche et droite dans la documentation de l'espace de travail
- Lien de style Google mis à jour dans les directives de contribution
- Corrections de diverses traductions françaises dans la documentation

4.8 DataLab Version 0.18.0

Informations générales :

- PlotPy v2.7 est requis pour cette version.
- Prise en charge de Python 3.8 abandonnée.
- Python 3.13 n'est pas encore pris en charge, car certaines dépendances ne sont pas compatibles avec cette version.

Nouvelles fonctionnalités et améliorations :

- Nouvelle fonctionnalité de mode d'opération :
  - Ajout de la fonctionnalité « Mode d'opération » à l'onglet « Traitement » de la boîte de dialogue « Paramètres »
  - Cette fonctionnalité permet de choisir entre les modes d'opération « single » et « pairwise » pour toutes les opérations de base (addition, soustraction, multiplication, division, etc.) :
    - Mode « single » : mode d'opérande unique (mode par défaut : l'opération est effectuée sur chaque objet indépendamment)
    - Mode « pairwise » : mode d'opérande par paire (l'opération est effectuée sur chaque paire d'objets)
  - Cela s'applique à la fois aux signaux et aux images, et aux calculs prenant *N* entrées
  - Les calculs prenant *N* entrées sont ceux où :
    - *N(>=2)* objets en entrée donnent *N* objets en sortie
    - *N(>=1)* objet(s) + 1 objet en entrée donnent N objets en sortie
- Nouvelles fonctionnalités de ROI (Région d'intérêt) :
  - Nouvelle fonctionnalité de ROI polygonale
  - Refonte complète des interfaces utilisateur de l'éditeur de ROI, améliorant l'ergonomie et la cohérence avec le reste de l'application
  - Refactorisation interne majeure du système de ROI pour le rendre plus robuste (plus de tests) et plus facile à maintenir
- Implémentation de l'"Issue #102 - Lancer DataLab en utilisant `datalab` au lieu de `datalab`. Notez que la commande `datalab` est toujours disponible pour la compatibilité ascendante.
- Implémentation de l'"Issue #101 - Configuration : définir l'interpolation d'image par défaut sur l'anti-crénelage (`5` au lieu de `0` pour le plus proche). Ce changement est motivé par le fait qu'une amélioration des performances a été apportée dans PlotPy v2.7 sur Windows, ce qui permet d'utiliser l'interpolation par anti-crénelage par défaut sans impact significatif sur les performances.
- Implémentation de l'"Issue #100 - Utiliser le même programme d'installation et exécutable sur Windows 7 SP1, 8, 10, 11. Avant ce changement, un programme d'installation spécifique était requis pour Windows 7 SP1, en raison du fait que Python 3.9 et les versions ultérieures ne sont pas prises en charge sur cette plateforme. Une solution de contournement a été mise en œuvre pour faire fonctionner DataLab sur Windows 7 SP1 avec Python 3.9.

Corrections de bugs :

- Correction de l'"Issue #103 - `proxy.add_annotations_from_items` : la couleur de la forme de cercle semble être ignorée.

4.9 DataLab Version 0.17.1

PlotPy v2.6.2 est requis pour cette version.

Nouvelles fonctionnalités et améliorations :

- Vue Image :
  - Avant cette version, lors de la sélection d'un grand nombre d'images (par exemple lors de la sélection d'un groupe d'images), l'application était très lente car toutes les images étaient affichées dans la vue image, même si elles étaient toutes superposées sur la même image
  - La solution de contournement était d'activer l'option « Afficher uniquement le premier »

    - Maintenant, pour améliorer les performances, si plusieurs images sont sélectionnées, seule la dernière image de la sélection est affichée dans la vue image si cette dernière image n'a pas de transparence et si les autres images sont complètement recouvertes par cette dernière image
- Clarification : l'action « Afficher uniquement le premier » a été renommée en « Afficher uniquement le premier objet », et une nouvelle icône a été ajoutée à l'action
- API : ajout des propriétés `width` et `height` à la classe `ImageObj` (retourne la largeur et la hauteur de l'image en unités physiques)
- Lanceur Windows « start.pyw » : écriture d'un fichier journal « datalab_error.log » lorsqu'une exception se produit au démarrage

Corrections de bugs :

- Changer le thème de couleur met maintenant à jour correctement tous les composants de l'interface utilisateur de DataLab sans avoir besoin de redémarrer l'application

Autres changements :

- OpenCV est désormais une dépendance facultative :
    - Ce changement est motivé par le fait que le paquet conda OpenCV n'est pas maintenu sur Windows (au moins), ce qui entraîne une erreur lors de l'installation de DataLab avec conda
    - Lorsque OpenCV n'est pas installé, seule la fonctionnalité « Détection de blob OpenCV » ne fonctionnera pas, et un message d'avertissement sera affiché lors de la tentative d'utilisation de cette fonctionnalité

4.10 DataLab Version 0.17.0

PlotPy v2.6 est requis pour cette version.

Nouvelles fonctionnalités et améliorations :

- Le menu « Calcul » a été renommé en « Analyse » pour les panneaux Signal et Image, pour mieux refléter la nature des fonctionnalités de ce menu
- Les régions d'intérêt (ROIs) sont désormais prises en compte partout dans l'application là où cela a du sens, et pas seulement pour les anciennes fonctionnalités du menu « Calcul » (maintenant « Analyse »). Cela clôture l'"Issue #93. Si un signal ou une image a une ROI définie :
    - Les opérations sont effectuées uniquement sur la ROI (sauf si l'opération modifie la forme des données, ou la taille des pixels pour les images)
    - Les fonctionnalités de traitement sont effectuées uniquement sur la ROI (si le type de données de l'objet de destination est compatible avec le type de données de l'objet source, ce qui exclut le seuillage, par exemple)
    - Les fonctionnalités d'analyse sont effectuées uniquement sur la ROI, comme avant
- En conséquence du point précédent, et pour plus de clarté :
    - Les fonctionnalités « Modifier les régions d'intérêt » et « Supprimer toutes les régions d'intérêt » ont été déplacées du vieux menu « Calcul » (maintenant « Analyse ») vers le menu « Édition » où se trouvent toutes les fonctionnalités liées aux métadonnées
    - L'action « Modifier les régions d'intérêt » a été ajoutée aux barres d'outils verticales de la Vue Signal et Image (en deuxième position, après l'action « Voir dans une nouvelle fenêtre »)
- Suite à la correction des problèmes de conversion du type de données des images avec les opérations de base, une nouvelle fonctionnalité « Opération arithmétique » a été ajoutée au menu « Opérations » pour les panneaux Signal et Image. Cette fonctionnalité permet d'effectuer des opérations linéaires sur les signaux et les images, avec les opérations suivantes :
    - Addition : `obj3 = (obj1 + obj2) * a + b`
    - Soustraction : `obj3 = (obj1 - obj2) * a + b`
    - Multiplication : `obj3 = (obj1 * obj2) * a + b`
    - Division : `obj3 = (obj1 / obj2) * a + b`
- Amélioration de la gestion de la taille des boîtes de dialogue « Voir dans une nouvelle fenêtre » et « Éditeur de ROI » : la taille par défaut ne sera pas plus grande que la taille de la fenêtre principale de DataLab
- Éditeur de ROI :
    - Ajout de barres d'outils pour les éditeurs de ROI Signal et Image, pour permettre de zoomer et dézoomer, et de réinitialiser facilement le niveau de zoom

    - Réorganisation des boutons dans la boîte de dialogue de l'éditeur de ROI pour une meilleure ergonomie et une cohérence avec l'éditeur d'annotations (boîte de dialogue « Voir dans une nouvelle fenêtre »)
  - Application de thème de couleur :
    - Ajout de la prise en charge du thème de couleur (auto, clair, foncé) dans la boîte de dialogue « Paramètres »
    - Le thème de couleur est appliqué sans redémarrer l'application

Corrections de bugs :

- Extraction de profil d'intensité / Profil de segment :
  - Lors de l'extraction d'un profil sur une image avec une ROI définie, la fonction PlotPy associée affiche un message d'avertissement ("UserWarning : Attention : conversion d'un élément masqué en nan.") mais le profil est correctement extrait et affiché, avec des valeurs NaN là où la ROI n'est pas définie.
  - Les valeurs NaN sont maintenant supprimées du profil avant de le tracer
- Les fonctionnalités de traitement simples avec une correspondance un-à-un avec une fonction Python (par exemple `numpy.absolute`, `numpy.log10`, etc.) et sans paramètres : correction du titre de l'objet résultat qui se terminait systématiquement par « | » (le caractère qui précède généralement la liste des paramètres)
- Filtrage de Butterworth : correction de la valeur par défaut et de la plage de valeurs valides de la fréquence de coupure
- Toutes les actions liées à la visualisation sont désormais regroupées dans la barre d'outils verticale du panneau de visualisation
  - Lors du démarrage de DataLab avec le panneau Signal actif, le passage à la Vue Image affichait les actions « Voir dans une nouvelle fenêtre » ou « Modifier les régions d'intérêt » activées dans la barre d'outils verticale, même si aucune image n'était affichée dans la Vue Image
  - La barre d'outils verticale de la Vue Image est maintenant correctement mise à jour au démarrage
- Voir dans une nouvelle fenêtre : les outils de coupe transversale (profils d'intensité) restaient désactivés à moins que l'utilisateur ne sélectionne une image via la liste des éléments - c'est maintenant corrigé
- Vue Image : le bouton de la barre d'outils « Afficher le panneau de contraste » n'était pas activé au démarrage, et n'était activé que lorsqu'au moins une image était affichée dans la Vue Image - il est maintenant toujours activé, comme prévu
- Conversion du type de données des images :
  - Précédemment, la fonction de conversion de type de données était commune aux fonctionnalités de traitement de signal et d'image, c'est-à-dire une simple conversion du type de données en utilisant la méthode `astype` de NumPy
  - Cela n'était pas suffisant pour les fonctionnalités de traitement d'image, en particulier pour les images avec des données de type entier, car même si le résultat était correct d'un point de vue numérique, un débordement ou un dépassement pouvait légitimement être considéré comme un bug d'un point de vue mathématique
  - La fonction de conversion du type de données des images repose désormais sur la fonction interne `clip_astype`, qui recadre les données dans la plage valide du type de données cible avant de les convertir (dans le cas des images de type entier)
- Problèmes d'extraction de ROI d'image :
  - Plusieurs régressions ont été introduites dans la version 0.16.0 :
    - L'extraction d'une seule ROI circulaire ne fonctionnait pas comme prévu (une ROI rectangulaire était extraite, avec des coordonnées inattendues)
    - L'extraction de plusieurs ROI circulaires conduisait à une extraction de ROI rectangulaire
    - L'extraction de plusieurs ROI ne recadrait plus l'image à la boîte englobante globale des ROIs
  - Ces problèmes sont maintenant corrigés, et des tests unitaires ont été ajoutés pour éviter les régressions :
    - Un algorithme de test indépendant a été implémenté pour vérifier la correction de l'extraction de ROI dans tous les cas mentionnés ci-dessus
    - Les tests couvrent à la fois l'extraction de ROI unique et multiple, avec des ROIs circulaires et rectangulaires
- Problèmes de débordement et de dépassement dans certaines opérations sur des images de type entier :
  - Lors du traitement d'images de type entier, certaines fonctionnalités provoquaient des problèmes de débordement ou de dépassement, entraînant des résultats inattendus (résultats corrects d'un point de vue numérique, mais pas d'un point de vue mathématique)
  - Ce problème ne concernait que les opérations de base (addition, soustraction, multiplication, division et opérations constantes) - toutes les autres fonctionnalités fonctionnaient déjà comme prévu

    - Cela est maintenant corrigé car les résultats sont désormais des images à virgule flottante.
    - Des tests unitaires ont été ajoutés pour éviter les régressions pour toutes ces opérations

4.11 DataLab Version 0.16.4

Cette version concerne une maintenance mineure.

Corrections de bugs :

- PlotPy v2.4.1 ou ultérieur est requis pour corriger les problèmes suivants liés à la fonction d'ajustement de contraste :
    - Une régression a été introduite dans une version antérieure de PlotPy : l'histogramme des niveaux n'était plus supprimé du panneau d'ajustement de contraste lorsque l'image associée était supprimée du graphique
    - Cela est maintenant corrigé : lorsque une image est supprimée, l'histogramme est également supprimé et le panneau de contraste est rafraîchi (ce qui n'était pas le cas même avant la régression)
- Ignorer `AssertionError` dans *config_unit_test.py* lors de l'exécution de la suite de tests sur WSL

Documentation :

- Correction de la référence de classe dans la documentation de `Wrap11Func`

4.12 DataLab Version 0.16.3

Corrections de bugs :

- Correction de l'"Issue #84 - Problèmes de construction avec V0.16.1 : conflit de nom `signal`, ...
    - Ce problème devait être corrigé dans la version 0.16.2, mais la correction n'était pas suffisante
    - Merci à @rolandmas pour avoir signalé le problème et pour l'aide apportée dans l'investigation du problème et le test de la correction
- Correction de l'"Issue #85 - Les chemins des données de test peuvent être ajoutés plusieurs fois à `datalab.utils.tests.TST_PATH`
    - Ce problème est lié à l'"Issue #84
    - Ajouter les chemins des données de test plusieurs fois à `datalab.utils.tests.TST_PATH` entraînait le chargement multiple des données de test, ce qui a conduit à l'échec de certains tests (une solution de contournement simple a été ajoutée à V0.16.2 : ce problème est maintenant corrigé)
    - Merci encore à @rolandmas pour avoir signalé le problème dans le contexte de l'emballage Debian
- Correction de l'"Issue #86 - La moyenne de N images entières dépasse le type de données
- Correction de l'"Issue #87 - Extraction du profil moyen de l'image : `AttributeError` lors de la tentative de modification des paramètres du profil
- Correction de l'"Issue #88 - Profil de segment d'image : inversion des coordonnées des points

4.13 DataLab Version 0.16.2

Cette version nécessite PlotPy v2.4.0 ou ultérieure, qui apporte les corrections de bugs et les nouvelles fonctionnalités suivantes :

- Nouvelles fonctionnalités d'ajustement de contraste et corrections de bugs :
    - Nouvelle disposition : la barre d'outils verticale (qui était contrainte dans une petite zone sur le côté droit du panneau) est désormais une barre d'outils horizontale en haut du panneau, à côté du titre
    - Nouveau bouton « Définir l'échelle » : permet à l'utilisateur de définir manuellement les valeurs minimale et maximale de la plage de l'histogramme
    - Correction des problèmes de mise à jour de l'histogramme lorsque aucune image n'était actuellement sélectionnée (même si une image était affichée et sélectionnée auparavant)

    - La plage de l'histogramme n'était pas mise à jour lorsque la valeur minimale ou maximale était définie à l'aide des boutons « Valeur minimale » ou « Valeur maximale » (qui ont été renommés en « Min. » et « Max. » dans cette version)
    - La plage de l'histogramme n'était pas mise à jour lorsque le bouton « Définir la plage complète » était cliqué, ou lorsque la plage de LUT était modifiée à l'aide du formulaire « Échelles / Plage de LUT » dans le groupe « Propriétés »
  - Menu contextuel de la Vue Image : nouvelle fonctionnalité « Inverser l'axe X »

Nouvelles fonctionnalités mineures et améliorations :

- Types de fichiers image :
  - Ajout de la prise en charge native de la lecture des fichiers image .SPE, .GEL, .NDPI et .REC
  - Ajout de la prise en charge de tout format de fichier pris en charge par `imageio` via le fichier de configuration (l'entrée `imageio_formats` peut être personnalisée pour compléter la liste par défaut des formats pris en charge : voir la documentation pour plus de détails)

Corrections de bugs :

- Analyse de Fourier des images :
  - Correction de l'échelle logarithmique pour le spectre de magnitude (calcul en dB au lieu du logarithme naturel)
  - Correction du calcul de la DSP avec une échelle logarithmique (calcul en dB au lieu du logarithme naturel)
  - Mise à jour de la documentation pour mentionner explicitement que l'échelle logarithmique est en dB
- Correction de l'"Issue #82 - Les macros ne sont pas renommées dans DataLab après les avoir exportées vers des scripts Python
- L'objet `ResultProperties` peut désormais être ajouté aux métadonnées de `SignalObj` ou `ImageObj` même en dehors d'une boucle d'événements Qt (car l'élément d'étiquette n'est plus créé immédiatement)
- La barre de progression est désormais automatiquement fermée en cas d'erreur lors d'une opération longue (par exemple, lors de l'ouverture d'un fichier)
- Soustraction, division… : la boîte de dialogue pour la sélection du second opérande permettait de sélectionner un groupe (seul un signal ou une image devrait être sélectionné)
- Lorsqu'une opération implique un objet (signal ou image) ayant un numéro d'ordre supérieur à l'objet actuel (par exemple, lors de la soustraction d'une image avec une image d'un groupe en dessous de l'image actuelle), le titre de l'objet résultant fait désormais correctement référence aux numéros d'ordre des objets impliqués dans l'opération (par exemple, pour continuer avec l'exemple de soustraction mentionné ci-dessus, le titre de l'objet résultant faisait précédemment référence au numéro d'ordre avant l'insertion de l'image résultante)
- Ajout de la prise en charge d'un dossier de données de test supplémentaire grâce à la variable d'environnement `DATALAB_DATA` (utile à des fins de test, et notamment dans le contexte de la préparation du paquet Debian de DataLab)

4.14 DataLab Version 0.16.1

Depuis la version 0.16.0, de nombreuses fonctions de validation ont été ajoutées à la suite de tests. Le pourcentage de fonctions de calcul validées est passé de 37 % à 84 % dans cette version.

La prise en charge de NumPy 2.0 a été ajoutée avec cette version.

Nouvelles fonctionnalités mineures et améliorations :

- Filtres moyenne mobile et médiane pour les signaux et les images :
  - Ajout du paramètre « Mode » pour choisir le mode du filtre (par exemple « reflect », « constant », « nearest », « mirror », « wrap »)
  - Le mode par défaut est « reflect » pour la moyenne mobile et « nearest » pour la médiane mobile
  - Cela permet de gérer les effets de bord lors du filtrage des signaux et des images

Corrections de bugs :

- Correction de la détection de bord de Canny pour renvoyer une image binaire en `uint8` au lieu de `bool` (par soucis de cohérence avec les autres fonctionnalités de traitement d'image)
- Correction de la normalisation d'image : la borne inférieure était mal définie pour la méthode `maximum`

- Correction de l'erreur `ValueError` lors du calcul de la DSP avec une échelle logarithmique
- Correction de l'algorithme de dérivation d'un signal : utilisation de `numpy.gradient` au lieu d'une implémentation personnalisée
- Correction de l'obsolescence de `cumtrapz` de SciPy : utilisation de `cumulative_trapezoid` à la place
- La sélection de courbe affiche désormais les points individuels de la courbe (auparavant, seule la largeur de la ligne de la courbe était élargie)
- Installeur Windows : ajout de la prise en charge des versions instables (par exemple, 0.16.1.dev0), permettant ainsi d'installer facilement la dernière version de développement de DataLab sur Windows
- Correction de l'"Issue #81 - Lors de l'ouverture de fichiers, afficher la boîte de dialogue de progression uniquement si nécessaire
- Correction de l'"Issue #80 - Tracé des résultats : prise en charge de deux cas d'utilisation
  - Les fonctionnalités du menu Analyse » produisent des *résultats* (scalaires) : détection de taches (coordonnées de cercle), détection de pics 2D (coordonnées de points), etc. Selon la fonctionnalité, des tables de résultats sont affichées dans la boîte de dialogue « Résultats », et les résultats sont également stockés dans les métadonnées du signal ou de l'image : chaque ligne de la table de résultats est un résultat individuel, et chaque colonne est une propriété du résultat - certains résultats peuvent ne consister qu'en un seul résultat individuel (par exemple, le centroïde de l'image ou la FWHM de la courbe), tandis que d'autres peuvent consister en plusieurs résultats individuels (par exemple, détection de taches, détection de contours, etc.).
  - Avant ce changement, la fonctionnalité « Tracer les résultats » ne prenait en charge que le tracé du premier résultat individuel d'une table de résultats, en fonction de l'index (des objets de signal ou d'image) ou de l'une des colonnes de la table de résultats. Cela n'était pas suffisant pour certains cas d'utilisation, où l'utilisateur souhaitait tracer plusieurs résultats individuels d'une table de résultats.
  - A présent, la fonctionnalité « Tracer les résultats » prend en charge deux cas d'utilisation :
    - « Une courbe par titre de résultat » : Tracer le premier résultat individuel d'une table de résultats, comme auparavant
    - « Une courbe par objet (ou ROI) et par titre de résultat » : Tracer tous les résultats individuels d'une table de résultats, en fonction de l'index (des objets de signal ou d'image) ou de l'une des colonnes de la table de résultats
  - La sélection du cas d'utilisation se fait dans la boîte de dialogue « Tracer les résultats »
  - Le cas d'utilisation par défaut est « Une courbe par titre de résultat » si la table de résultats ne comporte qu'une seule ligne, et « Une courbe par objet (ou ROI) et par titre de résultat » dans les autres cas

4.15 DataLab Version 0.16.0

Nouvelles fonctionnalités et améliorations :

- Refonte majeure de l'interface utilisateur :
  - La barre de menu et les barres d'outils ont été réorganisées pour rendre l'application plus intuitive et plus facile à utiliser
  - Les fonctionnalités d'opérations et de traitement ont été regroupées dans des sous-menus
  - Toutes les actions liées à la visualisation sont désormais regroupées dans la barre d'outils verticale du panneau de visualisation
  - Clarification de la gestion des « Annotations » (nouveaux boutons, action de la barre d'outils…)
- Nouveau processus de validation pour les fonctionnalités de traitement du signal et de l'image :
  - Avant cette version, le processus de validation de DataLab était exclusivement effectué du point de vue du programmeur, en écrivant des tests unitaires et des tests d'intégration, garantissant ainsi que le code fonctionnait comme prévu (c'est-à-dire qu'aucune exception n'était levée et que le comportement était correct)
  - Avec cette version, un nouveau processus de validation a été introduit, du point de vue de l'utilisateur, en ajoutant de nouvelles fonctions de validation (marquées avec le décorateur `@pytest.mark.validation`) dans la suite de tests
  - Une nouvelle section « Validation » dans la documentation explique comment la validation est effectuée et contient une liste de toutes les fonctions de validation avec les statistiques du processus de validation (générées à partir de la suite de tests)

    - Le processus de validation est en cours et sera amélioré dans les versions futures
- Groupe « Propriétés » :
    - Ajout de l'onglet « Échelles », pour afficher et définir les échelles du graphique :
        - X, Y pour les signaux
        - X, Y, Z (plage de LUT) pour les images
- Options d'affichage :
    - Nouvelle option « Afficher uniquement le premier » dans le menu « Affichage », pour afficher uniquement la première courbe (ou image) lorsque plusieurs courbes (ou images) sont affichées dans la vue du graphique
    - Nouvelle étiquette (déplaçable) pour les calculs de FWHM, en plus de l'annotation de segment existante
- Fonctionnalités d'E/S :
    - Ajout de la prise en charge de la lecture et de l'écriture des fichiers .MAT (format MATLAB)
    - Création d'un nouveau groupe lors de l'ouverture d'un fichier contenant plusieurs signaux ou images (par exemple, un fichier CSV avec plusieurs courbes)
- Ajout de la prise en charge des images binaires
- Extraction de ROI de signal : ajout d'une nouvelle boîte de dialogue pour éditer manuellement les bornes inférieure et supérieure du ROI après avoir défini le ROI graphiquement

Nouvelles fonctionnalités de traitement des **Signaux** :

Menu	Sous-menu	Fonctionnalités
Nouveau	Nouveau signal	Exponentielle, impulsion, polynomiale, personnalisé (entrée manuelle)
Opérations		Exponentielle, Racine carrée, Puissance
Opérations	Opérations avec une constante	+, -, *, /
Traitement	Transformation des axes	Inverser l'axe X
Traitement	Ajustement du niveau	Correction d'offset
Traitement	Analyse de Fourier	Spectre de puissance, Spectre de phase, Spectre de magnitude, Densité spectrale de puissance
Traitement	Filtres fréquentiels	Passe-bas, Passe-haut, Passe-bande, Coupure de bande
Traitement		Fenêtrage (Hanning, Hamming, Blackman, Blackman-Harris, Nuttall, Flat-top…)
Traitement	Ajustement	Ajustement linéaire, Ajustement sinusoïdal, Ajustement exponentiel, Ajustement CDF
Analyse		LMH (Méthode du zéro-crossing), Valeur X @ min/max, Période/fréquence d'échantillonnage, Paramètres dynamiques (ENOB, SNR, SINAD, THD, SFDR), Largeur de bande -3dB, Contraste

Nouvelles fonctionnalités de traitement des **Images** :

Menu	Sous-menu	Fonctionnalités
Opérations		Exponentielle
Opérations	Profils d'intensité	Profil le long d'un segment
Opérations	Opérations avec une constante	+, -, *, /
Traitement	Ajustement du niveau	Normalisation, Recadrage, Correction d'offset
Traitement	Analyse de Fourier	Spectre de puissance, Spectre de phase, Spectre de magnitude, Densité spectrale de puissance
Traitement	Seuillage	Paramétrique, ISODATA, Li, Moyenne, Minimum, Otsu, Triangle, Yen

Corrections de bugs :

- Correction d'un problème de performance dû à un rafraîchissement inutile de la vue du graphique lors de l'ajout d'un nouveau signal ou d'une nouvelle image
- Correction de l'"Issue #77 - Profils d'intensité : impossible d'accepter la boîte de dialogue la deuxième fois
- Correction de l'"Issue #75 - Afficher dans une nouvelle fenêtre : l'anti-crénelage de la courbe n'est pas activé par défaut
- La visibilité des annotations est désormais correctement sauvegardée et restaurée :
  - Avant cette version, lorsque la visibilité des annotations était modifiée dans la vue du graphique séparée, elle n'était pas sauvegardée et restaurée lors de la réouverture de la vue du graphique
  - Cela a été corrigé en amont dans PlotPy (v2.3.3)

4.16 DataLab Version 0.15.1

Corrections de bugs :

- Correction de l'"Issue #68 - Chargement lent même pour des graphiques simples :
  - Sur macOS, l'expérience utilisateur était dégradée lors de la manipulation de graphiques simples
  - Cela était dû à la manière dont macOS gère les fenêtres contextuelles, par exemple lors du rafraîchissement de la vue du graphique (barre de progression « Création d'éléments de graphique »), provoquant ainsi un effet de scintillement très gênant et un ralentissement global de l'application
  - Le problème est maintenant résolu en affichant la barre de progression uniquement après un court délai (1s), c'est-à-dire lorsqu'elle est vraiment nécessaire (c'est-à-dire pour les opérations longues)
  - Merci à @marcel-goldschen-ohm pour la description très détaillée du problème et l'aide apportée pour tester la correction
- Correction de l'"Issue #68 - Les annotations devraient être en lecture seule dans la vue Signal/Image
  - En ce qui concerne les annotations, le comportement actuel de DataLab est le suivant :
    - Les annotations ne sont créées que lors de l'affichage du signal/image dans une fenêtre séparée (double-clic sur l'objet, ou « Affichage » > « Afficher dans une nouvelle fenêtre »)
    - Quand les objets sont affichés dans la « Vue Signal » ou la « Vue Image », les annotations devraient être en lecture seule (c'est-à-dire non déplaçables, ni redimensionnables ou supprimables)
  - Toutefois, certaines annotations étaient toujours supprimables dans la « Vue Signal » et la « Vue Image » : c'est maintenant corrigé
  - Notons que le fait que les annotations ne peuvent pas être créées dans la « Vue Signal » ou la « Vue Image » est une limitation de l'implémentation actuelle, et peut être amélioré dans les versions futures

4.17 DataLab Version 0.15.0

Nouvel installeur pour la version autonome sur Windows :

- La version autonome sur Windows est désormais distribuée en tant qu'installeur MSI (au lieu d'un installeur EXE)
- Cela évite la détection de faux positifs de la version autonome comme une menace potentielle par certains logiciels antivirus
- Le programme installera des fichiers et des raccourcis :
  - Pour l'utilisateur actuel, si l'utilisateur n'a pas de privilèges d'administrateur
  - Pour tous les utilisateurs, si l'utilisateur a des privilèges d'administrateur
  - Le répertoire d'installation peut être personnalisé
- L'installeur MSI permet d'intégrer l'installation de DataLab de manière transparente dans le système de déploiement d'une organisation

Nouvelles fonctionnalités et améliorations :

- Ajout de la prise en charge des fichiers texte/CSV volumineux :
  - Les fichiers de plus de 1 Go (et avec un nombre raisonnable de lignes) peuvent désormais être importés en tant que signaux ou images sans quitter inopinément l'application ou même la ralentir
  - Le fichier est lu par morceaux et, pour les signaux, les données sont rééchantillonnées à un nombre raisonnable de points pour la visualisation
  - Les fichiers volumineux sont pris en charge lors de l'ouverture d'un fichier (ou du glisser-déposer d'un fichier dans le Panneau Signal) et lors de l'importation d'un fichier dans l'Assistant d'importation de texte
- Nouvelle fonctionnalité de sous-échantillonnage automatique :
  - Ajout de la fonctionnalité « Sous-échantillonnage automatique » aux paramètres de visualisation des signaux (voir la boîte de dialogue « Paramètres »)
  - Cette fonctionnalité permet de rééchantillonner automatiquement les données du signal pour la visualisation lorsque le nombre de points est trop élevé et entraînerait un rendu lent
  - Le facteur de sous-échantillonnage est automatiquement calculé en fonction du nombre maximal de points à afficher configuré
  - Cette fonctionnalité est activée par défaut et peut être désactivée dans les paramètres de visualisation du signal
- Gestion du format CSV :
  - Amélioration de la prise en charge des fichiers CSV avec une ligne d'en-tête (noms de colonnes)
  - Ajout de la prise en charge des fichiers CSV avec des colonnes vides
- Gestion des erreurs d'ouverture/enregistrement de fichiers :
  - Les messages d'erreur sont désormais plus explicites lorsque l'ouverture ou l'enregistrement d'un fichier échoue
  - Ajout d'un lien vers le dossier contenant le fichier dans le message d'erreur
- Ajout de la page « Plugins et fonctionnalités d'entrée/sortie » au Visualiseur d'installation et de configuration (voir le menu « Aide »)
- Réinitialisation de la configuration de DataLab :
  - Dans certains cas, il peut être utile de réinitialiser le fichier de configuration de DataLab à ses valeurs par défaut (par exemple, lorsque le fichier de configuration est corrompu)
  - Ajout de la nouvelle option de ligne de commande `--reset` pour supprimer le dossier de configuration
  - Ajout du nouveau raccourci du Menu Démarer « Reset DataLab » à l'installeur Windows

Corrections de bugs :

- Correction de l'"Issue #64 - L'explorateur HDF5 ne montre pas les ensembles de données de taille 1x1 :
  - Les datasets HDF5 de taille 1x1 n'étaient pas affichés dans l'explorateur HDF5
  - Même si ces données ne devraient pas être considérées comme des signaux ou des images, elles sont désormais affichées dans l'explorateur HDF5 (mais non sélectionnables, c'est-à-dire non importables en tant que signaux ou images)

4.18 DataLab Version 0.14.2

Modifications de l'API nécessaires pour corriger la fonctionnalité de chargement de plusieurs signaux :

- Fusion des méthodes `open_object` et `open_objects` en `load_from_files` dans les classes proxy, la fenêtre principale et les panneaux de données
- Pour des raisons de cohérence : fusion de `save_object` et `save_objects` en `save_to_files`
- En résumé, ces changements mènent à la situation suivante :
  - `load_from_files` : charge une séquence d'objets à partir de plusieurs fichiers
  - `save_to_files` : sauvegarde une séquence d'objets dans plusieurs fichiers (pour le moment, il ne prend en charge que la sauvegarde d'un seul objet dans un seul fichier, mais il pourrait être étendu à l'avenir pour prendre en charge la sauvegarde de plusieurs objets dans un seul fichier)

Corrections de bugs :

- Correction de l'"Issue #61 - Assistant d'importation de fichiers texte : plantage de l'application lors de l'importation d'un fichier texte à courbes multiples :
  - Ce problème concerne un cas d'utilisation où le fichier texte contient plusieurs courbes
  - C'est maintenant corrigé et un test automatique a été ajouté pour éviter les régressions

4.19 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. 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 - `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`

4.20 DataLab Version 0.14.0

Nouvelles fonctionnalités :

- Nouvelle fonctionnalité « Histogramme » dans le menu Analyse » :
  - 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 - 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 - 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 - 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

4.21 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 - Sauvegarder/restaurer la disposition de la fenêtre principale

Corrections de bugs :

- Correction de l'"Issue #41 - Extraction du profil radial : impossible d'entrer les coordonnées du centre définies par l'utilisateur
- Correction de l'"Issue #49 - Erreur lors de l'ouverture d'un fichier texte (UTF-8 BOM) en tant qu'image
- Correction de l'"Issue #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

4.22 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 - 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 - 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 pour sa contribution à l'interpolation spline
  - Ceci corrige l'"Issue #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 - 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 - 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 - Ajout de la fonctionnalité d'élimination de tendance 1D
- Résultats d'analyse 2D :
  - Avant cette version, les résultats d'analyse 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 - 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 d'analyse 1D et 2D :
  - En complément de l'amélioration précédente, plus de résultats d'analyse sont désormais affichés dans la boîte de dialogue « Résultats »
  - Cela concerne à la fois les résultats d'analyse 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 Analyse » :
  - Cette fonctionnalité permet de tracer les résultats d'analyse (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 *datalab_testdata.py* en conséquence)
- API de calculs sur les signaux et les images (`datalab.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 *datalab_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 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 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 - 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 - 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 ROI 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 ROI 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 - Extraction d'une seule ROI circulaire : bascule automatique vers la fonction `compute_extract_roi`
- Analyse 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 - Analyse sur ROI circulaire : 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 `datalab.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 pour sa contribution)
    - Ceci corrige l'"Issue #34 - Détection des contours : résultats inattendus en dehors de la ROI
- Fusion des résultats d'analyse :
    - Avant cette version, lors d'un calcul `1->N` (somme, moyenne, produit) sur un groupe de signaux/images, les résultats d'analyse 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 d'analyse associés à chaque signal/image sont fusionnés en un seul résultat, quel que soit le type de résultat.
- Ceci corrige l'"Issue #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

4.23 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 Analyseer ») :
  - 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 - 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 - 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 - 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 - Enregistrer/Réouvrir le projet HDF5 sans nettoyage entraîne une `ValueError`
- Correction de l'"Issue #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
- Visualiseur 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 - Version autonome sur Windows 7 : `api-ms-win-core-path-l1-1-0.dll` manquant)
- Correction de l'"Issue #13 - ArchLinux : `qt.qpa.plugin: Could not load the Qt platform plugin "xcb" in "" even though it was found`

4.24 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 ROI 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 - 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 - 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 - 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

4.25 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 - Windows : la traduction gettext ne fonctionne pas sur les applications gelées)
  - Issue #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 - 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)
  - Issue #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

4.26 DataLab Version 0.9.0

Nouvelles dépendances :

- DataLab est désormais alimenté par PlotPyStack :
  - PythonQwt
  - guidata
  - PlotPy
- 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)
  - Ajout de commandes de macro : DataLab peut être contrôlé via un fichier macro (voir Commandes de macro)
- 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 analyse), 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.

Note : Ce projet (DataLab Platform) ne doit pas être confondu avec le projet datalab-org, qui est une initiative distincte et sans lien, axée sur les bases de données de science des matériaux et les outils de calcul.

Index des modules Python

d

datalab.control.proxy, 247
datalab.gui, 259
datalab.gui.actionhandler, 289
datalab.gui.docks, 303
datalab.gui.h5io, 304
datalab.gui.main, 260
datalab.gui.objectview, 295
datalab.gui.panel, 266
datalab.gui.panel.base, 267
datalab.gui.panel.image, 281
datalab.gui.panel.macro, 286
datalab.gui.panel.signal, 278
datalab.gui.plothandler, 298
datalab.gui.processor, 302
datalab.gui.roieditor, 301
datalab.plugins, 207