

DataLab

Version 1.4.0

06 novembre 2026

Table des matières

1 Premiers pas 3
- 1.1 Installation . . . 4
- 1.2 Installation hors ligne . . . 11
- 1.3 Cas d'usage, principales fonctionnalités et points forts . . . 16
- 1.4 La plateforme DataLab . . . 18
- 1.5 Fonctionnalités principales . . . 23
- 1.6 Tutoriels . . . 25

2 Cas d'usage 111
- 2.1 DataLab pour la spectroscopie . . . 111
- 2.2 DataLab pour la photonique et les lasers . . . 112
- 2.3 DataLab pour le contrôle non destructif . . . 113

3 Fonctionnalités 115
- 3.1 Vue d'ensemble et fonctionnalités communes . . . 115
- 3.2 Traitement du signal . . . 150
- 3.3 Traitement d'image . . . 193
- 3.4 Validation . . . 240
- 3.5 Fonctionnalités avancées . . . 249

4 Contribuer 415
- 4.1 Partagez vos idées et vos expériences . . . 415
- 4.2 Partagez vos connaissances scientifiques/techniques . . . 415
- 4.3 Contribuer aux nouvelles fonctionnalités . . . 416
- 4.4 Développer de nouvelles fonctionnalités . . . 416

5 Notes de version 433
- 5.1 Version 1.4 . . . 433
- 5.2 Version 1.3 . . . 449
- 5.3 Version 1.2 . . . 454
- 5.4 Version 1.1 . . . 459
- 5.5 Version 1.0 . . . 460
- 5.6 Version 0.20 . . . 473
- 5.7 Version 0.19 . . . 475
- 5.8 Version 0.18 . . . 477
- 5.9 Version 0.17 . . . 479
- 5.10 Version 0.16 . . . 482

5.11 Version 0.15 486
5.12 Version 0.14 487
5.13 Version 0.12 489
5.14 Version 0.11 490
5.15 Version 0.10 494
5.16 Version 0.9 496

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	CEA, le Commissariat à l'énergie atomique et aux énergies alternatives, financeur principal de DataLab, contribue significativement au projet.
CODRA	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	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).

L'application de bureau elle-même est construite sur SigimaX, le framework d'application Qt de la plateforme DataLab : squelette de la fenêtre principale, système de configuration, widgets de tracé amovibles, espace de travail et navigateur HDF5, visionneuse de journaux et dialogues scientifiques (voir 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). Its graphical user interface is built on SigimaX, the Qt application framework 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.

> **Note**
>
> DataLab est l'application de bureau au cœur d'une famille plus large, la plateforme DataLab, qui comprend également une édition native pour navigateur (DataLab-Web) s'exécutant sans aucune installation. Voir La plateforme DataLab pour découvrir les éditions et choisir celle qui correspond à vos besoins.

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)
- SigimaX pour le framework applicatif Qt sur lequel repose l'application de bureau (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.

Pour l'installation sur des systèmes sans accès à Internet, voir le guide d'installation hors ligne.

1.1.1 Modes d'installation

DataLab est disponible sous plusieurs formes :

— En tant que Paquet Conda.
— GNU/Linux macOS En tant que Paquet NixOS, pour les utilisateurs de l'écosystème NixOS.
— 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
```

Paquet NixOS

GNU/Linux macOS

DataLab est packagé pour NixOS et distribué via la NGI Forge, le système de distribution logicielle des projets financés par l'initiative Next Generation Internet (NGI). Ce packaging a été réalisé par l'équipe Nix@NGI, partenaire de NLnet, dans le cadre du financement de DataLab par le NGI0 Commons Fund.

Sur tout système où Nix est installé, entrez dans le shell Nix qui fournit DataLab (la commande prête à copier est disponible sur la page NGI Forge), puis démarrez l'application de bureau :

```
$ datalab
```

Ou, pour une démonstration automatisée de certaines fonctionnalités de DataLab :

```
$ datalab-demo
```

Gestionnaire de paquets `pip`

GNU/Linux Windows macOS

Le paquet `datalab-platform` de DataLab est disponible sur le Python Package Index (PyPI) à l'adresse suivante : 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 !

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).

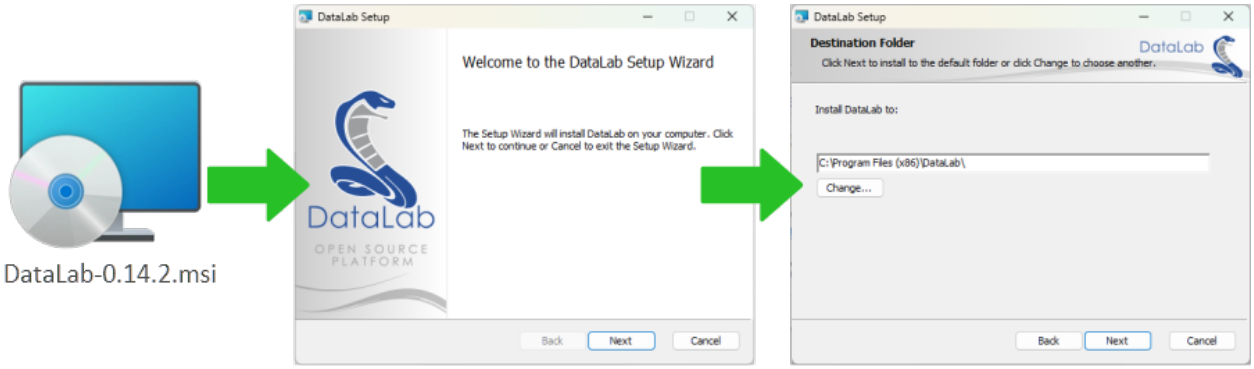


Fig. 1 – Installeur tout-en-un de DataLab pour Windows

> **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.

> **Note**
>
> DataLab-WinPython inclut Spyder, un IDE puissant pour la programmation scientifique en Python, ainsi que Jupyter Notebook pour le calcul interactif.

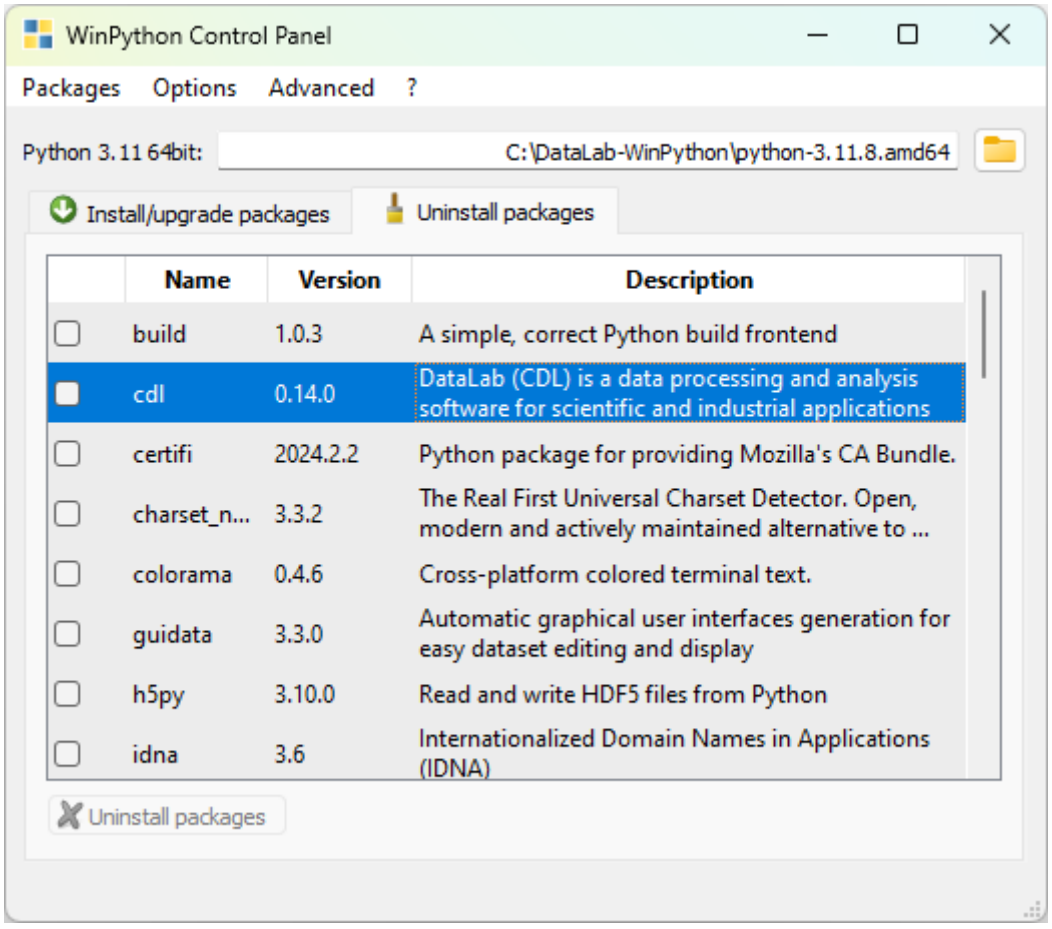


Fig. 3 – Panneau de contrôle DataLab-WinPython

> **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-1.0.1-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-1.0.1.tar.gz
```

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

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

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

```
$ pip install build wheel  # Install build and wheel packages (if needed)
$ python -m build  # Build the wheel package
$ pip install --upgrade dist/datalab-1.0.1-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	>=3.10, <4	Langage de programmation Python
guidata	>= 3.17.2	Génération automatique d'interfaces graphiques pour l'édition et l'affichage de jeux de données
PlotPy	>= 2.12.1	Outils de tracé de courbes et d'images pour les applications Python/Qt
Sigima	>= 1.4.0	Moteur de calcul scientifique pour signaux 1D et images 2D, faisant partie de la plateforme open-source DataLab.
SigimaX	>= 1.1.0	Composants d'interface graphique réutilisables et utilitaires applicatifs, faisant partie de la plateforme DataLab de traitement du signal et des images
datalab-capsule	>= 0.2.0, < 0.3	Échange et vérification de la provenance d'espaces de travail scientifiques sous forme de capsules RO-Crate ouvertes (bibliothèque et ligne de commande, sans DataLab)
NumPy	>= 1.22, < 2.6	Paquet fondamental pour le calcul sur tableaux en Python
SciPy	>= 1.10.1, < 1.19	Algorithmes fondamentaux pour le calcul scientifique en Python
scikit-image	>= 0.19.2, < 0.27	Traitement d'images en Python
pandas	>= 1.4, < 3.1	Analyse de données, séries temporelles et statistiques
PyWavelets	>= 1.2, < 2.0	PyWavelets, module de transformation en ondelettes
psutil	>= 5.8	Bibliothèque multiplateforme pour la surveillance des processus et du système.
packaging	>= 21.3	Utilitaires de base pour les paquets Python
pip	>= 22.2	L'outil recommandé par la PyPA pour installer des paquets Python.
jedi	>= 0.18.2	Un outil d'autocomplétion Python utilisable dans les éditeurs de texte.
fastapi	>= 0.110.0	Framework FastAPI, haute performance, facile à apprendre, rapide à coder, prêt pour la production
uvicorn[standa	>= 0.27.0	
pydantic	>= 2.0	Validation de données en utilisant les annotations de type Python

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		Une interface simple et fiable pour la construction de paquets Python
babel		Utilitaires d'internationalisation
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		Un framework pour gérer et maintenir des hooks de pré-commit pour plusieurs langages.

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		Générateur de plan de site pour Sphinx
sphinxext-opengraph		Extension Sphinx pour la prise en charge du protocole Open Graph (OGP)
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
sphinxcontrib-svg2pdfconverter[CairoSVG]		

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.
httpx		Client HTTP de nouvelle génération.

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

1.2 Installation hors ligne

Ce guide explique comment installer DataLab sur un système sans accès Internet.

> **Note**
>
> Cette page se concentre sur l'installation hors ligne avec pip (gestionnaire de paquets Python). Pour les autres méthodes d'installation hors ligne, consultez la section Autres méthodes d'installation hors ligne ci-dessous.

1.2.1 Installation hors ligne via pip

GNU/Linux Windows macOS

Cette méthode fonctionne sur tout système d'exploitation avec Python 3.10 à 3.14. Elle suppose que vous disposez d'une machine en ligne pour télécharger les paquets, et d'une machine hors ligne distincte où DataLab sera installé.

1.2.2 Prérequis

Machine hors ligne

- GNU/Linux, Windows ou macOS
- Python 3.10 à 3.14 installé avec la même version majeure que la machine en ligne
- Aucune hypothèse concernant les paquets installés

> **Important**
>
> La compatibilité des versions est critique :
> - ✓ Autorisé : Python 3.10.2 ↔ Python 3.10.11 (même version majeure)
> - × Non autorisé : Python 3.10.x ↔ Python 3.11.x (versions majeures différentes)
>
> SciPy, NumPy et scikit-image dépendent fortement de la version majeure de Python. Les deux machines doivent utiliser la même version majeure de Python pour une installation hors ligne fiable.

Pour vérifier votre version de Python, exécutez :

```
python --version
```

Machine en ligne

- Windows, Linux ou macOS
- Python avec la même version majeure que la machine hors ligne
- `pip` installé

1.2.3 Préparation des paquets (machine en ligne)

Téléchargez tous les paquets Wheel nécessaires pour l'installation hors ligne.

Créer un dossier pour les paquets

Windows

```
mkdir C:\datalab-offline
cd C:\datalab-offline
```

GNU/Linux / macOS

```
mkdir ~/datalab-offline
cd ~/datalab-offline
```

Télécharger DataLab et ses dépendances

Windows

```
pip download datalab-platform --dest C:\datalab-offline
```

GNU/Linux / macOS

```
pip download datalab-platform --dest ~/datalab-offline
```

Cette commande télécharge :

- `datalab-platform`
- `sigima`
- `sigimax`
- `plotpy`
- `guidata`
- `pythonqwt`
- `numpy`, `scipy`, `scikit-image` et d'autres dépendances

Structure de dossier attendue

Après le téléchargement, votre dossier devrait contenir des fichiers similaires à :

```
datalab-offline/
├─ datalab_platform-1.x.x-py3-none-any.whl
├─ sigima-1.x.x-py3-none-any.whl
├─ sigimax-1.x.x-py3-none-any.whl
├─ plotpy-x.x.x-py3-none-any.whl
├─ guidata-3.x.x-py3-none-any.whl
├─ pythonqwt-x.x.x-py3-none-any.whl
├─ numpy-...-cp310-cp310-[platform].whl
├─ scipy-...-cp310-cp310-[platform].whl
├─ scikit_image-...-cp310-cp310-[platform].whl
└─ ...
```

> **Note**
>
> Le suffixe `[platform]` dans les noms de fichiers wheel variera en fonction de votre système : `win_amd64` (Windows), `manylinux` (Linux), `macosx` (macOS).

Transfert vers la machine hors ligne

Transférez le dossier `datalab-offline` vers votre machine hors ligne en utilisant :

- Clé USB
- Partage réseau
- Toute autre méthode approuvée dans votre organisation

1.2.4 Installation hors ligne

Sur la machine hors ligne, naviguez vers le dossier et installez DataLab.

Naviguer vers le dossier

Windows

```
cd C:\datalab-offline
```

GNU/Linux / macOS

```
cd ~/datalab-offline
```

Installer DataLab hors ligne

```
python -m pip install --no-index --find-links . datalab-platform
```

L'option `--no-index` indique à pip de ne pas se connecter à Internet, et `--find-links .` indique à pip de rechercher les paquets dans le répertoire courant.

1.2.5 Vérification

Pour vérifier que DataLab est correctement installé, exécutez :

```
python -m datalab.app
```

Cela devrait lancer l'application DataLab.

1.2.6 Optionnel : Environnement virtuel

Il est recommandé d'installer DataLab dans un environnement virtuel pour éviter les conflits avec d'autres paquets Python.

Créer et activer un environnement virtuel

Windows

```
python -m venv C:\venvs\datalab
C:\venvs\datalab\Scripts\activate
```

GNU/Linux / macOS

```
python -m venv ~/venvs/datalab
source ~/venvs/datalab/bin/activate
```

Installer DataLab dans l'environnement virtuel

Windows

```
python -m pip install --no-index --find-links C:\datalab-offline datalab-platform
```

GNU/Linux / macOS

```
python -m pip install --no-index --find-links ~/datalab-offline datalab-platform
```

Lancer DataLab

```
python -m datalab.app
```

1.2.7 WinPython pour une utilisation hors ligne

Windows

WinPython est particulièrement adapté aux installations hors ligne car il fournit un environnement Python portable et autonome.

Préparation sur la machine en ligne

1. Téléchargez WinPython avec la même version majeure de Python que votre système cible
2. Installez WinPython
3. Ouvrez l"invite de commandes WinPython
4. Téléchargez les paquets :

```
pip download datalab-platform --dest C:\datalab-offline
```

Installation sur la machine hors ligne

1. Transférez WinPython et le dossier `datalab-offline` vers la machine hors ligne
2. Ouvrez l"invite de commandes WinPython
3. Installez DataLab :

```
cd C:\datalab-offline
python -m pip install --no-index --find-links . datalab-platform
```

Avantages de WinPython

— Portable : Aucune installation requise, peut fonctionner depuis une clé USB
— Autonome : Toutes les dépendances incluses
— Idéal pour les environnements restreints : Aucun droit administrateur nécessaire
— Versions multiples : Peut avoir plusieurs versions de Python côte à côte

1.2.8 Résumé rapide

Le flux de travail complet en quatre étapes :

1. Sur la machine en ligne : Télécharger les paquets

```
pip download datalab-platform --dest /path/to/datalab-offline
```

2. Transfert : Copier le dossier `datalab-offline` vers la machine hors ligne
3. Sur la machine hors ligne : Installer hors ligne

```
python -m pip install --no-index --find-links /path/to/datalab-offline datalab-
↪platform
```

4. Vérification : Lancer DataLab

```
python -m datalab.app
```

1.2.9 Autres méthodes d'installation hors ligne

Bien que ce guide se concentre sur l'installation hors ligne avec pip, DataLab peut également être installé hors ligne en utilisant d'autres méthodes :

Installateur autonome (Windows)

Windows Recommandé pour l'installation hors ligne

L'installateur tout-en-un est le moyen le plus simple d'installer DataLab hors ligne sur Windows. Il ne nécessite aucune distribution Python ni connexion Internet sur la machine cible.

Avantages :
- Aucune installation Python requise
- Fichier exécutable unique
- Autonome avec toutes les dépendances
- Aucune connexion Internet nécessaire pendant l'installation
- Fonctionne sur des systèmes isolés

Comment l'utiliser :

1. Sur une machine avec accès Internet, téléchargez l'installateur depuis la page DataLab Releases
2. Transférez l'exécutable de l'installateur vers la machine hors ligne (clé USB, partage réseau, etc.)
3. Exécutez l'installateur sur la machine hors ligne

Pour plus d'informations, consultez Installeur tout-en-un.

Distribution WinPython (Windows)

Windows

DataLab-WinPython est une distribution Python portable qui inclut DataLab et toutes ses dépendances.

Avantages :
- Portable (peut fonctionner depuis une clé USB)
- Aucune installation requise
- Inclut un environnement Python complet
- Peut être étendu avec des paquets supplémentaires
- Idéal pour les environnements restreints

Comment l'utiliser :

1. Téléchargez DataLab-WinPython depuis la page des versions
2. Transférez vers la machine hors ligne

3. Extrayez et exécutez

Pour plus d'informations, consultez Distribution Python.

Installation hors ligne avec Conda

GNU/Linux Windows macOS

Les paquets Conda peuvent également être installés hors ligne en utilisant une approche similaire à pip :

1. Sur la machine en ligne : Téléchargez le paquet conda et ses dépendances :

   ```
   conda create -n datalab-offline --download-only --json conda-forge::datalab
   ```

   Ou utilisez `conda list --explicit` pour créer un fichier de spécification.
2. Transférez les paquets téléchargés vers la machine hors ligne
3. Sur la machine hors ligne : Installez à partir des paquets locaux :

   ```
   conda create -n datalab --offline -c file:///path/to/packages datalab
   ```

Pour plus d'informations sur l'installation hors ligne avec conda, consultez la documentation Conda.

Pour l'installation en ligne standard, consultez Paquet Conda.

> **Voir aussi**
>
> Pour les méthodes d'installation en ligne standard, consultez Installation.

1.3 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.

1.3.1 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.

1.3.2 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

1.3.3 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.

1.4 La plateforme DataLab

Lorsque DataLab a vu le jour en 2023, il s'agissait d'un produit unique : une application de bureau. Depuis, le même cœur scientifique a donné naissance à une petite famille d'outils complémentaires — la plateforme DataLab. Cette page clarifie le vocabulaire et vous aide à choisir l'édition la mieux adaptée à vos besoins.

> Voir aussi
>
> L'histoire de cette évolution — l'extraction du moteur de calcul Sigima et l'édition native pour navigateur — est documentée dans les rapports de projet et les enregistrements de décisions d'architecture publiés sur le wiki DataLab.

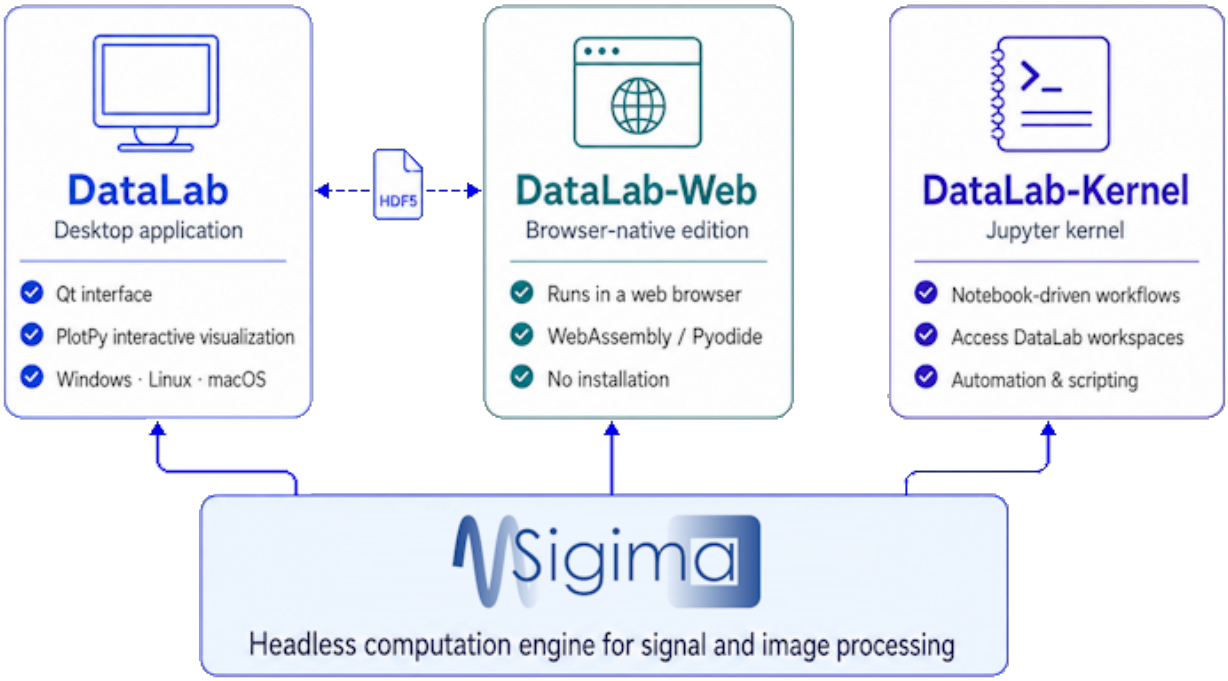


Fig. 4 – La plateforme DataLab : l'application de bureau, DataLab-Web et DataLab-Kernel sont trois modes d'accès partageant le même moteur de calcul Sigima et interopérables via des espaces de travail HDF5.

1.4.1 Une note sur le vocabulaire

Le terme DataLab désigne à la fois l'ensemble de la famille et son membre historique, l'application de bureau. Pour éviter toute confusion, cette documentation emploie le vocabulaire suivant :

Nom	Signification
DataLab Platform	Le terme générique désignant l'ensemble de l'écosystème (tous les produits listés ci-dessous).
DataLab	L"application de bureau de référence (basée sur Qt). Sauf indication contraire, le reste de cette documentation décrit ce produit.
DataLab-Web	L'édition native pour navigateur : la plateforme complète s'exécutant dans un onglet de navigateur web, sans aucune installation. Essayez-la sur datalab-platform.com/web – projet sur GitHub.
DataLab-Kernel	Un noyau Jupyter exposant les espaces de travail DataLab aux notebooks — soit synchronisé en direct avec une session de bureau en cours d'exécution, soit totalement autonome (y compris dans JupyterLite, dans le navigateur). Documentation.
Sigima	Le moteur de calcul sans interface (traitement du signal et de l'image) partagé par tous les produits ci-dessus. Documentation.
SigimaX	Le framework applicatif Qt sur lequel repose l'application de bureau : fenêtre principale générique, système de configuration, widgets de tracé ancrables, espace de travail et navigateur HDF5, visualiseur de journaux et boîtes de dialogue scientifiques. Destiné aux développeurs créant leur propre application scientifique Qt. Documentation.

Note

Ce site de documentation est, avant tout, la documentation de l'"application de bureau. Tout ce qui est commun aux différentes éditions — le modèle objet, le catalogue de traitements, les paramètres — est décrit ici et s'applique également à DataLab-Web. Seules les différences propres à une édition sont signalées explicitement.

1.4.2 Les membres de la plateforme

— DataLab (bureau) — l'application de référence, dotée d'une interface graphique Qt native et de la visualisation interactive complète de PlotPyStack.
— DataLab-Web — la même plateforme, s'exécutant entièrement dans votre navigateur via WebAssembly (datalab-platform.com/web).
— DataLab-Kernel — un noyau Jupyter pour les flux de travail pilotés par notebook, synchronisé en direct avec une session de bureau en cours d'exécution ou totalement autonome (documentation).
— Sigima — le moteur de calcul open source qui propulse chaque édition (documentation).
— SigimaX — le framework applicatif Qt sur lequel repose l'application de bureau, réutilisable pour créer d'autres applications scientifiques Qt (documentation).

1.4.3 Lequel choisir ?

L'application de bureau et DataLab-Web ne sont pas concurrentes : ce sont deux modes d'accès à la même plateforme, et elles peuvent être utilisées indifféremment (les espaces de travail HDF5 sont interopérables). Quelques règles générales :

— Utilisez l'application de bureau lorsque vous travaillez sur votre propre machine et souhaitez l'expérience la plus riche et la plus rapide, l'ensemble complet des outils interactifs PlotPy, un accès natif aux fichiers et la capacité de traiter de très grands jeux de données, limités uniquement par la mémoire vive de votre système.
— Utilisez DataLab-Web lorsque vous ne pouvez pas ou ne voulez rien installer — par exemple sur un ordinateur partagé ou verrouillé, sur la machine de quelqu'un d'autre, ou simplement pour essayer rapidement DataLab. Tout s'exécute localement dans l'onglet du navigateur ; aucun serveur, aucun compte, aucun téléversement.
— Utilisez DataLab-Kernel lorsque votre flux de travail est centré sur les notebooks et que vous souhaitez disposer du catalogue de traitements de DataLab depuis un notebook Jupyter — par exemple pour rédiger des rapports d'analyse reproductibles, rejouables avec ou sans DataLab. Vous pouvez l'essayer en ligne dans un environnement JupyterLite, sans rien installer.

Un usage courant consiste à utiliser l'application de bureau sur votre poste de travail personnel et DataLab-Web partout ailleurs, en échangeant le travail dans les deux sens via des fichiers d'espace de travail HDF5.

1.4.4 DataLab et DataLab-Web

DataLab-Web partage le moteur de calcul (Sigima) et le catalogue de traitements avec l'application de bureau : c'est le même code Sigima — et non un portage — qui s'exécute sur CPython (bureau) et sur Pyodide (navigateur), de sorte que les résultats sont identiques pour les mêmes entrées. Les différences portent sur l'"environnement d'exécution, non sur la science.

Sujet	DataLab (bureau)	DataLab-Web
Installation	Installé localement (Windows, Linux, macOS)	Aucune — s'exécute dans le navigateur
Environnement d'exécution Python	CPython du système	Pyodide (CPython compilé en WebAssembly)
Interface graphique	Qt + PlotPy	React + Plotly.js
Tracé	PlotPy (Qt), outils interactifs complets	Plotly.js (outils interactifs partiellement réimplémentés)
Accès aux fichiers	E/S natives du système de fichiers	Sélecteur de fichiers du navigateur, glisser-déposer
Persistance	HDF5 sur disque	Téléchargement / téléversement HDF5 (espaces de travail interopérables)
Mémoire	Limitée par la mémoire vive du système (64 bits)	Tas WebAssembly d'environ 2 Go par défaut ; un mode de stockage sur disque optionnel supprime entièrement cette limite, le jeu de travail étant alors borné par l'espace disque disponible plutôt que par la mémoire vive
Réactivité	Les calculs s'exécutent dans des processus de travail séparés ; l'interface reste réactive et les traitements peuvent être annulés	L'environnement Python s'exécute dans un Web Worker, hors du fil d'exécution de l'interface ; celle-ci reste réactive et les traitements par lots peuvent être annulés entre deux objets
Langues	Anglais et français	Anglais et français (langue du navigateur détectée automatiquement)
Pilotage à distance	XML-RPC et API Web FastAPI	Proxy intégré au navigateur et SDK `postMessage` optionnel

Ce qui ne change pas : le modèle objet (`SignalObj`, `ImageObj`, ROI, groupes), le catalogue de traitements découvert depuis Sigima, les boîtes de dialogue de paramètres générées automatiquement et le format d'espace de travail HDF5. Un espace de travail enregistré dans l'application de bureau peut être ouvert dans DataLab-Web, et inversement.

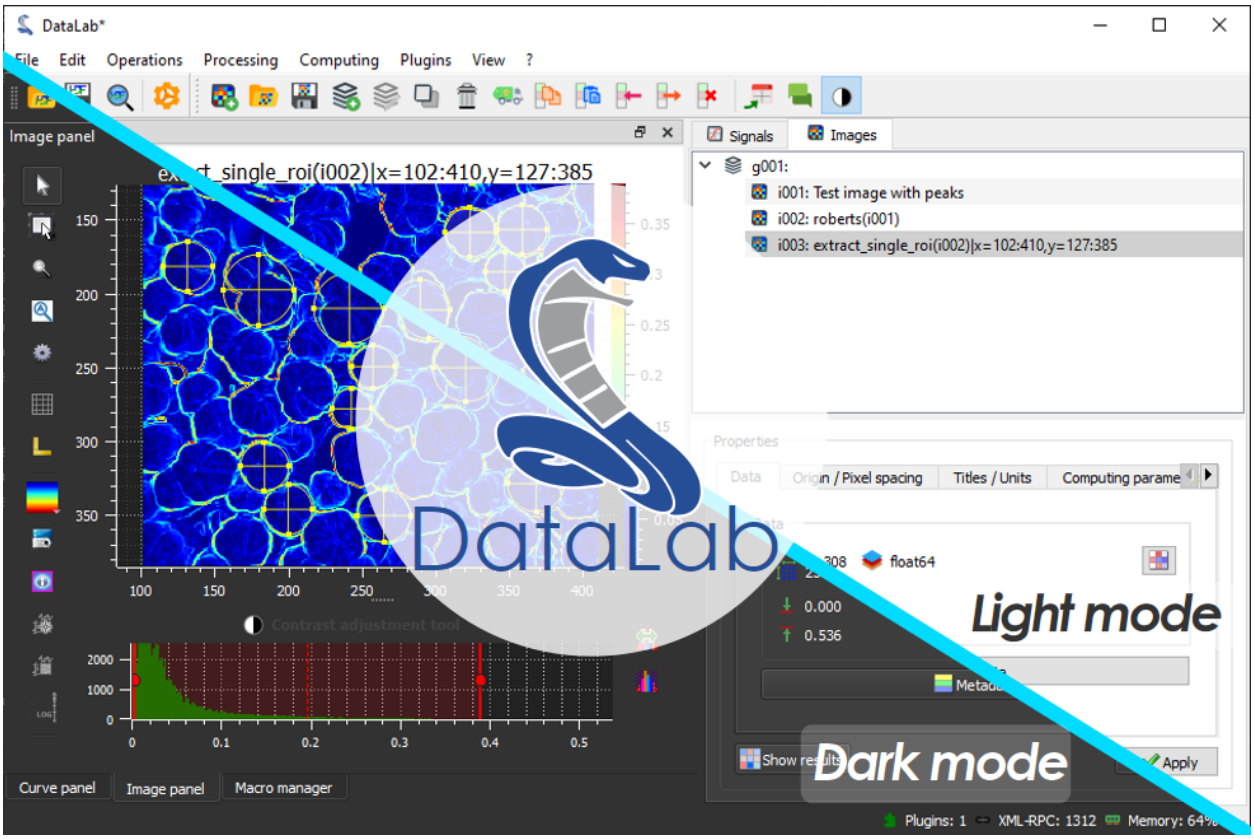


Fig. 5 – DataLab prend en charge le mode sombre et le mode clair en fonction des paramètres de votre plateforme (cela est géré par le paquet guidata et peut être remplacé en définissant la variable d'environnement QT_COLOR_MODE sur dark ou light).

> **Note**
>
> Les plugins sont en grande partie portables entre l'application de bureau et DataLab-Web, avec quelques contraintes liées à l'environnement d'exécution du navigateur. Voir Plugins pour plus de détails.

1.5 Fonctionnalités principales

Cette page présente brièvement les principales fonctionnalités de DataLab.

1.5.1 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

1.5.2 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…

suite sur la page suivante

Tableau 1 – suite de la page précédente

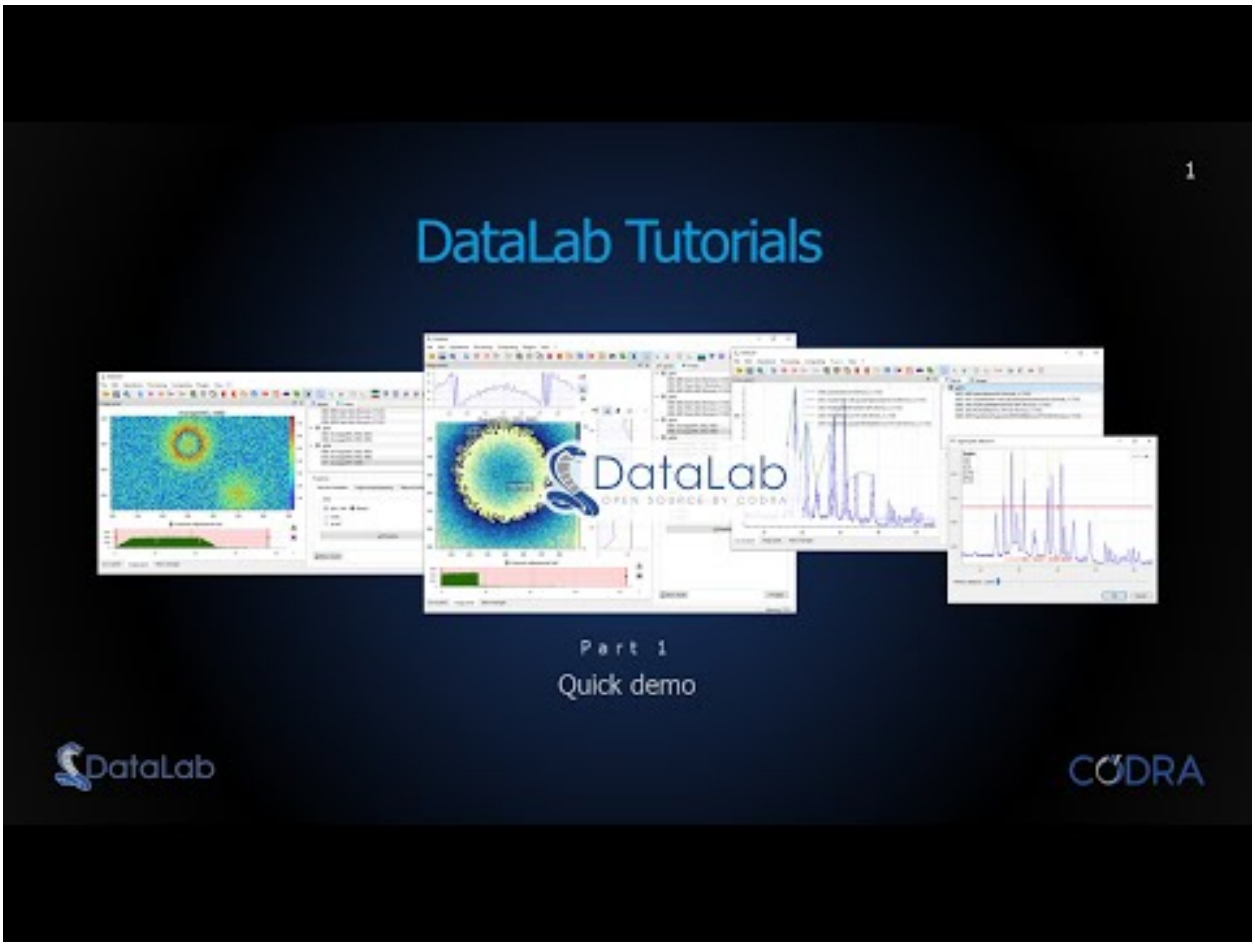
Signal	Image	Fonctionnalité
	✓	Profils d'intensité (ligne, moyen, radial)
	✓	Binning de pixels
✓	✓	Calibration linéaire
✓	✓	Normalisation, rognage, correction de décalage
✓		Inversion de l'axe X
✓		Remplacer X par le Y d'un autre signal (calibration)
✓		Mode X-Y
	✓	Seuillage (manuel, Otsu…)
✓	✓	Filtre 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
✓		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
	✓	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…)

1.6 Tutoriels

1.6.1 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



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.

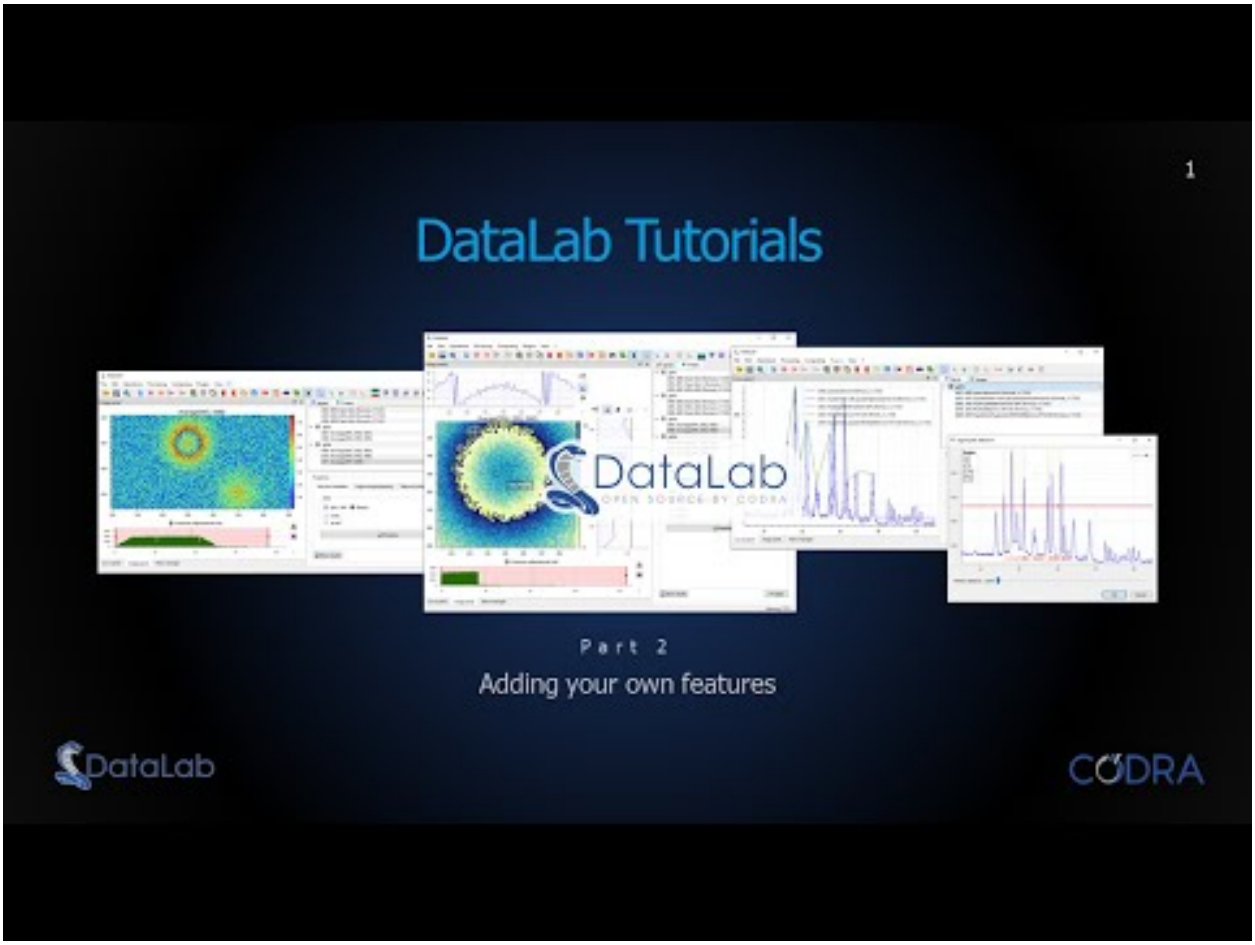
Dans cette première vidéo, nous allons rapidement passer en revue les principales fonctionnalités de DataLab, et montrer comment effectuer un traitement de signal et d'image de base.

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



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.

1.6.2 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.

Analyse de spectre : débruitage, correction de ligne de base et ajustement de pics

Cet exemple montre comment analyser 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.

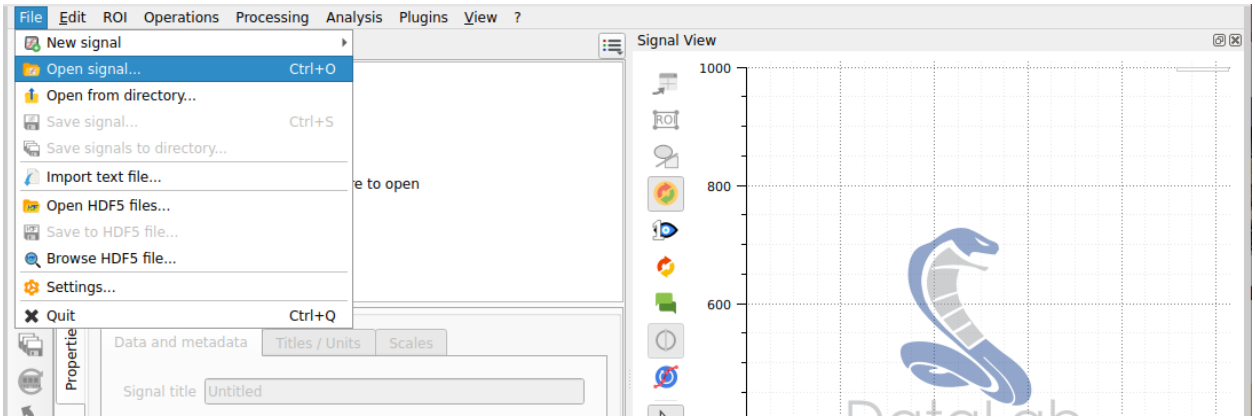


Fig. 6 – Le menu « Fichier > Ouvrir… » pour ouvrir un fichier de spectre.

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.

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.

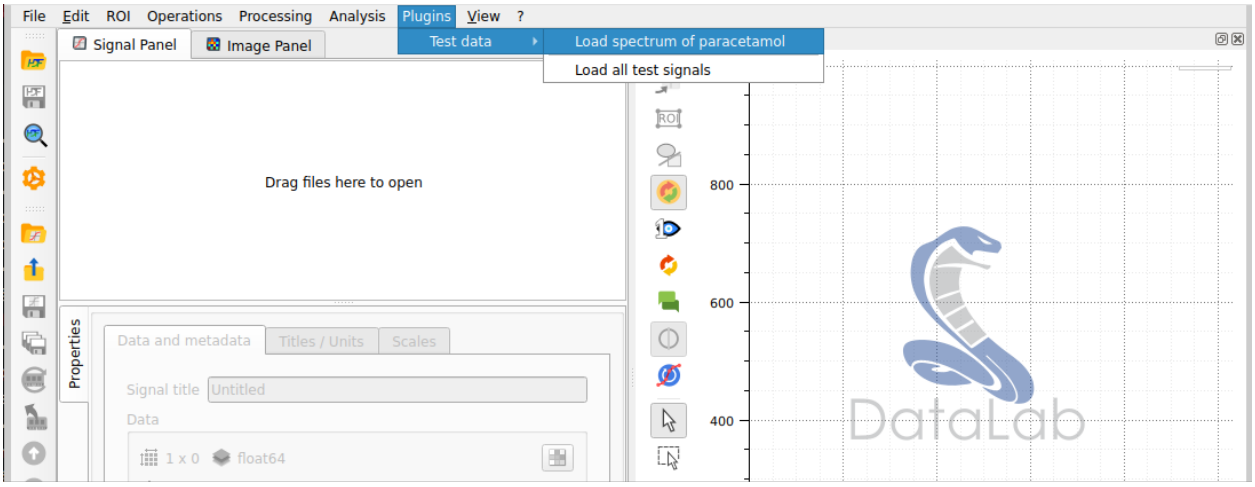


Fig. 7 – Le plugin « Plugins > Données de test > Charger le spectre de paracétamol » pour générer le spectre de test pour ce tutoriel.

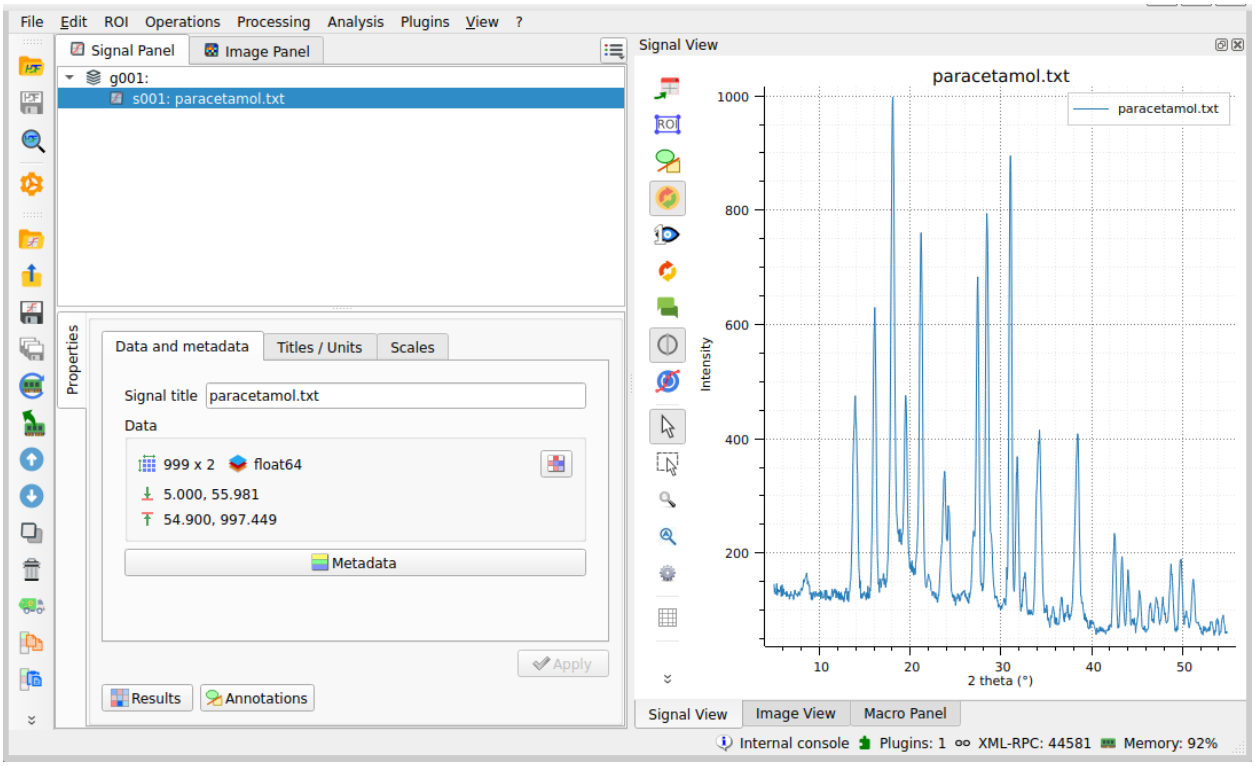


Fig. 8 – Le spectre affiché dans le panneau « Vue Signal ».

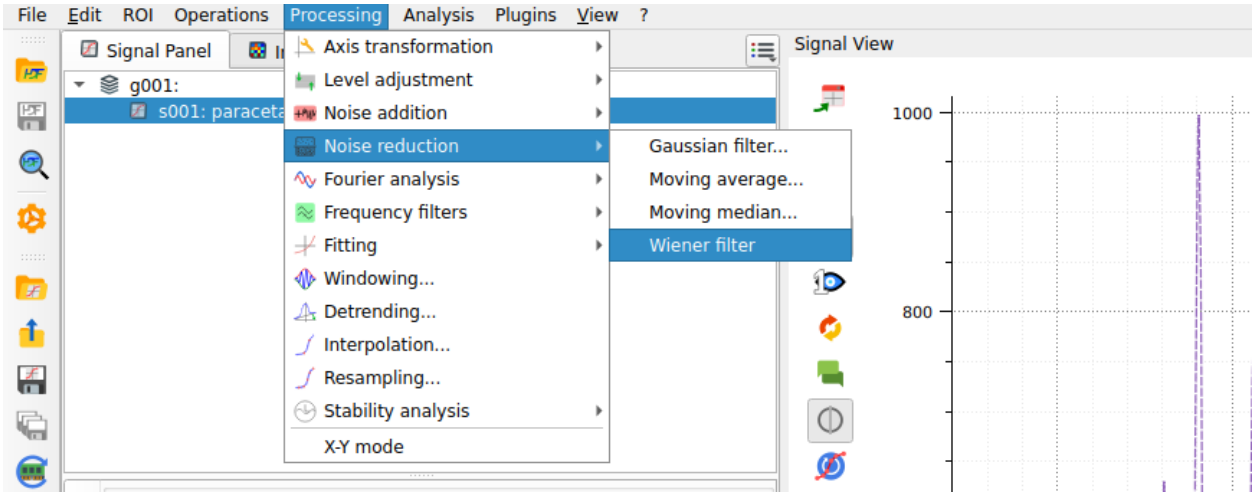


Fig. 9 – L'option de menu « Traitement > Réduction du bruit > Filtre de Wiener ».

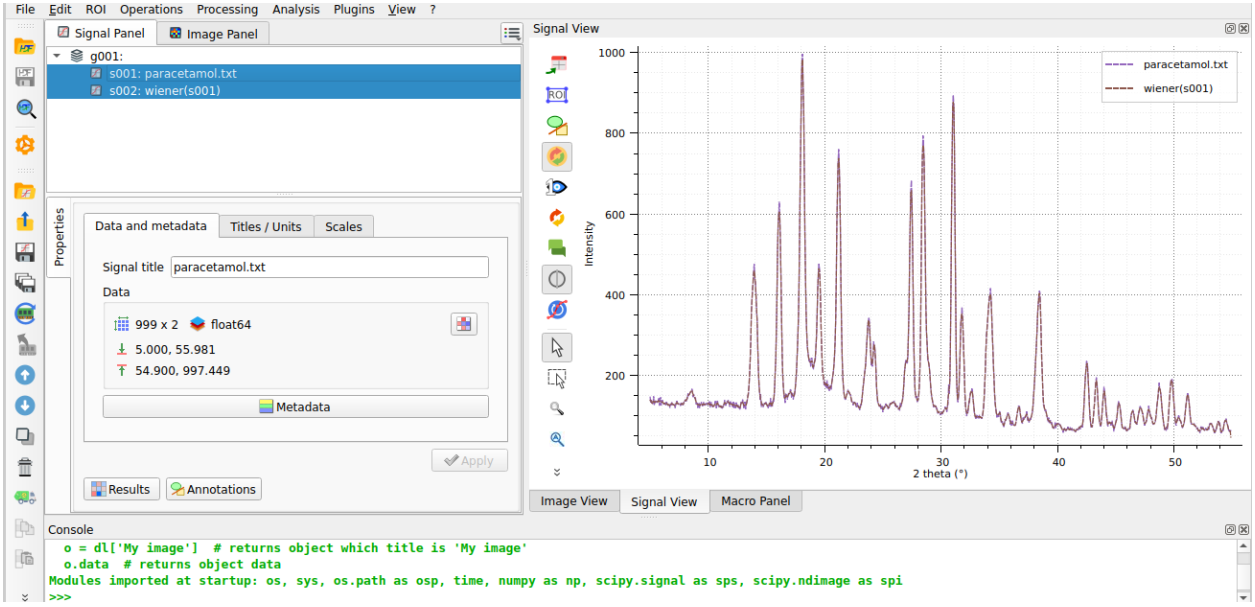


Fig. 10 – Le résultat affiché dans la fenêtre principale.

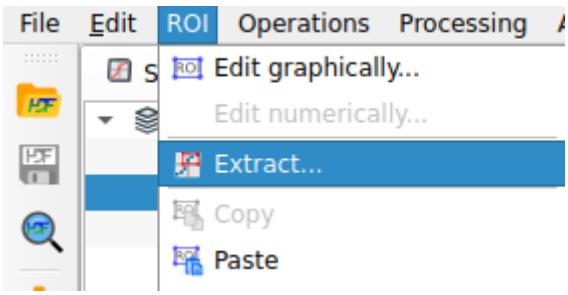


Fig. 11 – Le menu « Opérations > Extraction de ROI ».

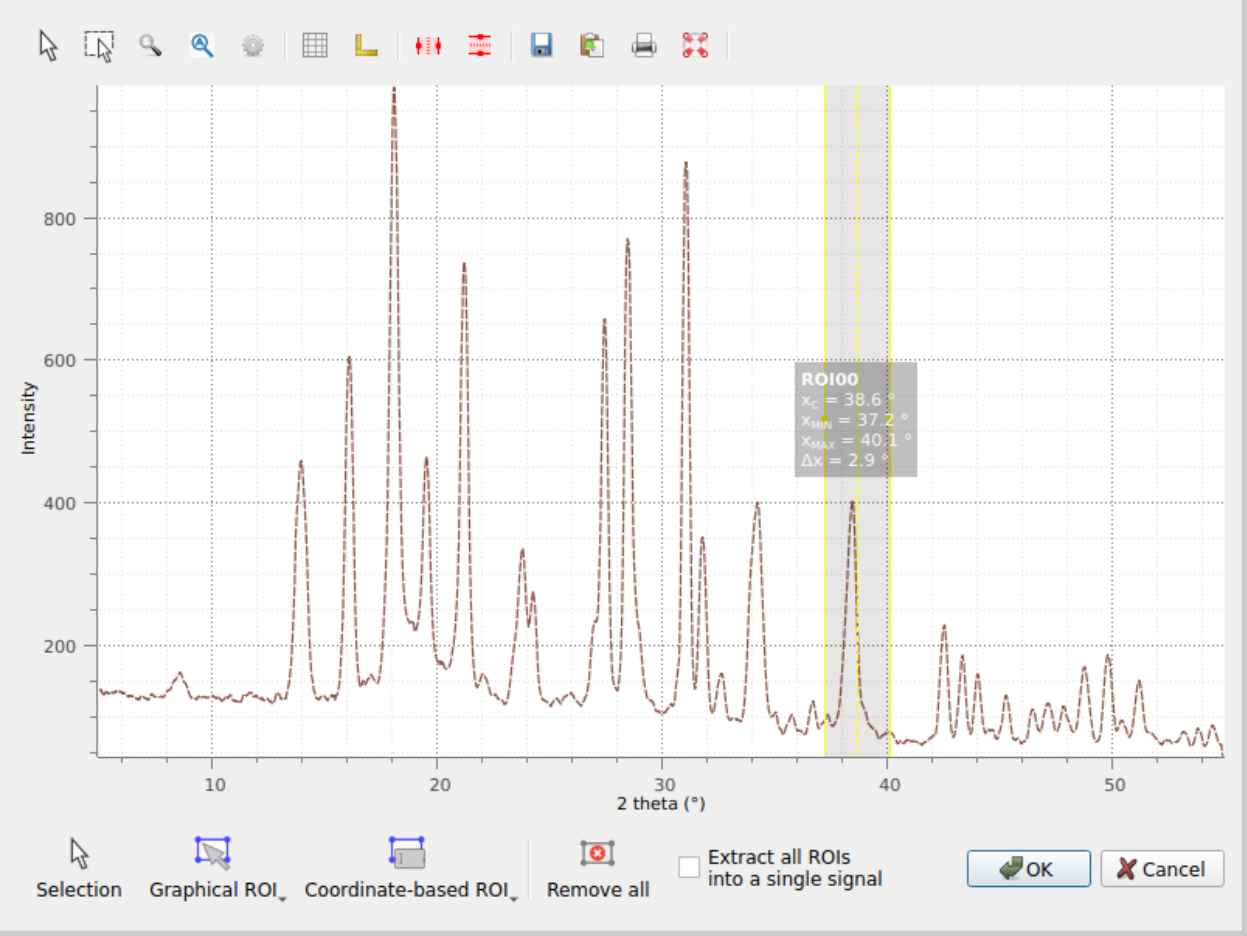


Fig. 12 – La boîte de dialogue « Régions d'intérêt » affichée.

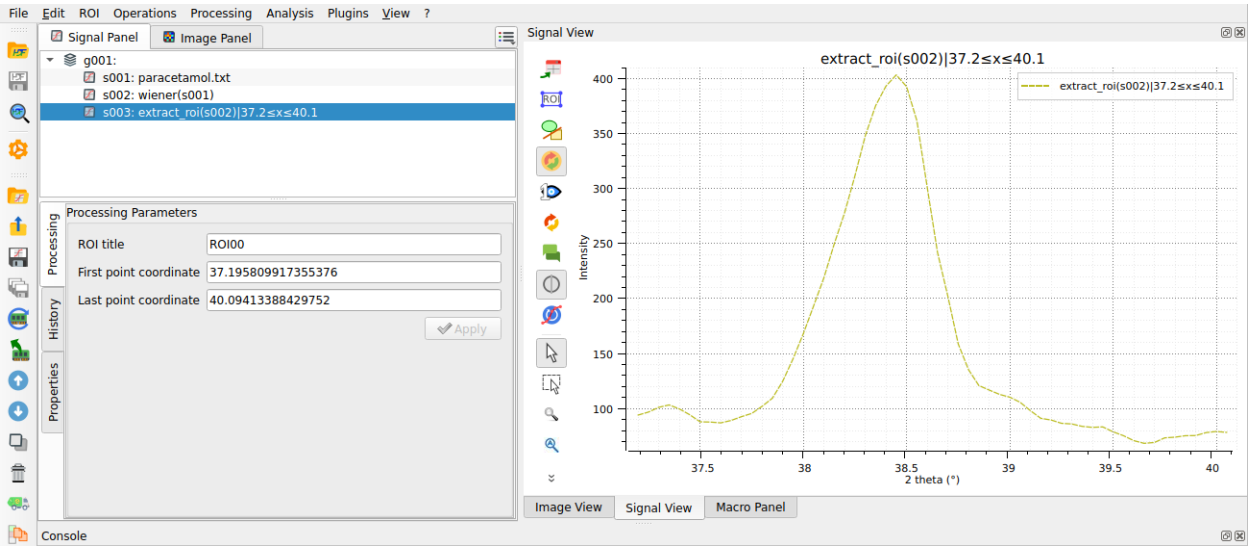


Fig. 13 – La région d'intérêt affichée dans la fenêtre principale.

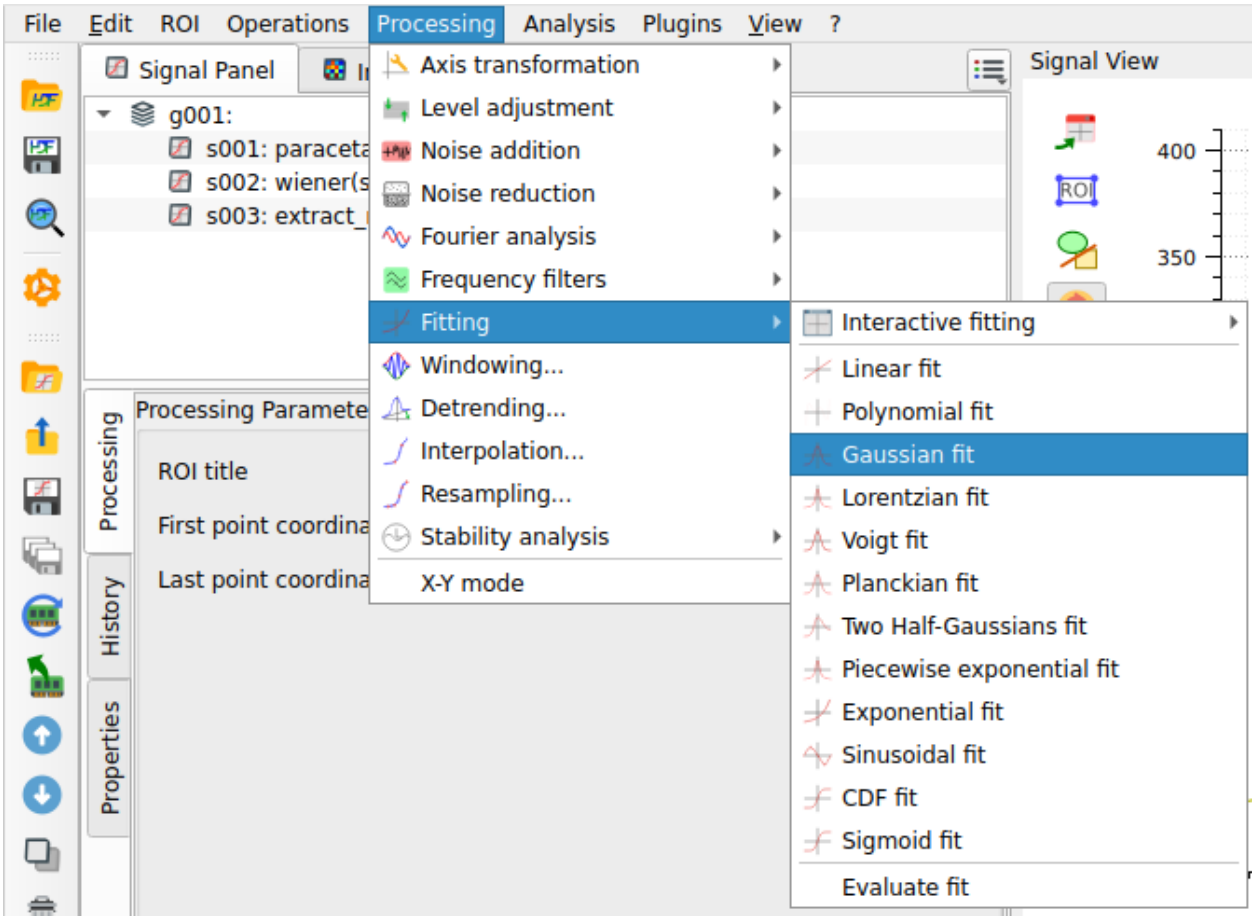


Fig. 14 – Ouvrir la fenêtre d'ajustement du modèle avec « Traitement > Ajustement > Ajustement gaussien ».

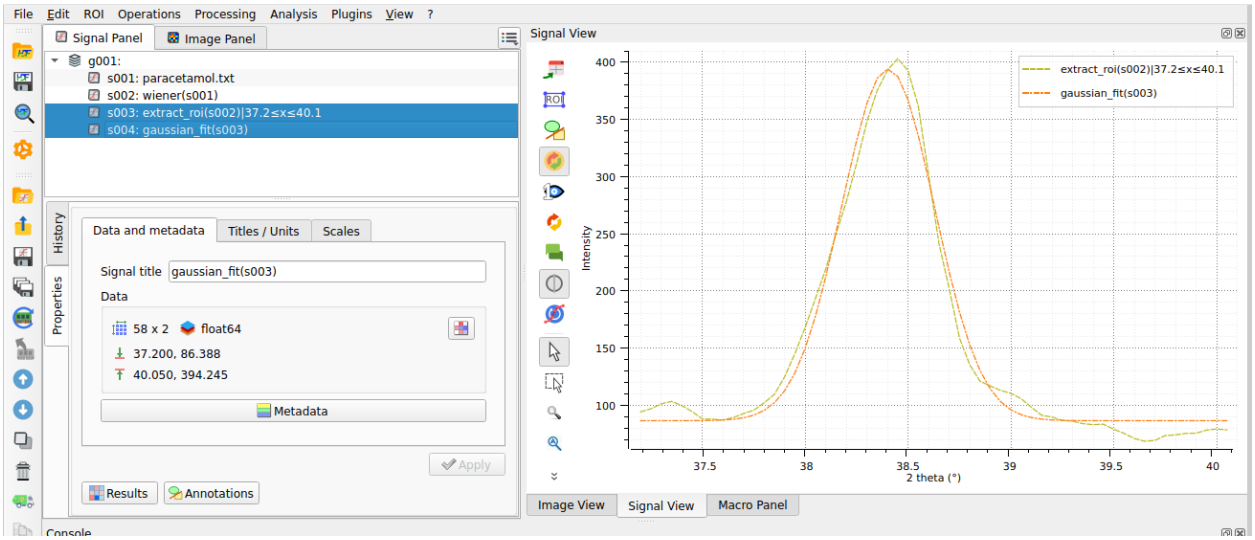


Fig. 15 – Le résultat de l'ajustement affiché dans la fenêtre principale.

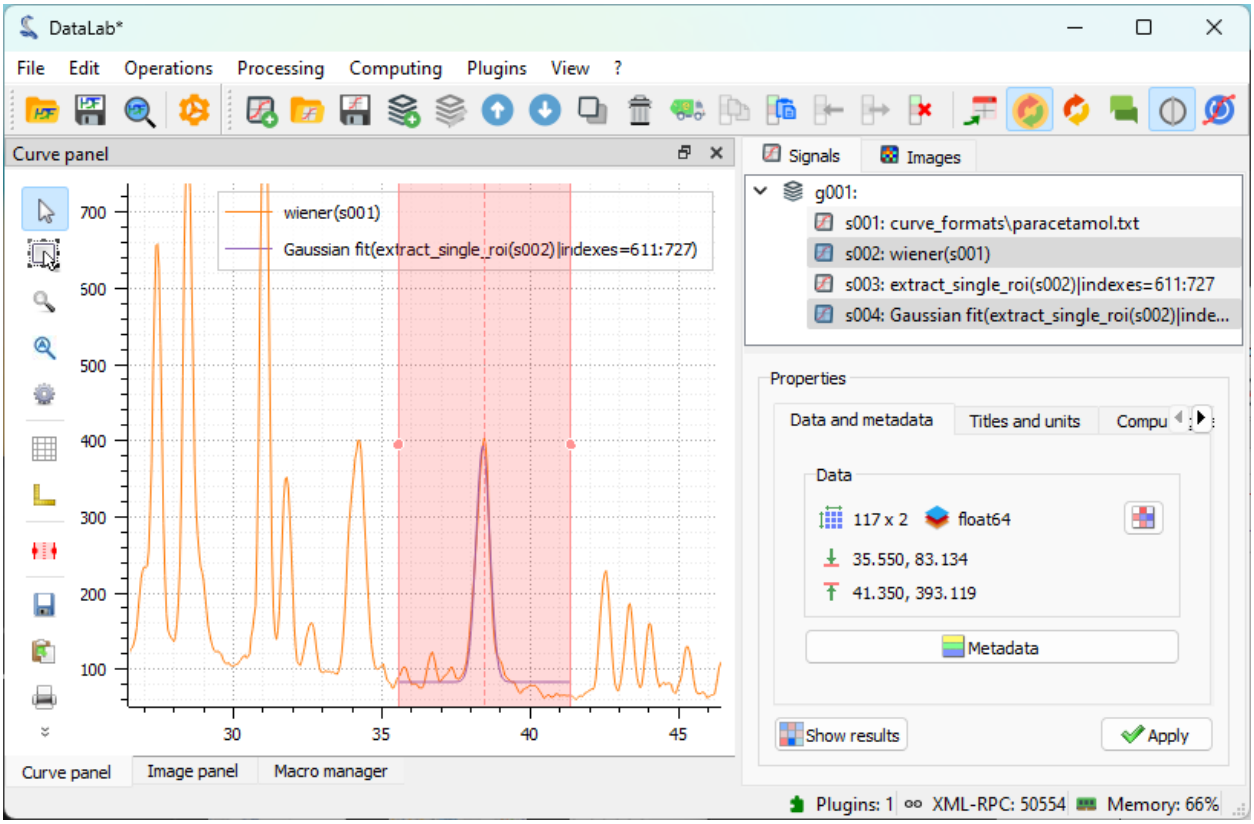


Fig. 16 – Le spectre complet et l'ajustement sont tous deux sélectionnés dans le panneau « Signaux », de sorte que les deux sont affichés dans le panneau de visualisation à gauche. Cela permet une comparaison visuelle facile si nécessaire pour l'analyse.

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.

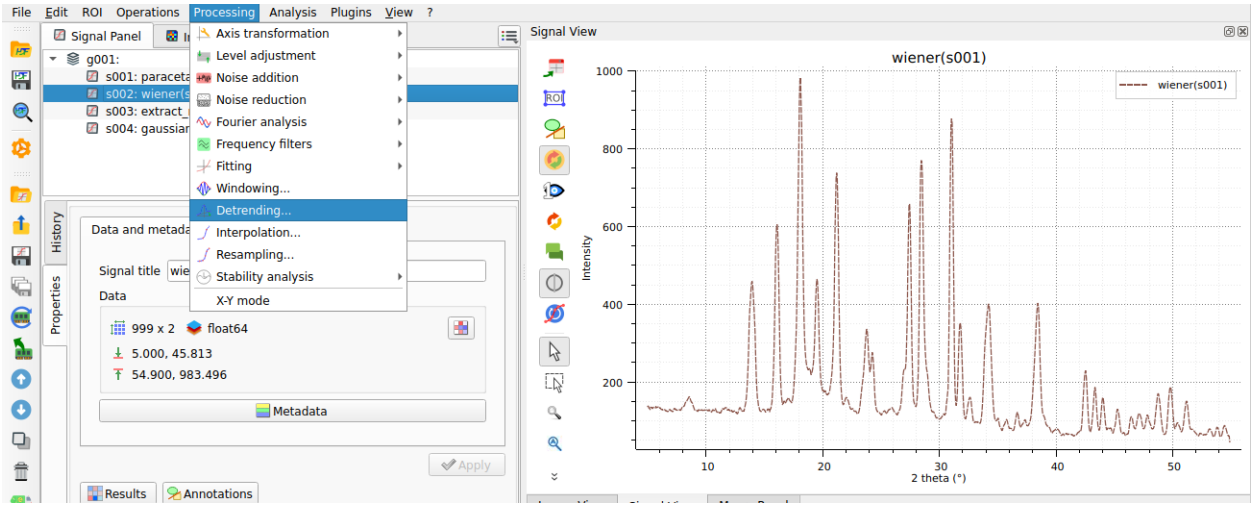


Fig. 17 – La fonction « Traitement > Élimination de tendance ».

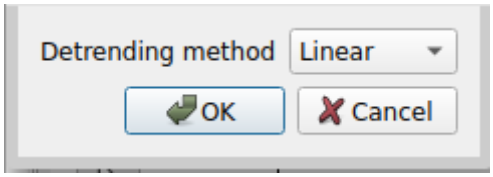


Fig. 18 – Nous choisissons une méthode d'élimination de tendance linéaire, et nous cliquons sur « OK ».

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

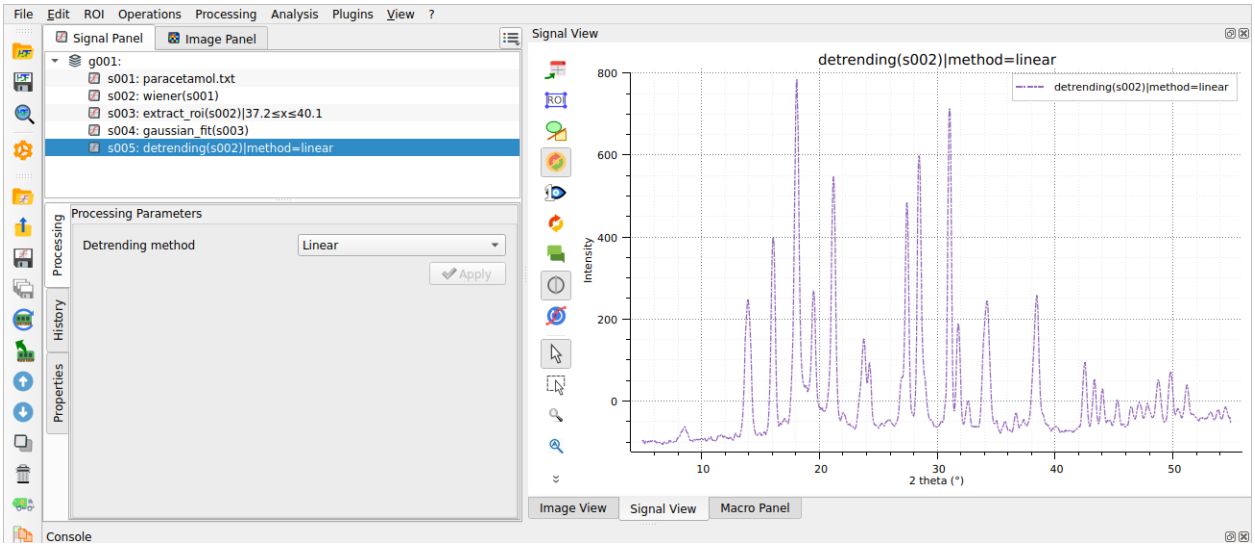


Fig. 19 – Le résultat de l'élimination de tendance affiché dans la fenêtre principale.

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 :

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. C'est la première étape de la fonction « Ajustement multi-gaussien » et peut être utilisée indépendamment pour extraire les positions des pics sous forme d'un tableau de résultats attaché au signal.

Pour reconstruire le signal historique sticks — une fonction delta à chaque position de pic détectée — appliquez Opérations Créer un signal à partir de la table des marqueurs… au spectre : cela crée un signal enfant dont les échantillons sont les couples `(x, y)` de la table des marqueurs, rendus sous forme de bâtons.

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).

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

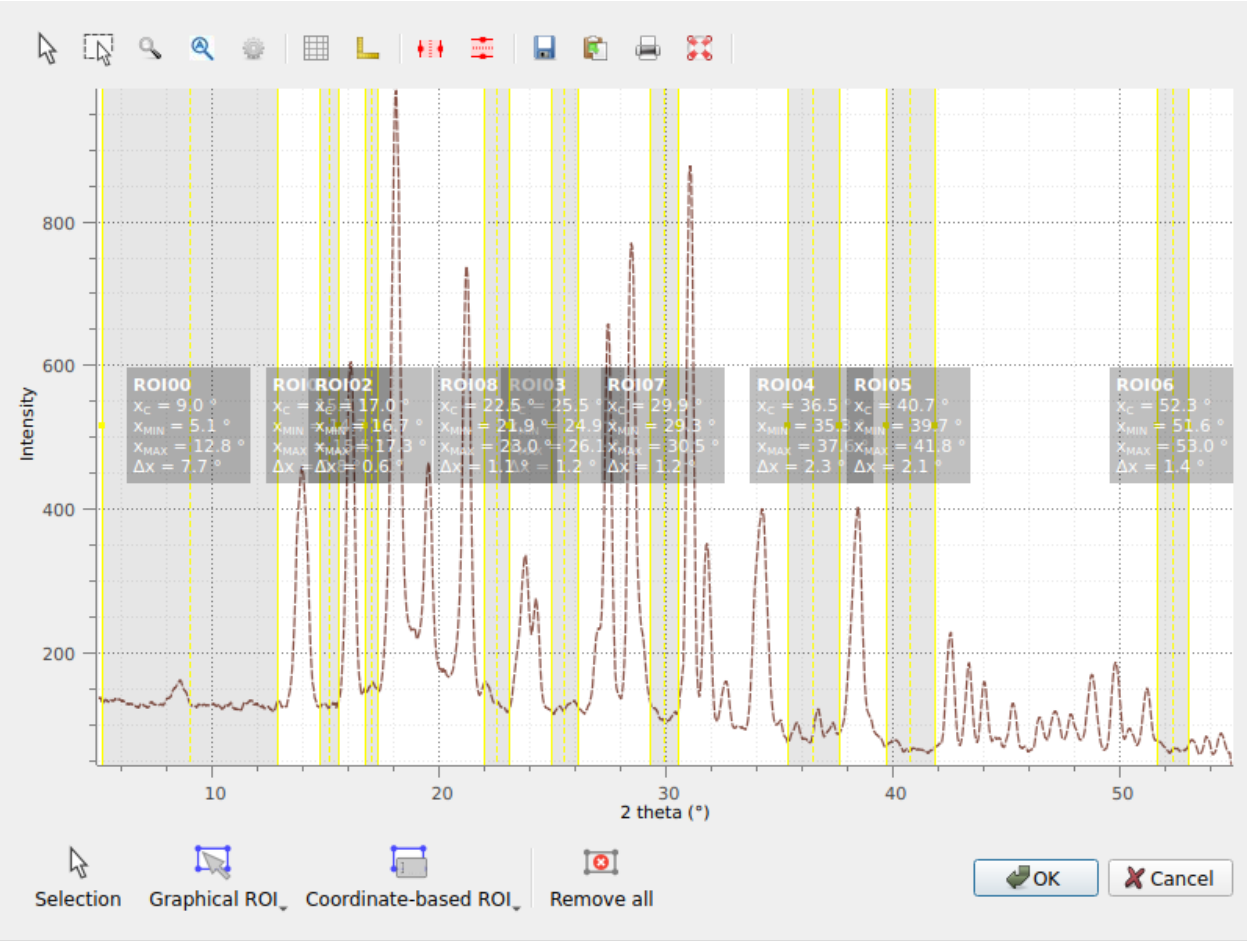


Fig. 20 – Nous sélectionnons les régions correspondant aux régions sans pics.

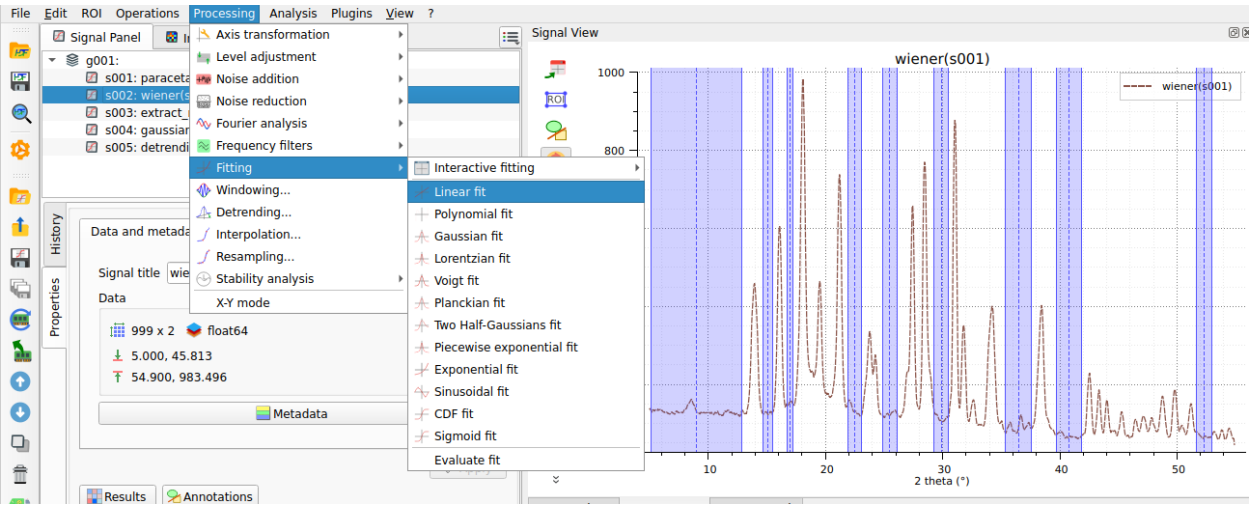


Fig. 21 – Un ajustement linéaire est effectué uniquement sur les régions sélectionnées.

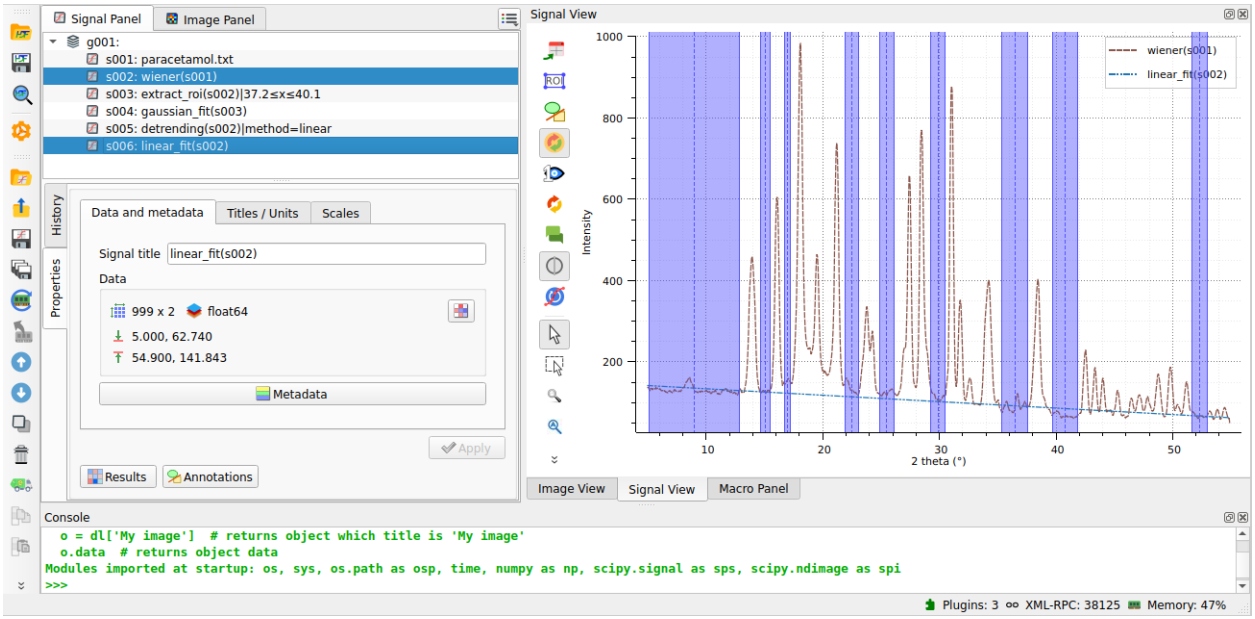


Fig. 22 – La ligne de base linéaire obtenue à partir de l'ajustement est affichée.

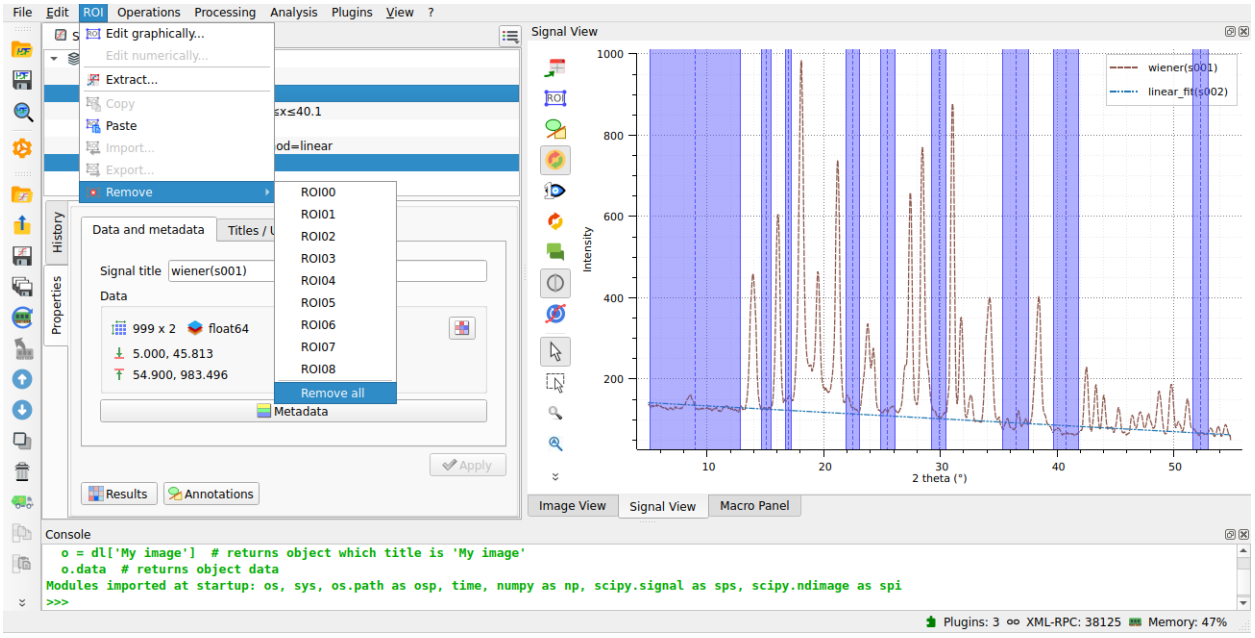


Fig. 23 – Nous supprimons les ROIs pour appliquer l'élimination de tendance sur l'ensemble du signal.

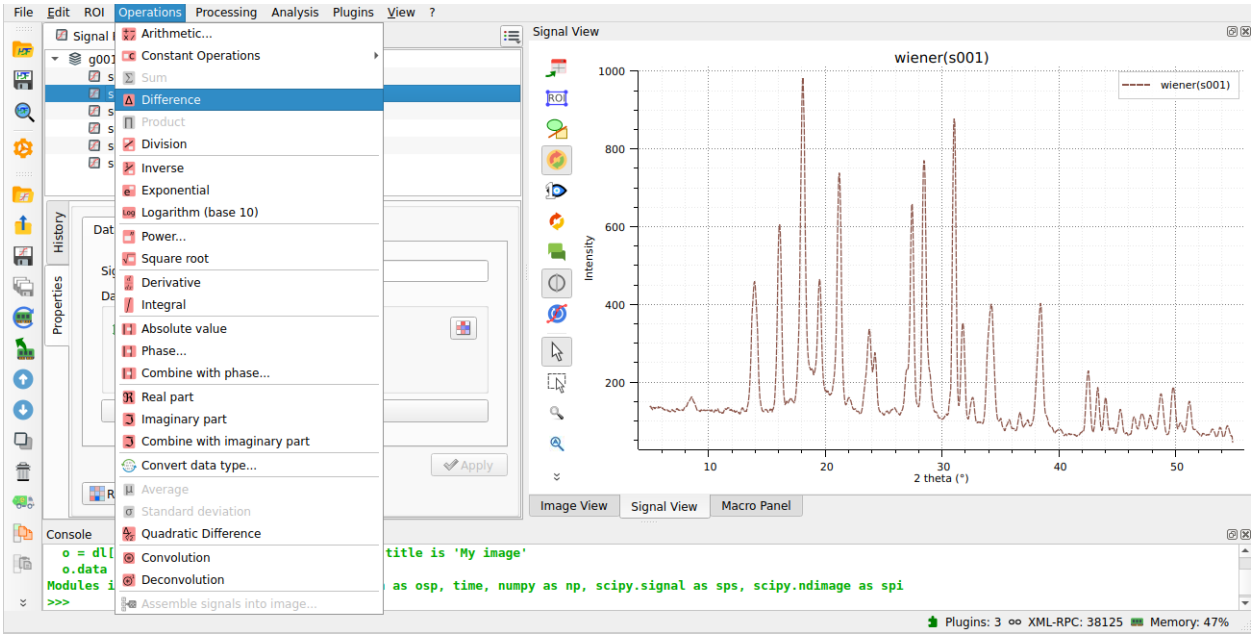


Fig. 24 – Nous utilisons l'opération « différence » pour soustraire la ligne de base du signal d'origine.

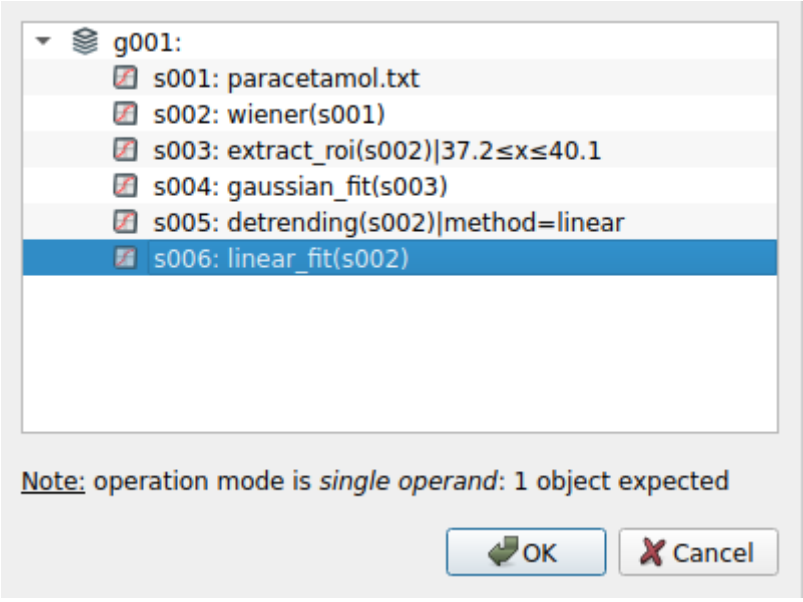


Fig. 25 – Nous sélectionnons la ligne de base linéaire pour effectuer la soustraction.

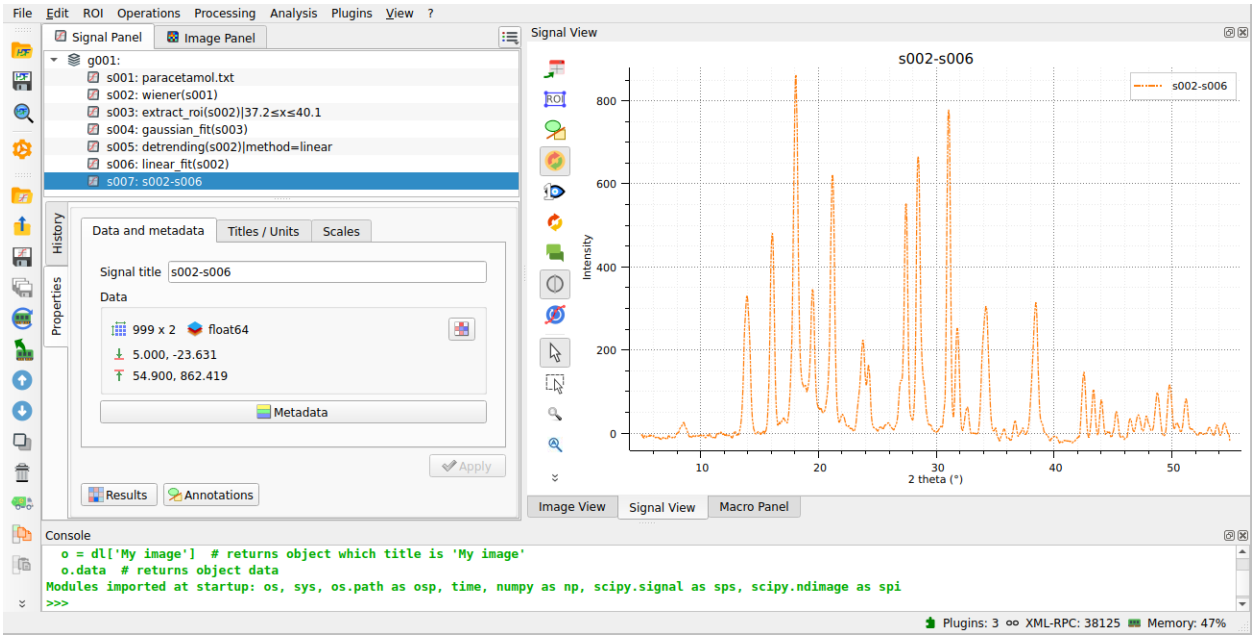


Fig. 26 – Le signal sans tendance résultant est affiché, maintenant avec une ligne de base correcte.

Détection automatisée de défauts et de particules sur des images

Cet exemple montre comment détecter automatiquement des éléments de type tache – défauts, particules, points – sur une image avec DataLab, et couvre également d'autres fonctionnalités telles que le système de plugins :

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

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

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

Ajoutons le plugin 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) ».

> **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 ».

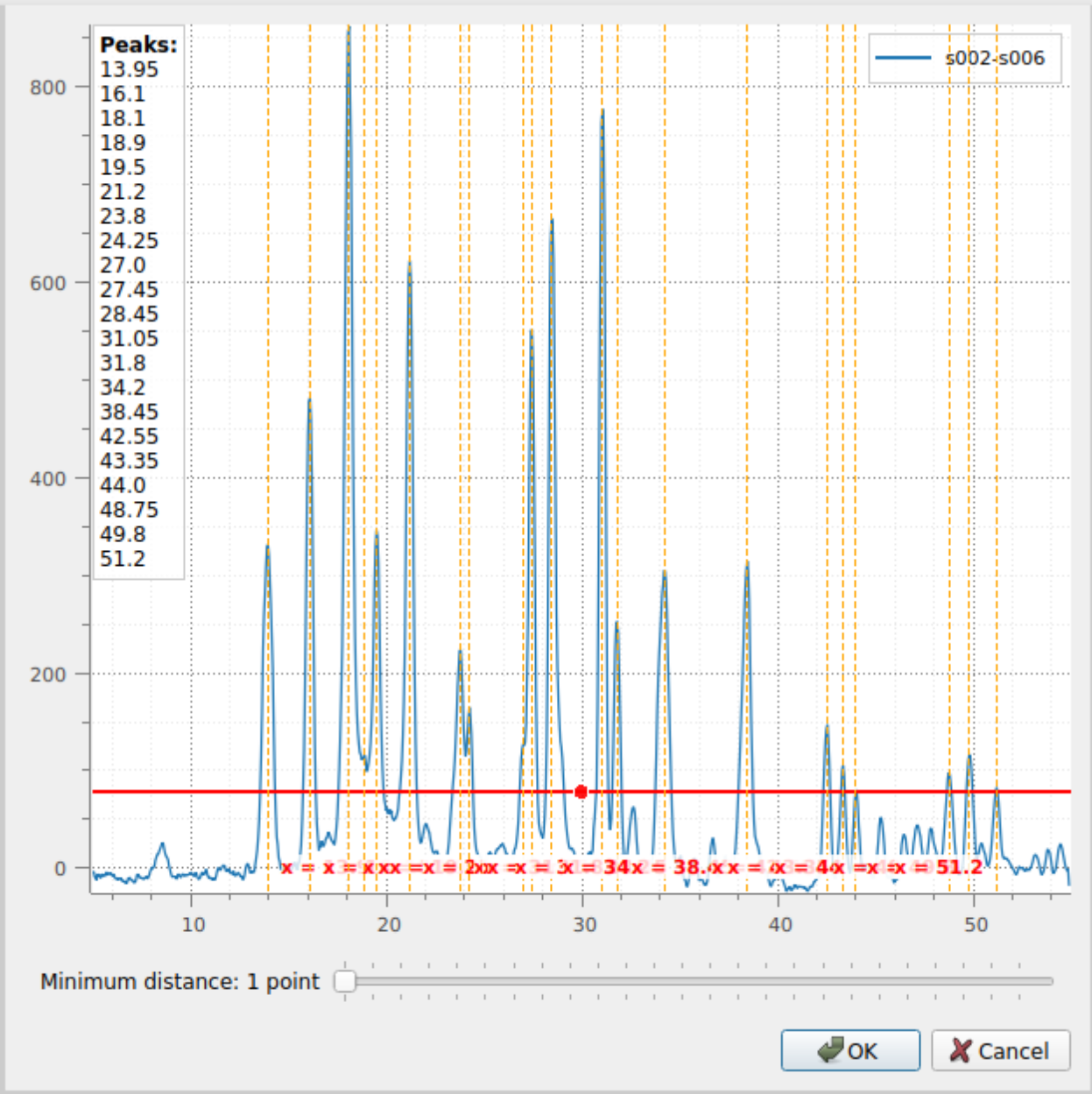


Fig. 27 – Tout d'abord, la boîte de dialogue « Détection de pics de signal » apparaît. Vous pouvez ajuster la position du curseur vertical pour définir le seuil de détection des pics et spécifier la distance minimale entre les pics. Ensuite, cliquez sur « OK ».

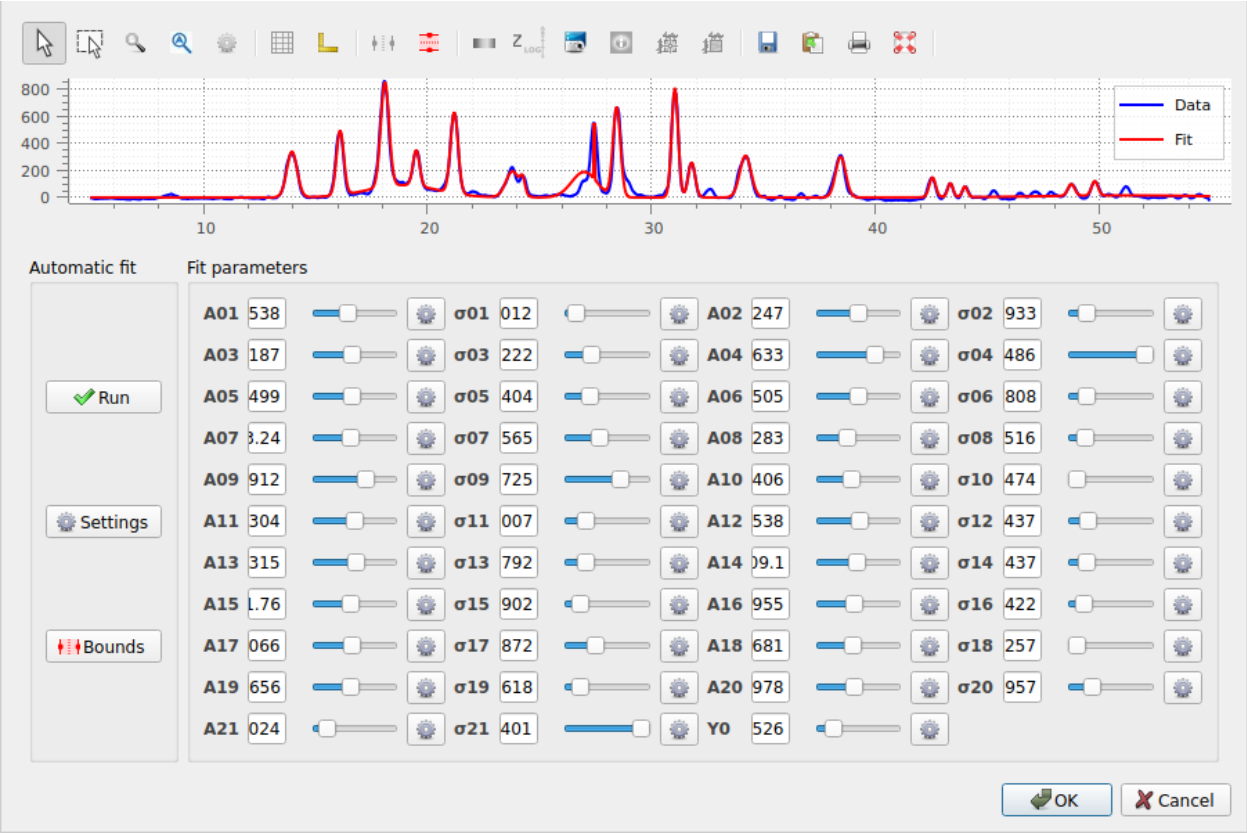


Fig. 28 – La boîte de dialogue « Ajustement multi-gaussien » apparaît. Un ajustement automatique est effectué par défaut. Cliquez sur « OK » (ou ajustez éventuellement les paramètres manuellement à l'aide des curseurs, ou modifiez les paramètres d'ajustement automatique).

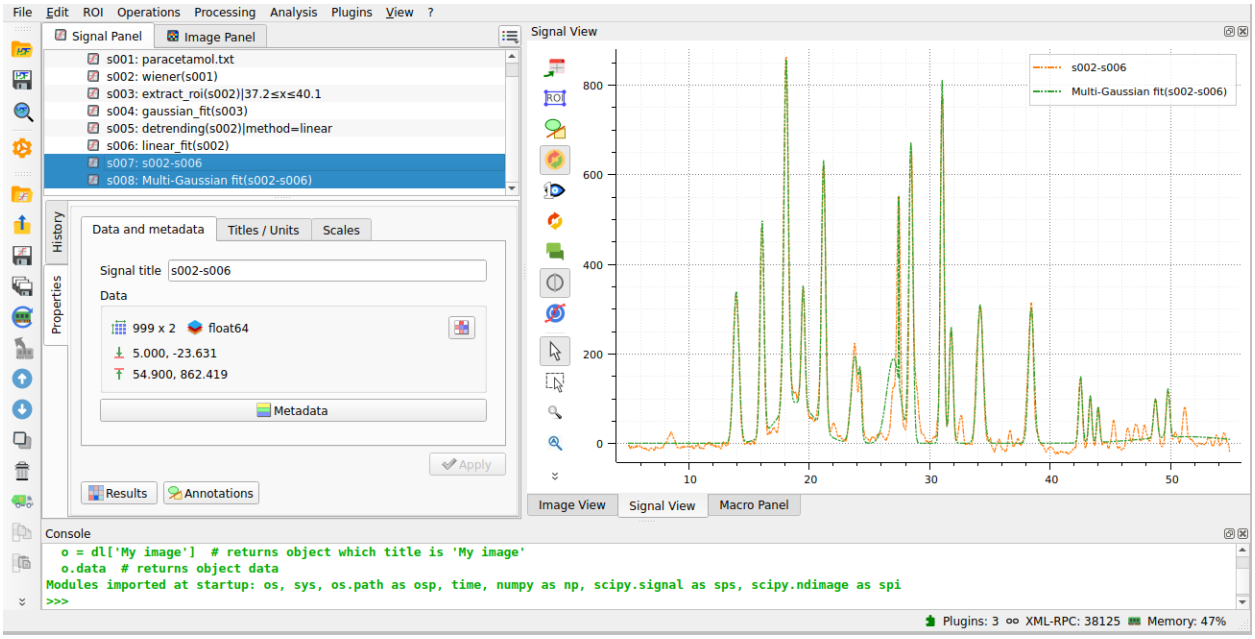


Fig. 29 – Le résultat de l'ajustement est affiché dans la fenêtre principale. Ici, nous avons sélectionné à la fois le spectre et l'ajustement dans le panneau « Signaux » à droite, donc les deux sont affichés dans le panneau de visualisation à gauche.

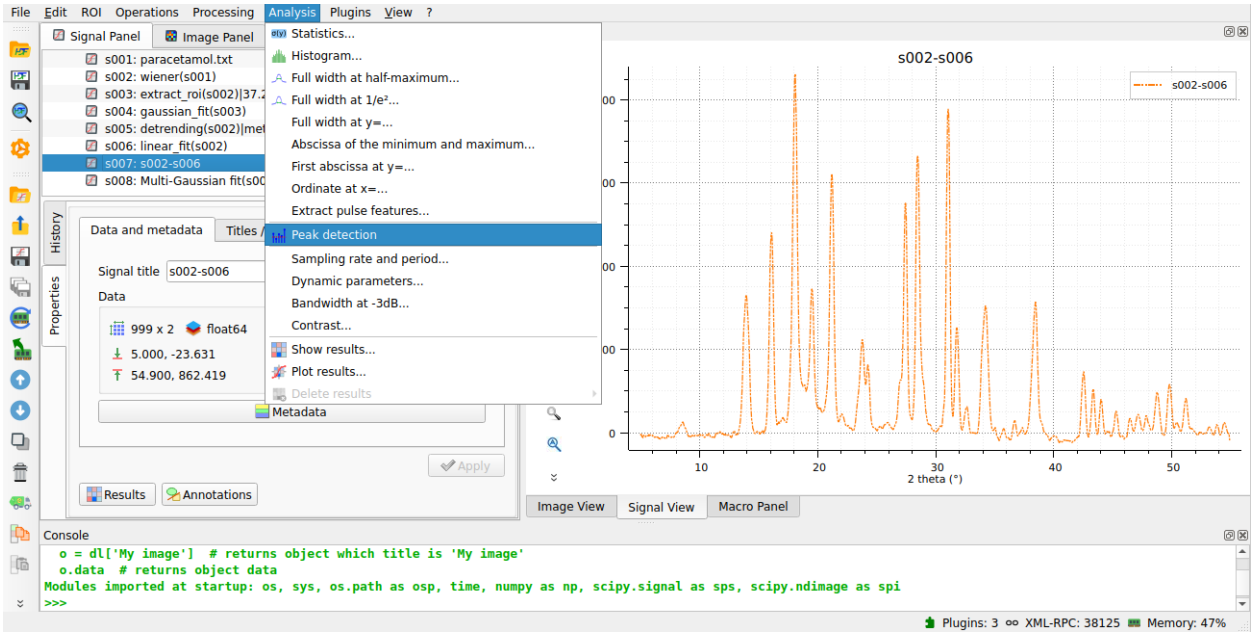


Fig. 30 – Ouvrir la fenêtre « Détection de pics » avec « Analyse > Détection de pics ».

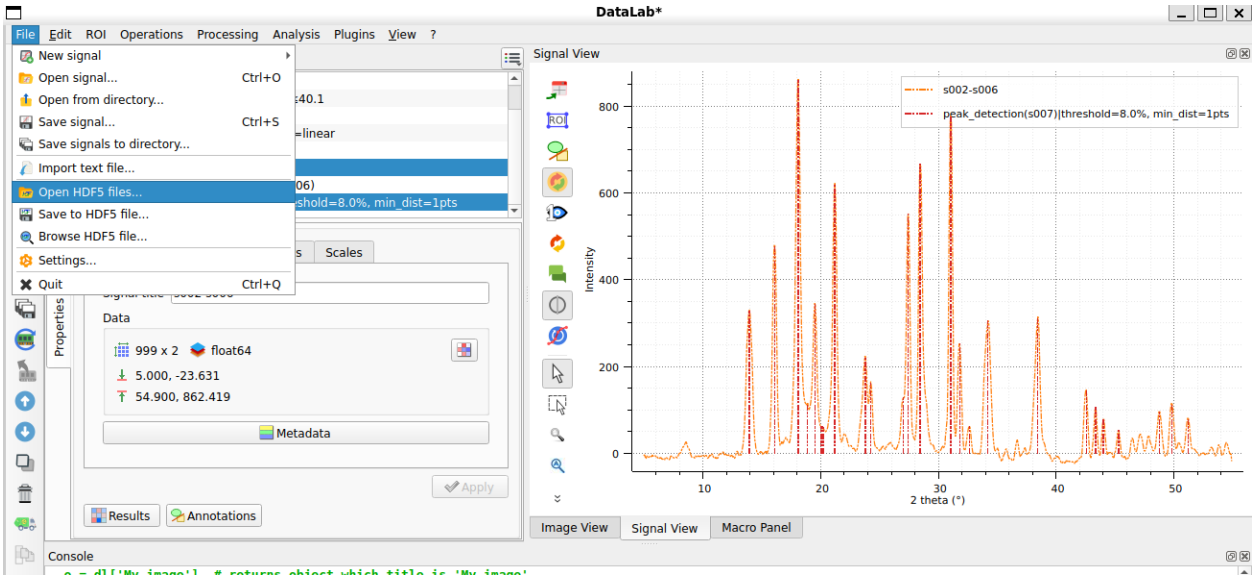


Fig. 31 – Après avoir ajusté les paramètres de détection des pics (en utilisant la même boîte de dialogue que pour l'ajustement multi-gaussien), cliquez sur « OK ». Les positions des pics détectés sont stockées sous forme de résultat XY-markers et superposées au spectre.

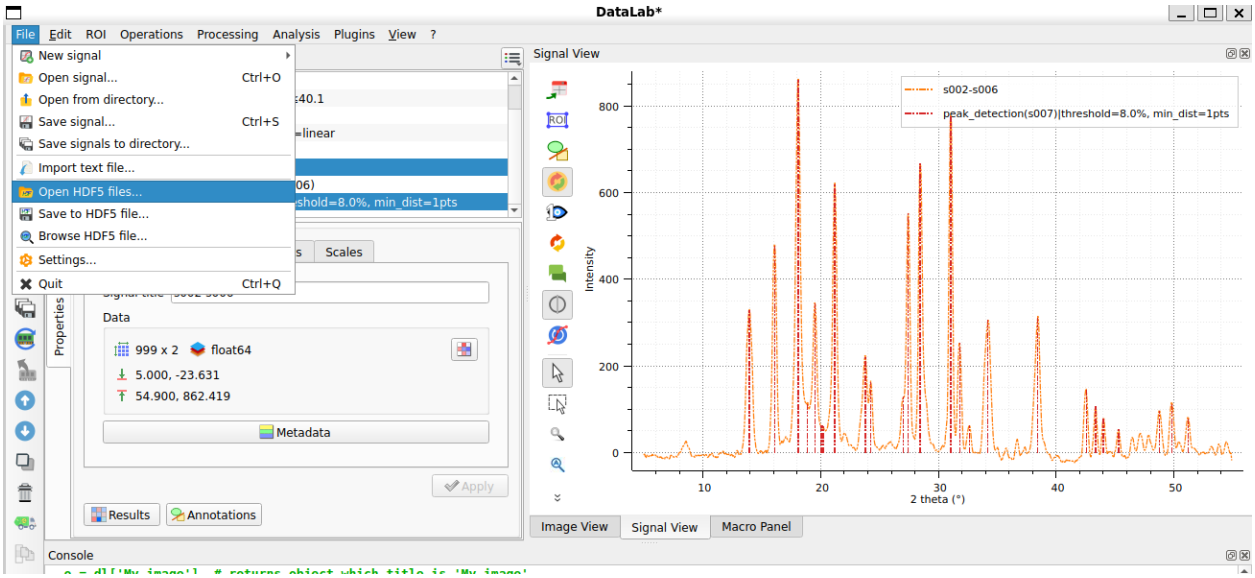


Fig. 32 – Sauvegarder l'espace de travail dans un fichier avec « Fichier > Sauvegarder dans un fichier HDF5… », ou le bouton  dans la barre d'outils.

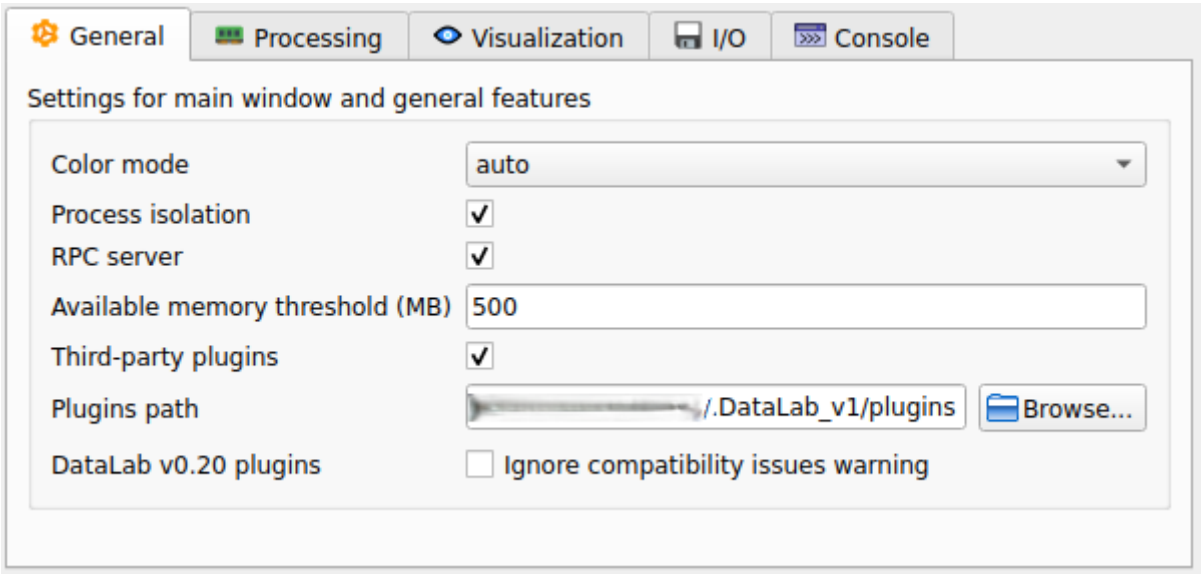


Fig. 33 – Dans l'onglet « Général », nous pouvons voir le champ « Chemin des plugins ». C'est le chemin où DataLab recherchera les plugins. Nous pouvons ajouter un nouveau plugin en copiant/collant le fichier du plugin dans ce répertoire.

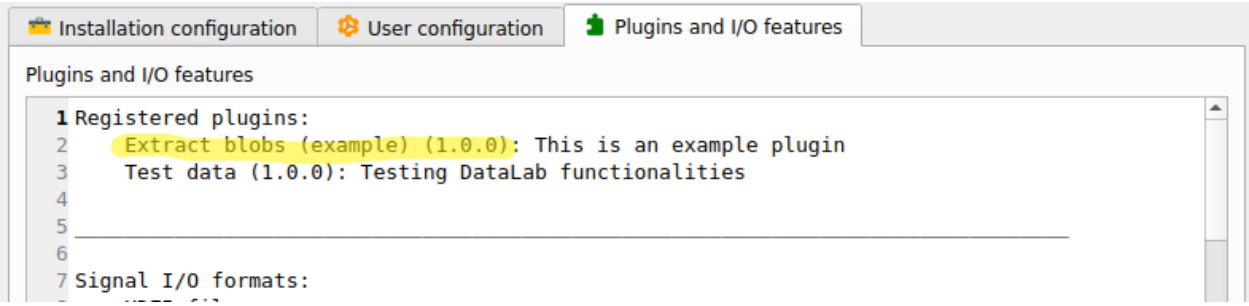


Fig. 34 – La boîte de dialogue « À propos de DataLab » affiche la liste des plugins disponibles.

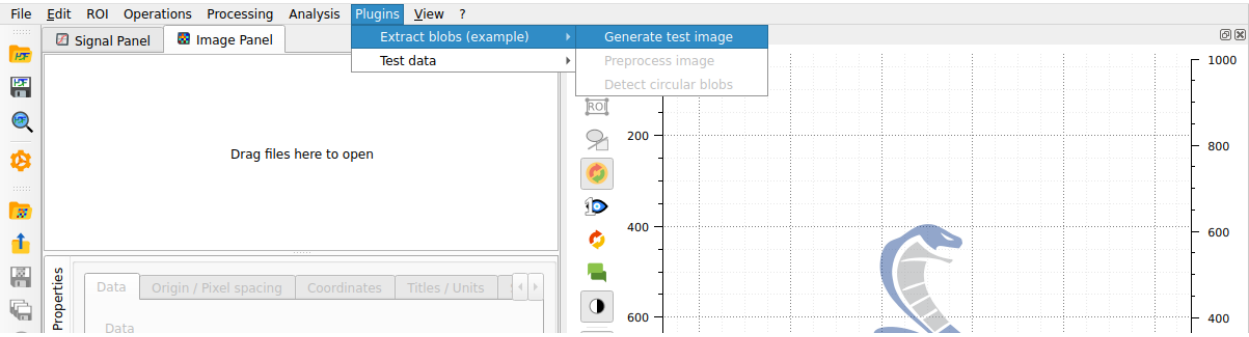


Fig. 35 – Cliquons sur « Extract blobs (example) > Generate test image »

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

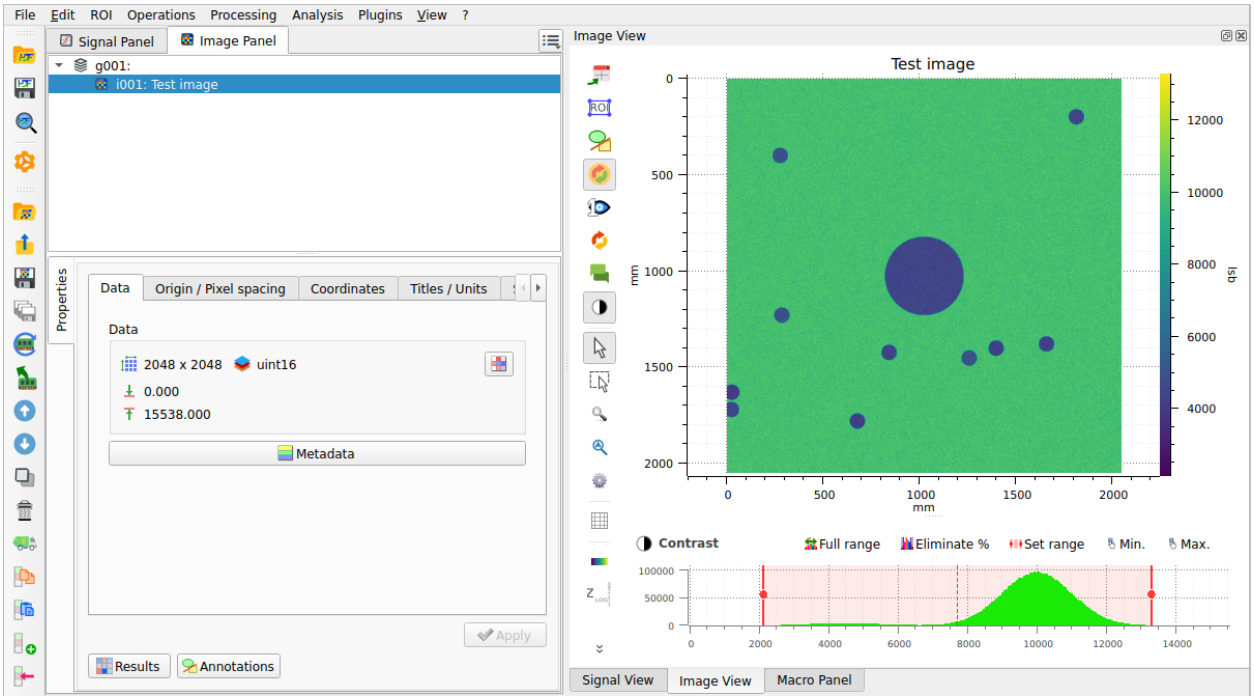


Fig. 36 – Le plugin a généré une image de test, et l'a ajoutée au panneau « Images ». L'image montre quelques taches, avec un disque central plus grand, et un fond bruité.

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 venons 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 :

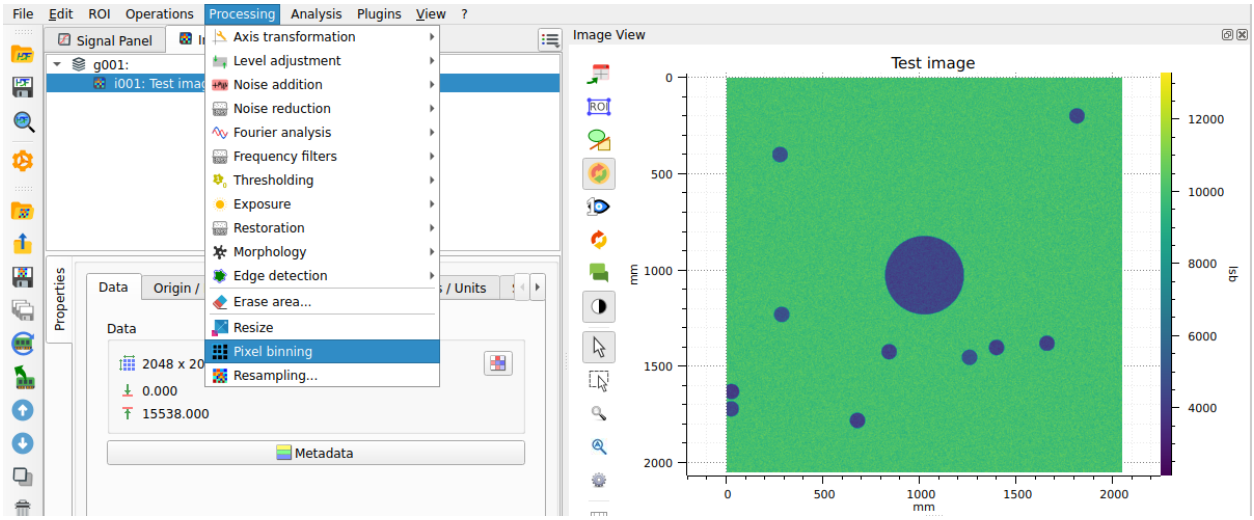


Fig. 37 – Cliquons sur « Opérations > Pixel binning ».

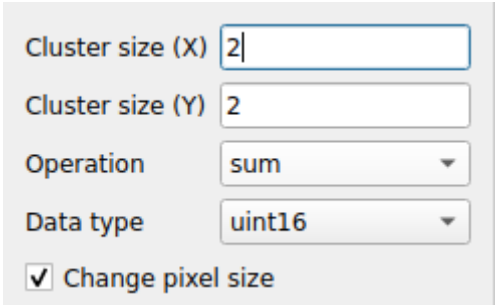


Fig. 38 – La boîte de dialogue « Binning » s'ouvre, avec plusieurs options. Réglez le facteur de binning à 2 pour les deux axes, sélectionnez la case « changer la taille des pixels » et cliquez sur « OK ».

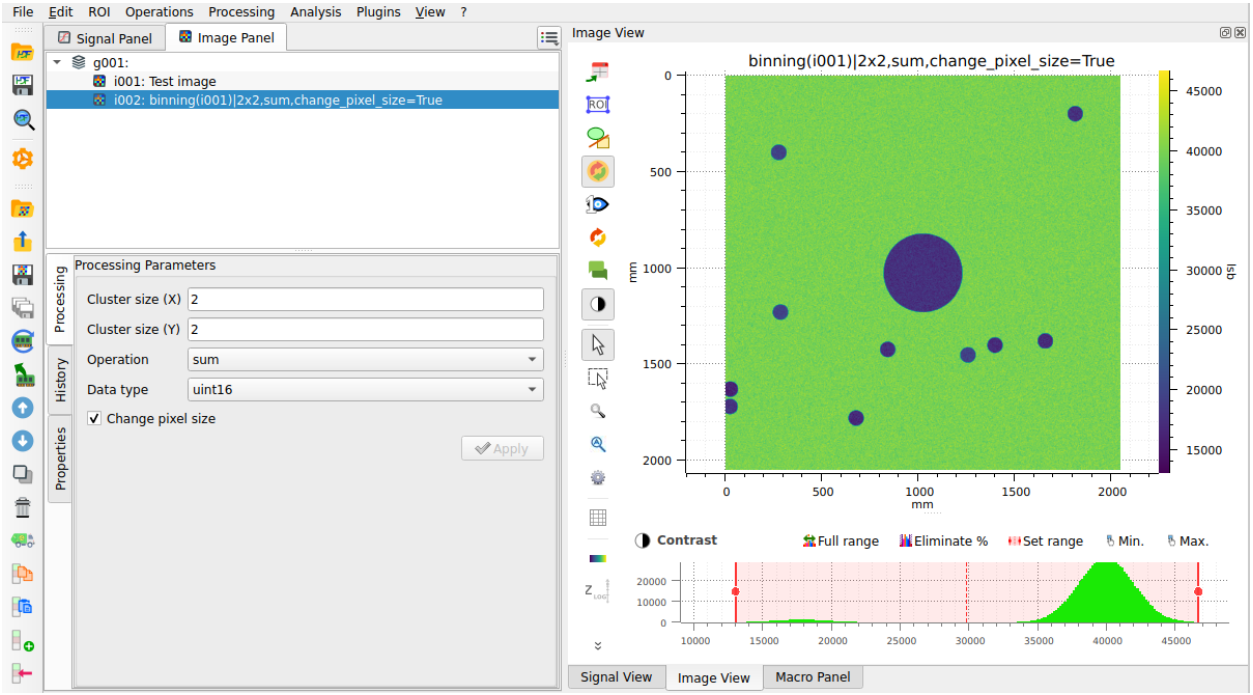


Fig. 39 – L'image binnée est ajoutée au panneau « Images ». Il est maintenant plus facile de voir les taches (même si elles étaient déjà assez visibles sur l'image d'origine : c'est juste un exemple), et l'image sera plus rapide à traiter.

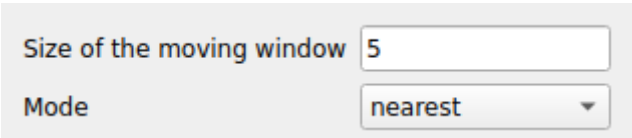


Fig. 40 – Cliquez sur l'entrée « Traitement > Réduction du bruit > Médiane mobile », et réglez la taille de la fenêtre sur `5`.

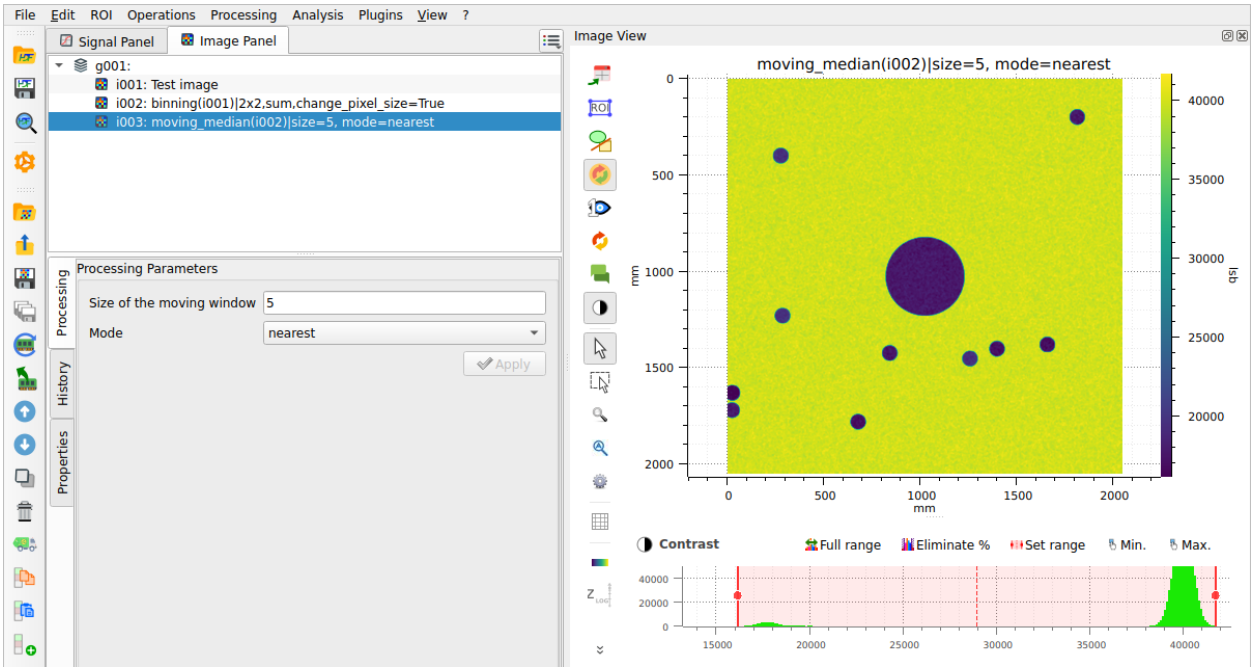


Fig. 41 – L'image filtrée est ajoutée au panneau « Images ». Le débruitage est assez efficace.

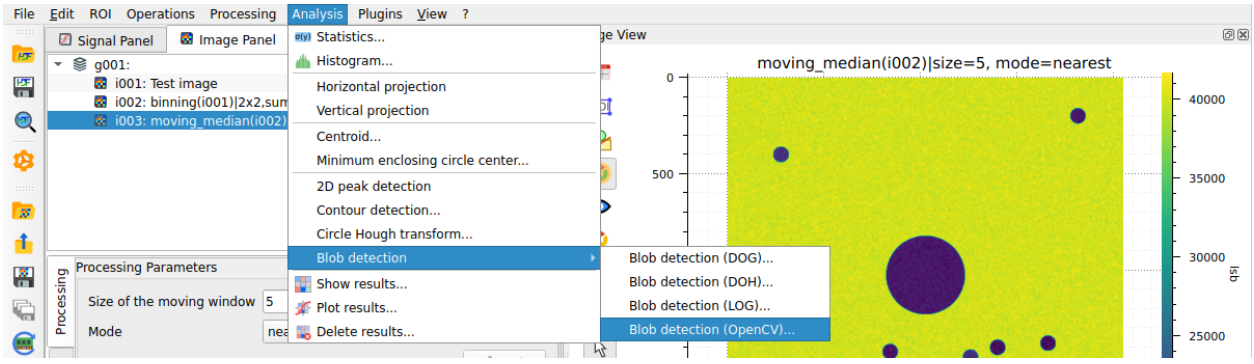


Fig. 42 – Cliquons sur « Analysis > Blob detection > Blob detection (OpenCV) ».

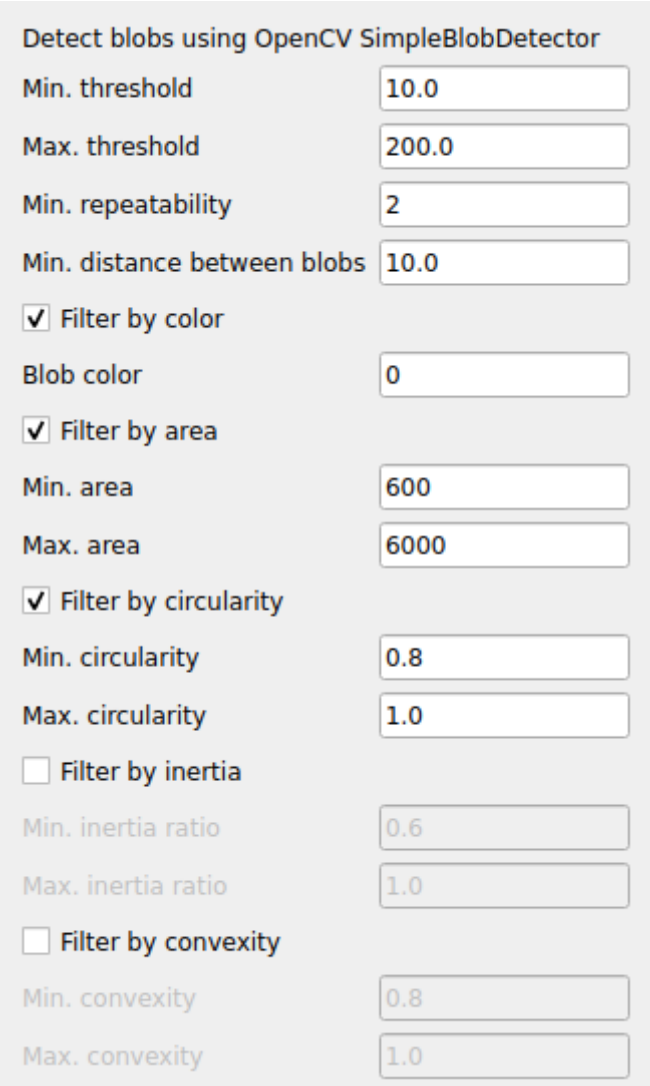


Fig. 43 – La boîte de dialogue « Blob detection (OpenCV) » s'ouvre. Réglez les paramètres comme indiqué sur la capture d'écran, et cliquez sur « OK ».

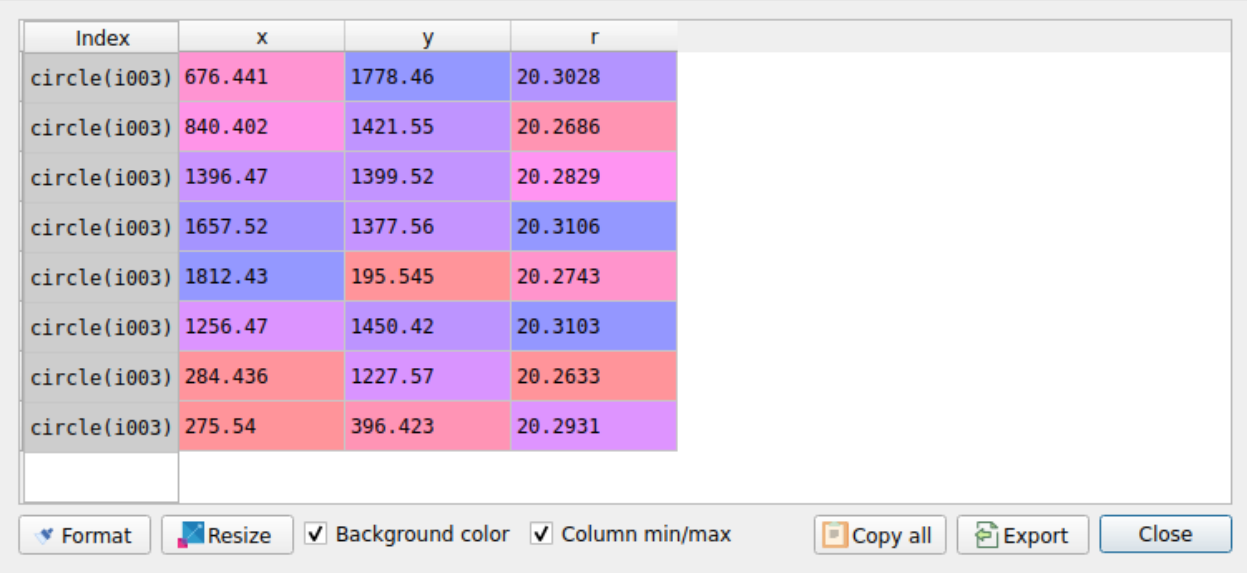
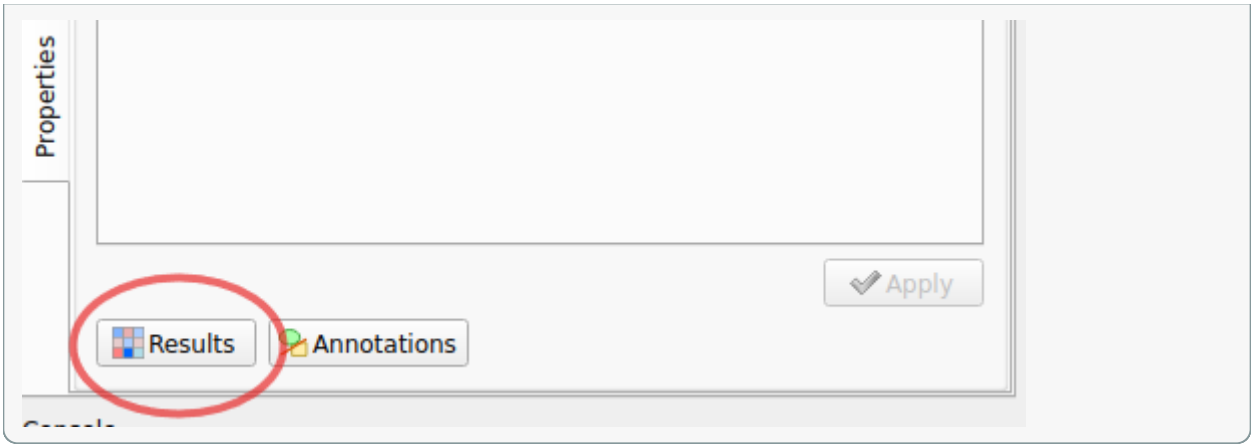


Fig. 44 – La boîte de dialogue « Résultats » s'ouvre, montrant les taches détectées : une ligne par tache, avec les coordonnées et le rayon de la tache.



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.

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

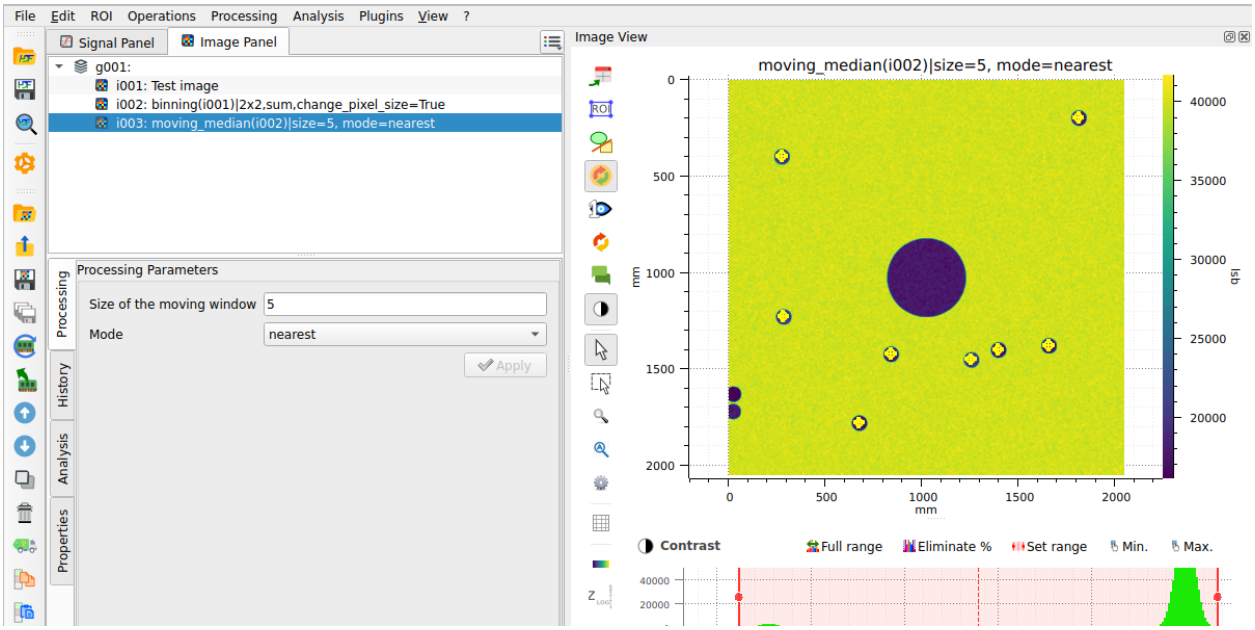


Fig. 45 – Les taches détectées sont également ajoutées aux métadonnées de l'image, et peuvent être visualisées dans le panneau de visualisation à droite.

Analyse d'interférogrammes Fabry-Perot : détection d'anneaux et ajustement de cercles

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

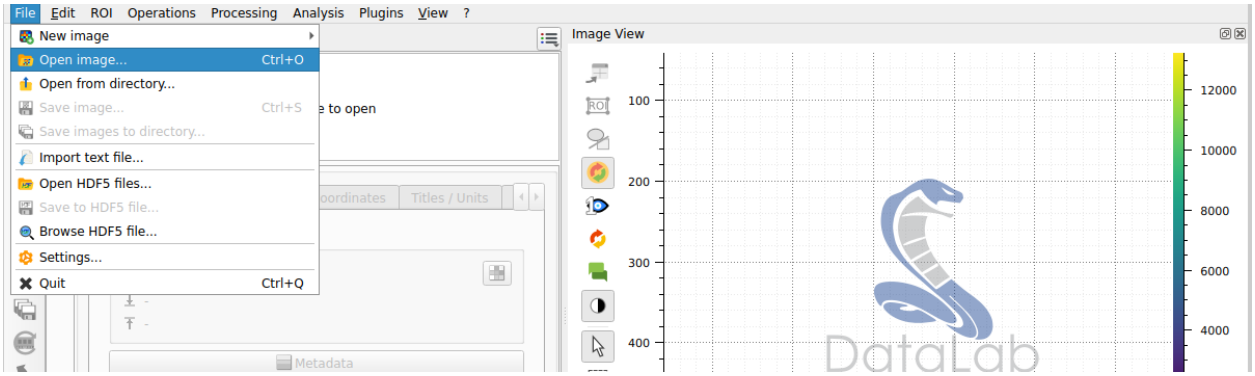


Fig. 46 – Ouvrez les fichiers d'image avec « Fichier > Ouvrir une image… », ou avec le bouton dans la barre d'outils, ou en faisant glisser-déposer les fichiers dans DataLab (sur le panneau de droite).

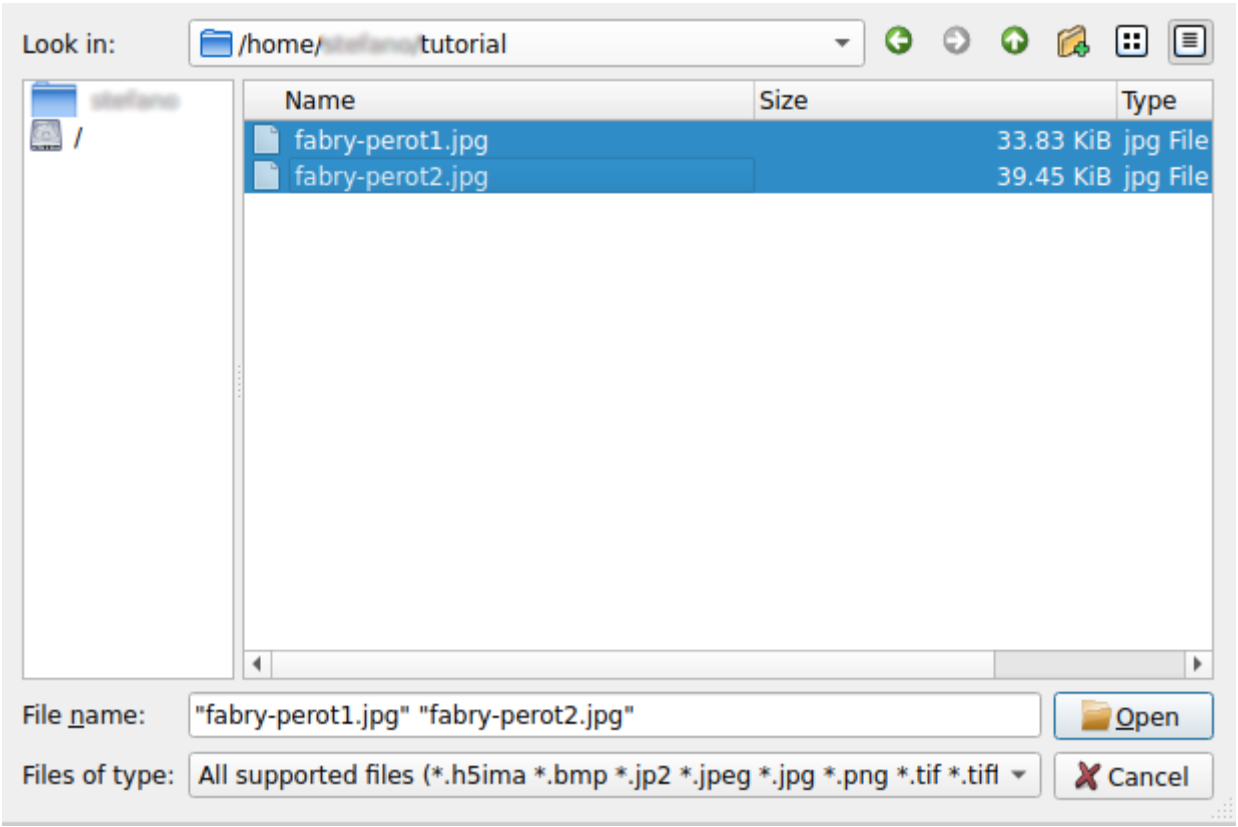


Fig. 47 – Allez dans le dossier de données du tutoriel dans le répertoire d'installation de DataLab (ou le dossier où vous avez placé les images), sélectionnez « fabry_perot1.jpg » et « fabry_perot2.jpg » et cliquez sur « Ouvrir ».

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

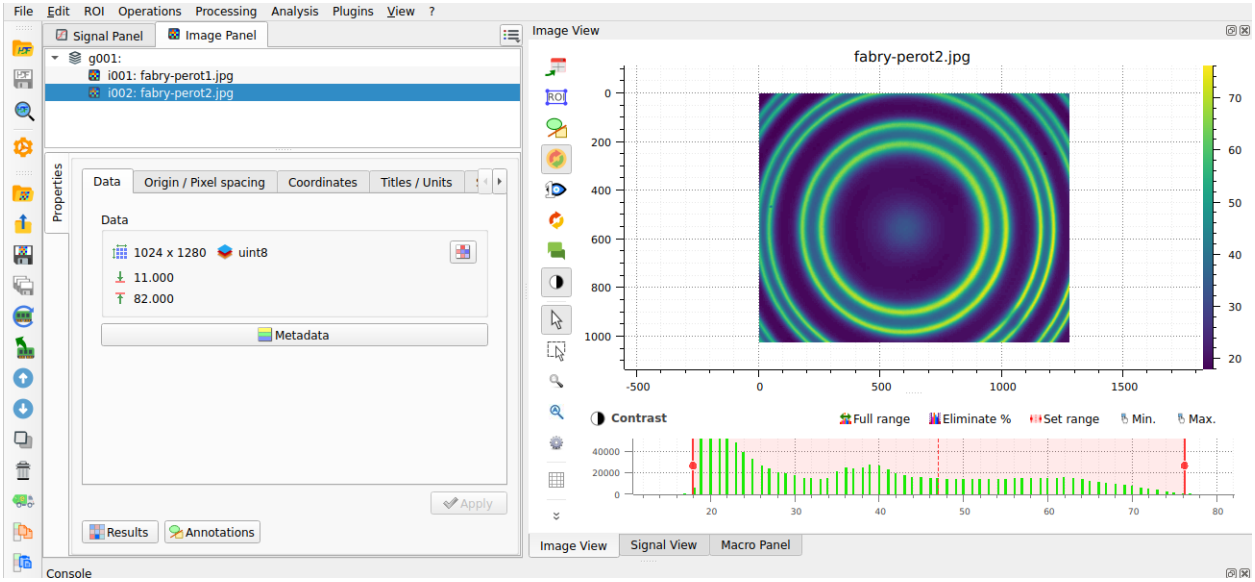


Fig. 48 – Images chargées dans DataLab. Zoomez avec le bouton droit de la souris, déplacez l'image avec le bouton du milieu de la souris.

> **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.

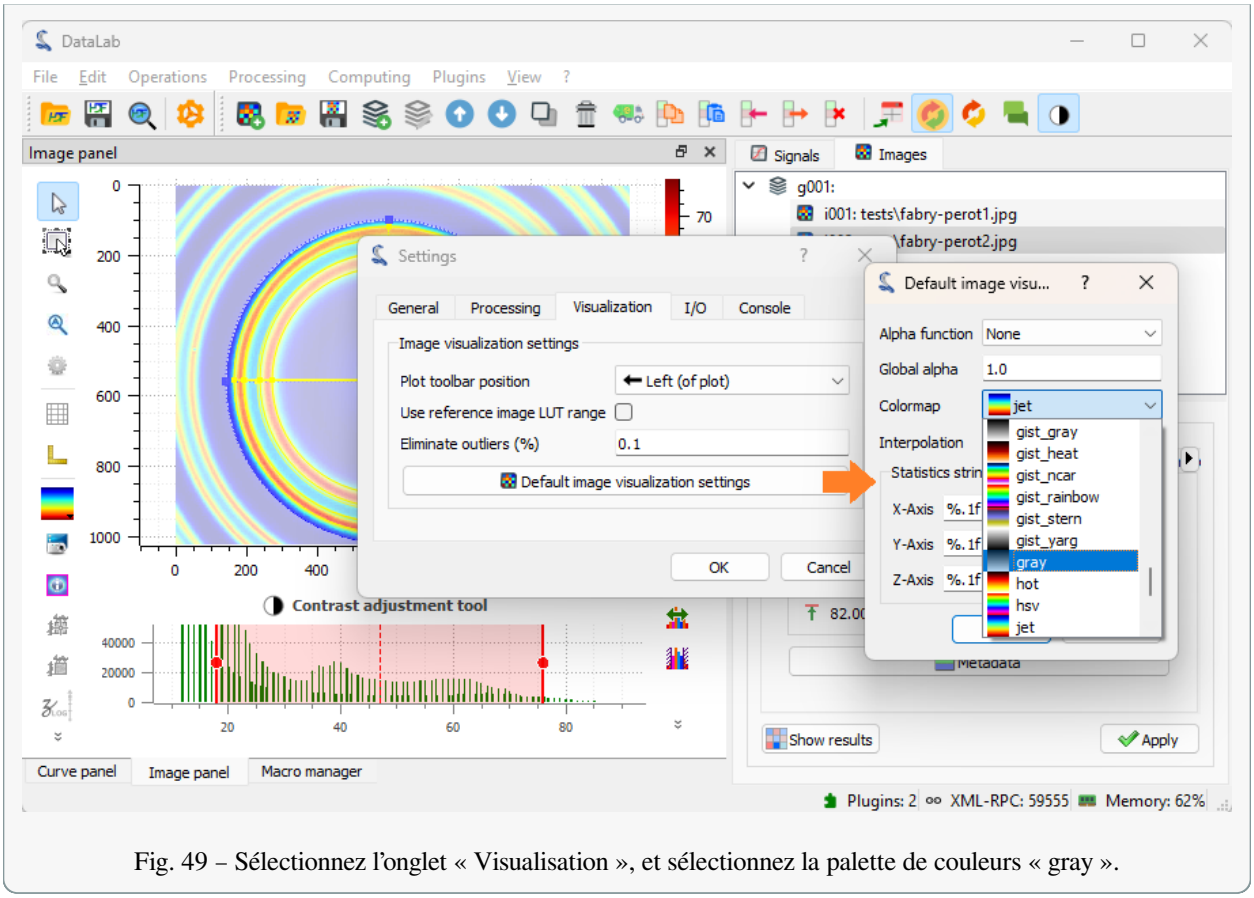


Fig. 49 – Sélectionnez l'onglet « Visualisation », et sélectionnez la palette de couleurs « gray ».

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 ».

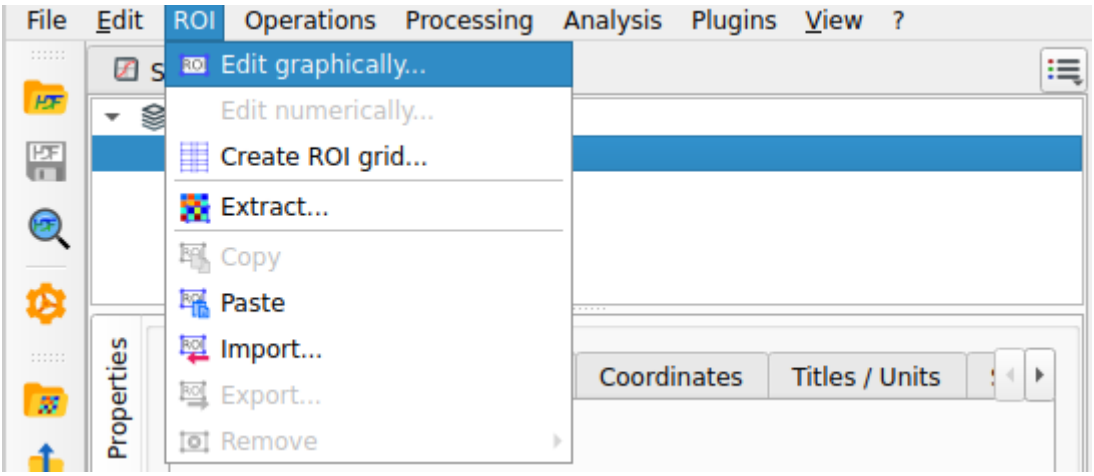


Fig. 50 – Sélectionnez l'outil « Édition graphique » dans le menu « ROI ».

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).

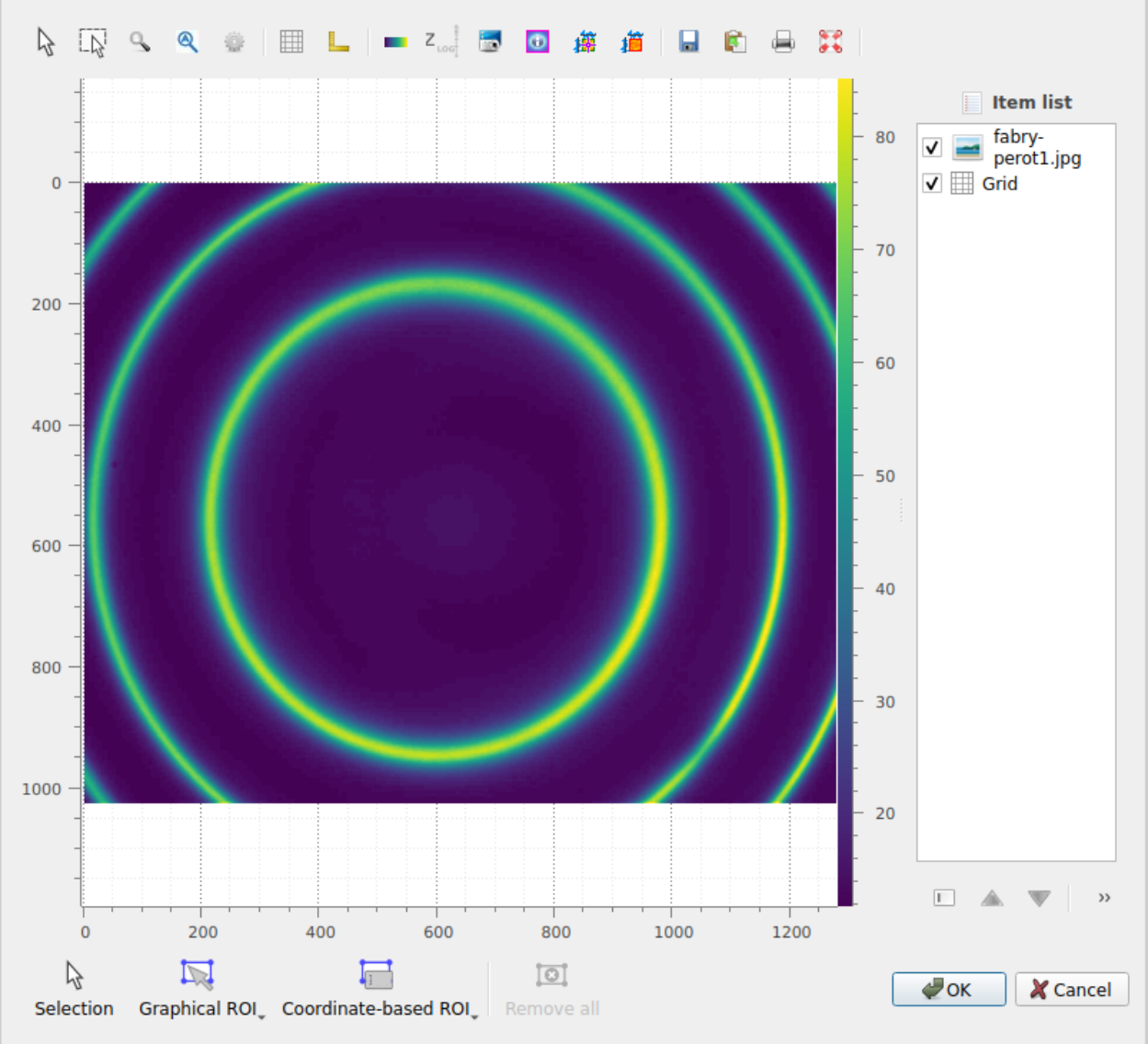


Fig. 51 – La boîte de dialogue « Régions d'intérêt » s'ouvre. Cliquez sur « Ajouter une ROI » et sélectionnez une ROI circulaire. Redimensionnez la ROI prédéfinie en faisant glisser les poignées. Notez que vous pouvez modifier le rayon de la ROI tout en gardant son centre fixe en appuyant sur la touche « Ctrl ». Cliquez sur « OK » pour fermer la boîte de dialogue.

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

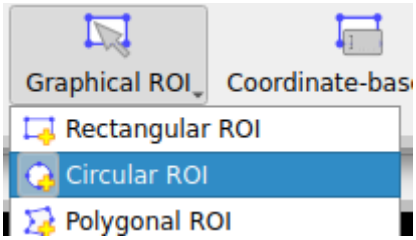


Fig. 52 – L'option pour définir graphiquement une ROI circulaire.

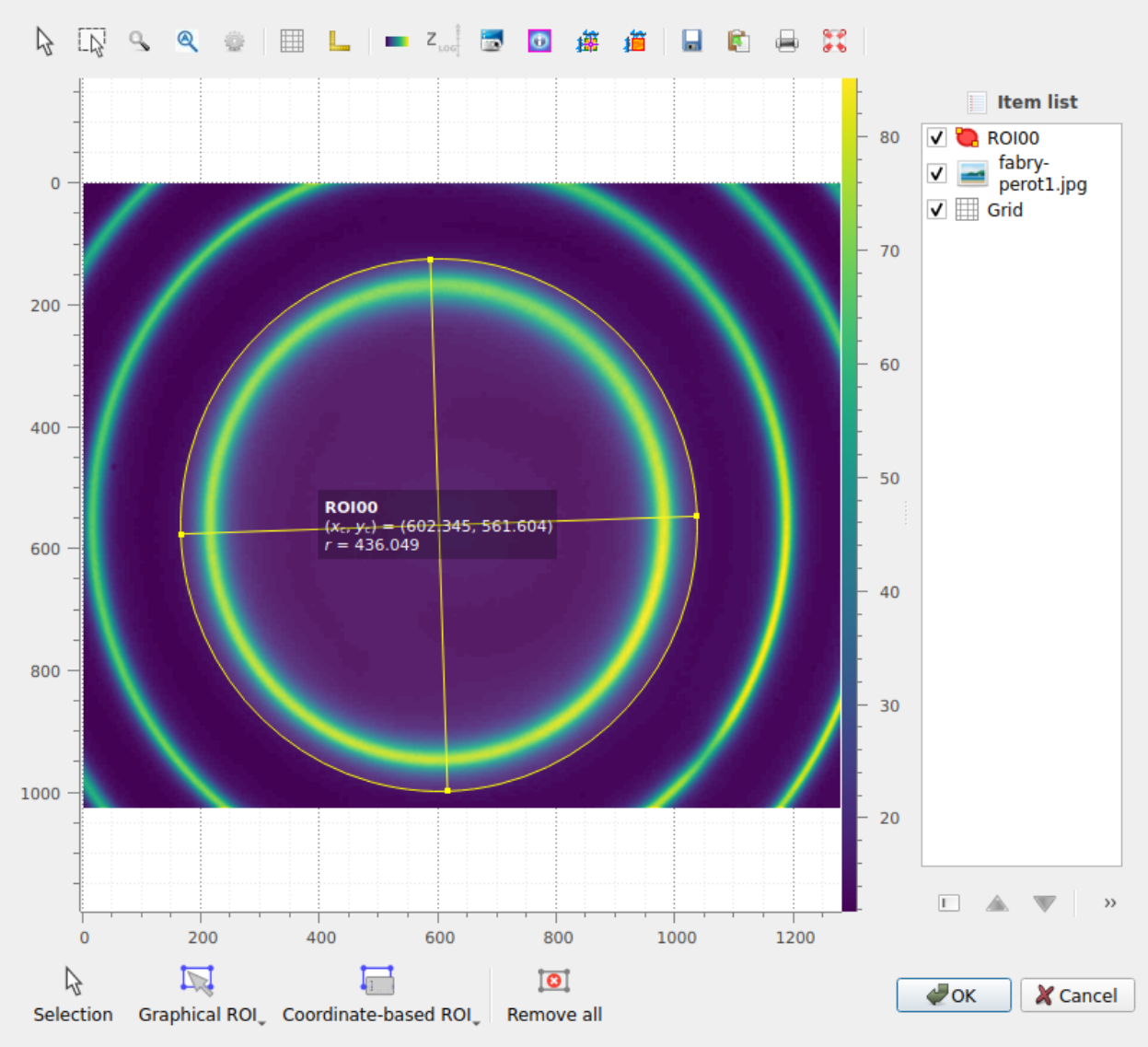


Fig. 53 – La ROI circulaire créée apparaît dans l'image.

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

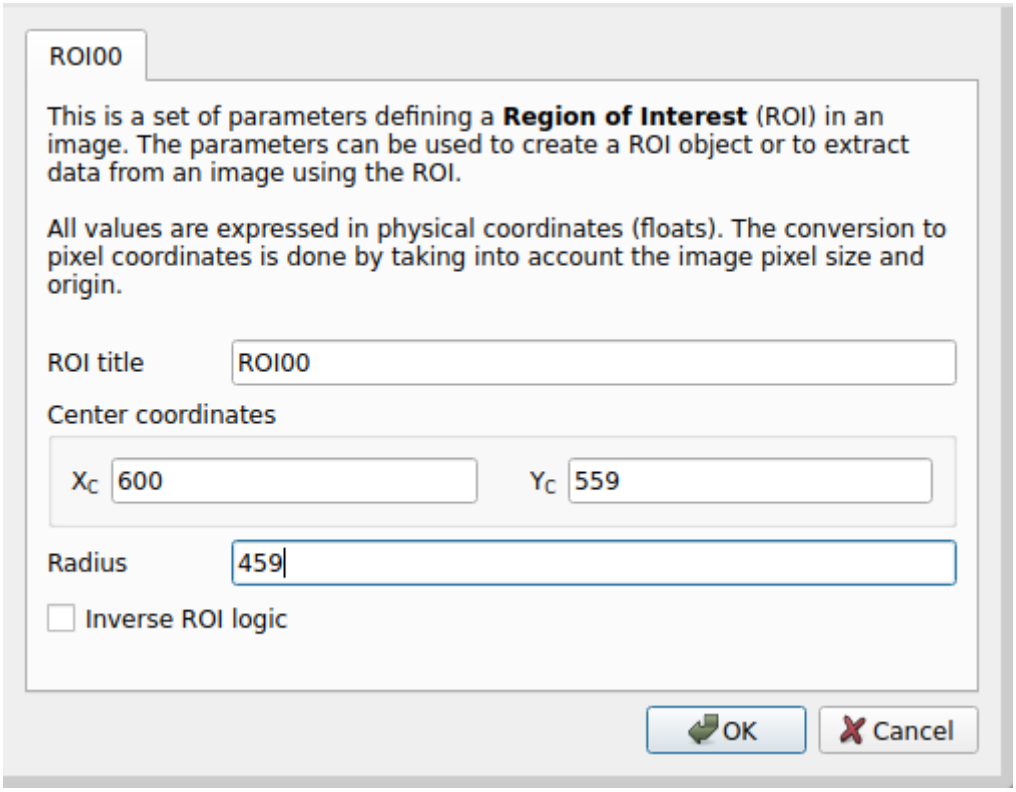


Fig. 54 – La boîte de dialogue de confirmation.

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 :

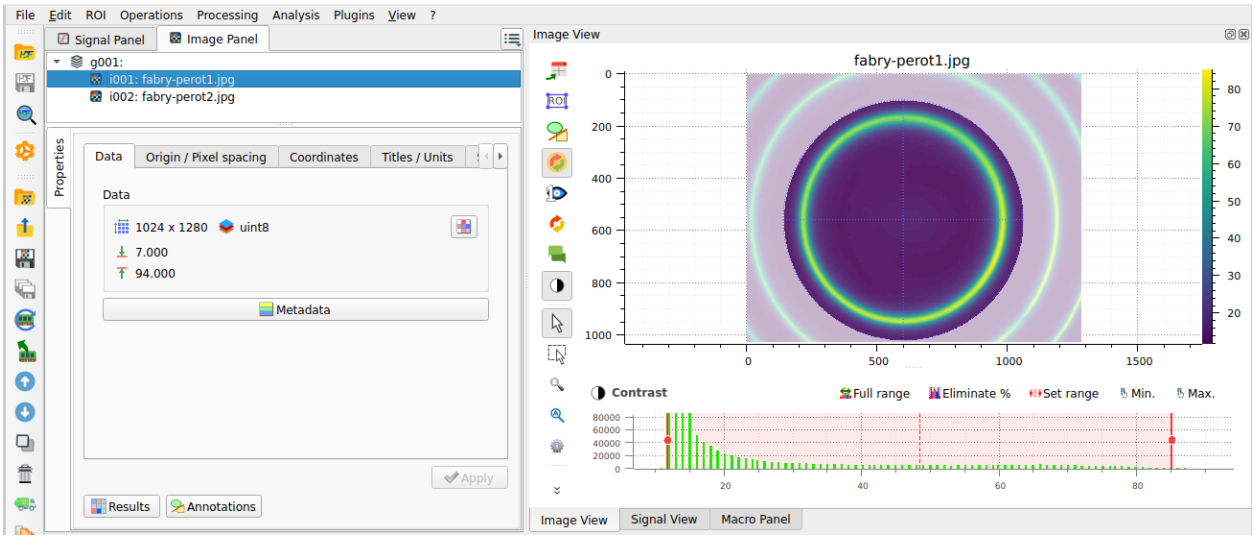


Fig. 55 – La ROI affichée sur l'image.

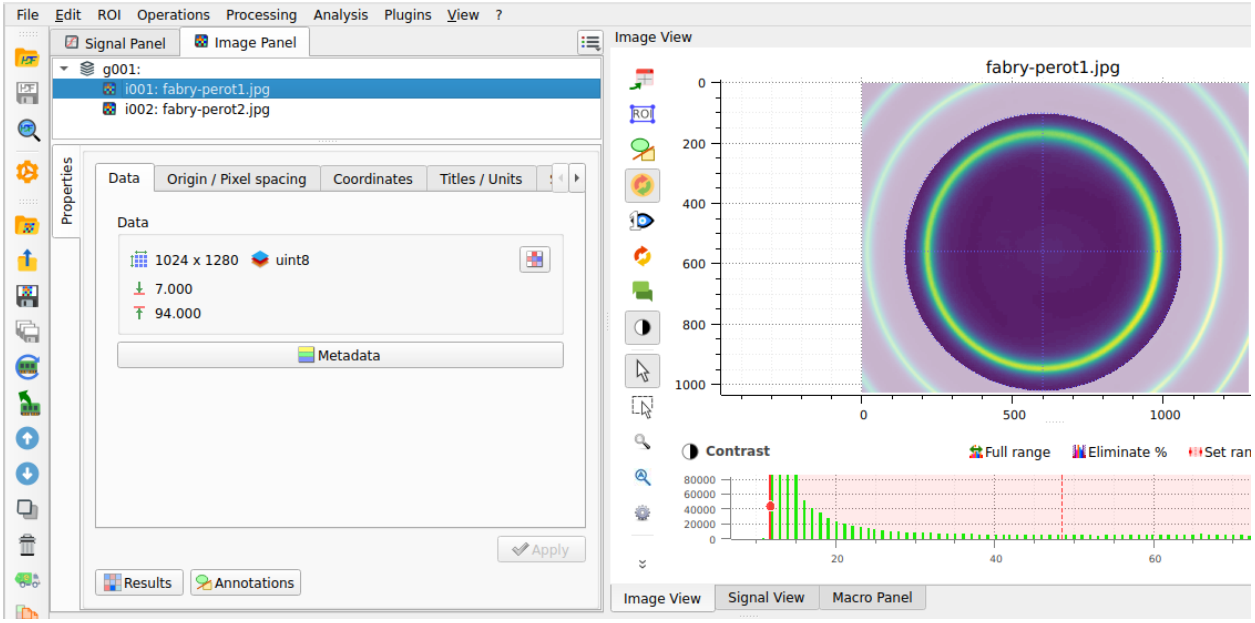


Fig. 56 – L'outil « Détection de contours » dans le menu « Analyse ».

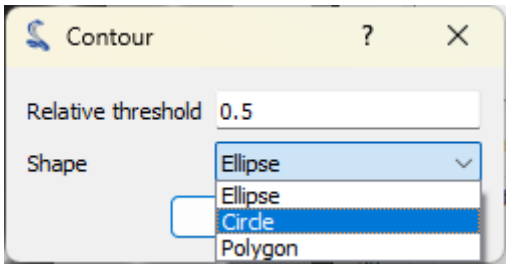


Fig. 57 – La boîte de dialogue des paramètres de contour.

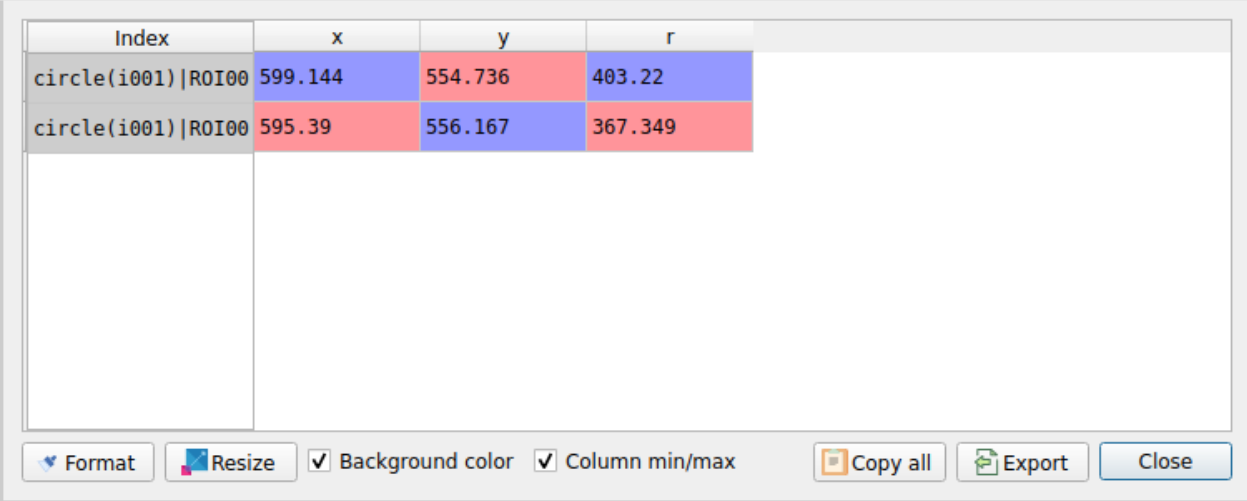


Fig. 58 – La boîte de dialogue des résultats.

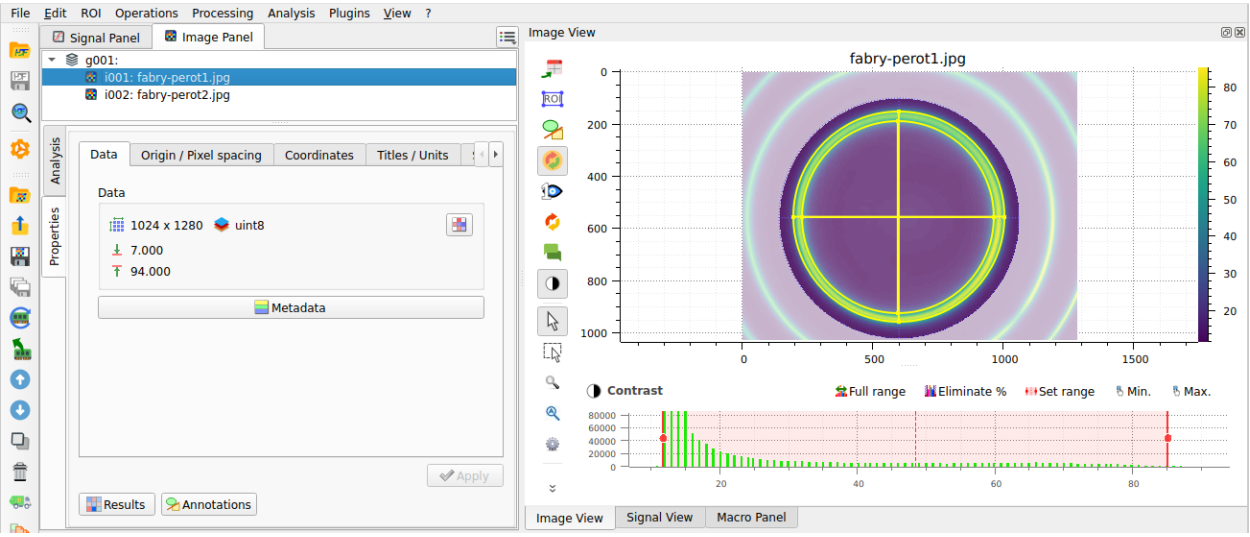
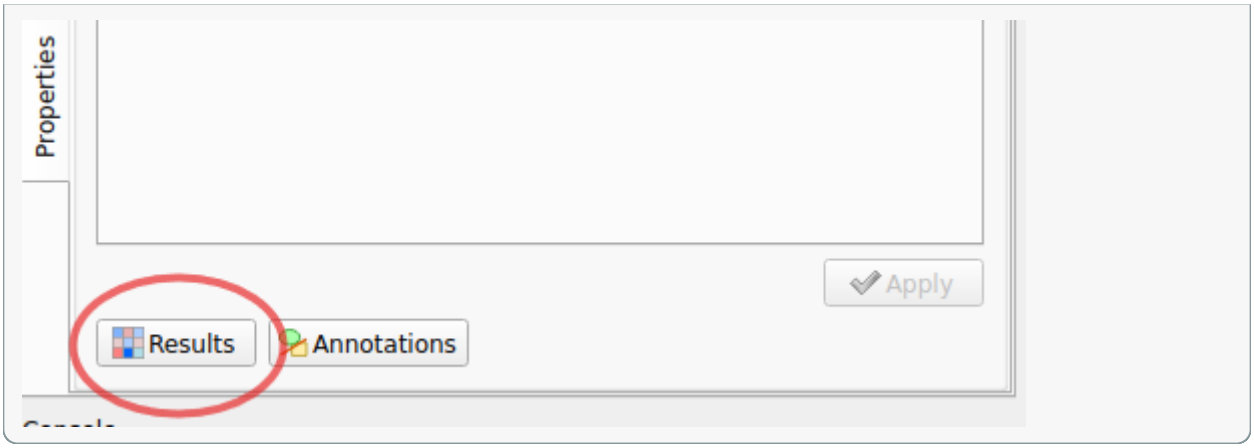


Fig. 59 – Les cercles ajustés sont affichés sur l'image.



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.

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

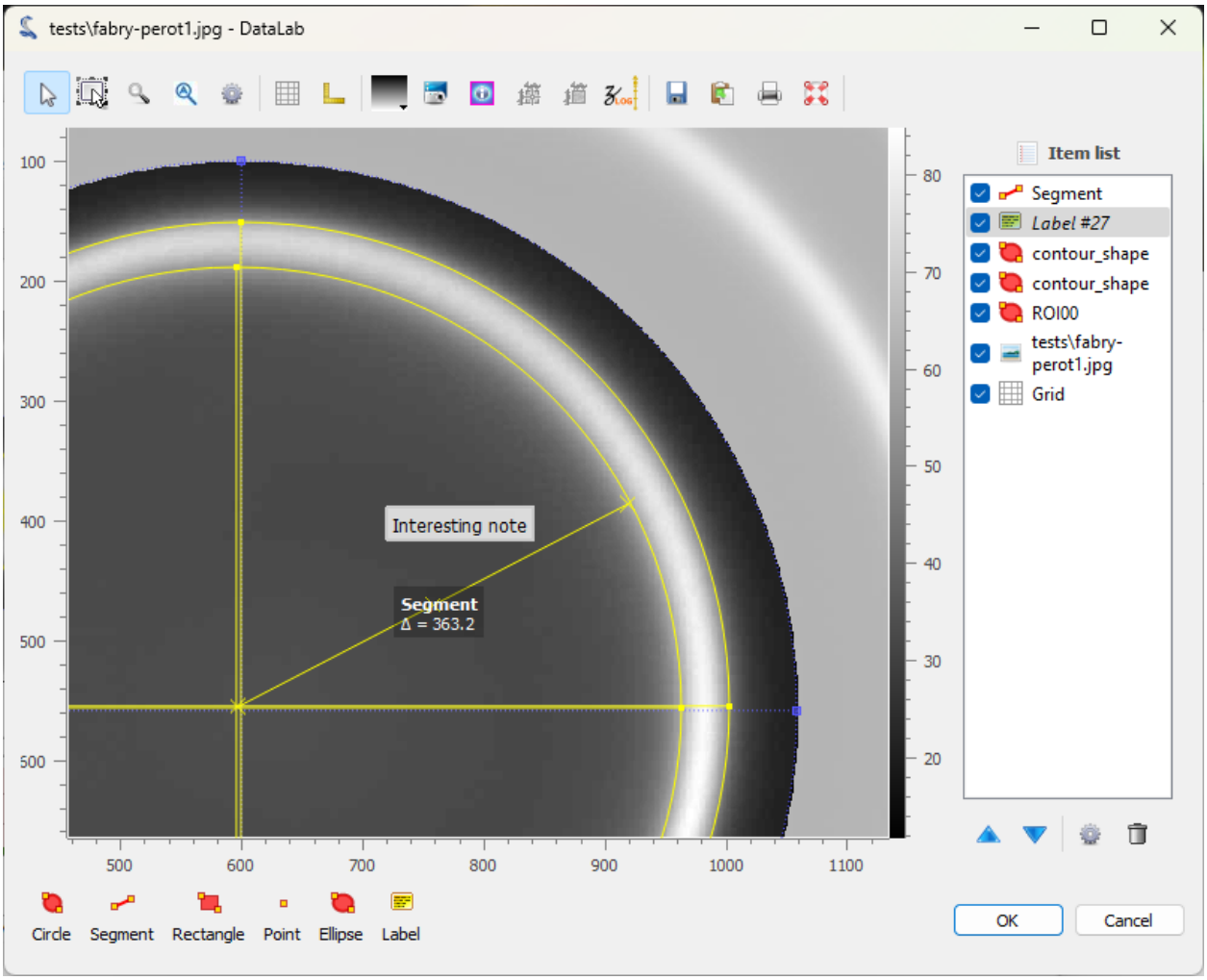


Fig. 60 – L'image affichée dans une fenêtre séparée. La ROI et les cercles ajustés sont également affichés. Des annotations peuvent être ajoutées à l'image en cliquant sur les boutons en bas de la fenêtre. Les annotations sont stockées dans les métadonnées de l'image, et avec les données de l'image lorsque l'espace de travail est sauvegardé. Cliquez sur « OK » pour fermer la fenêtre.

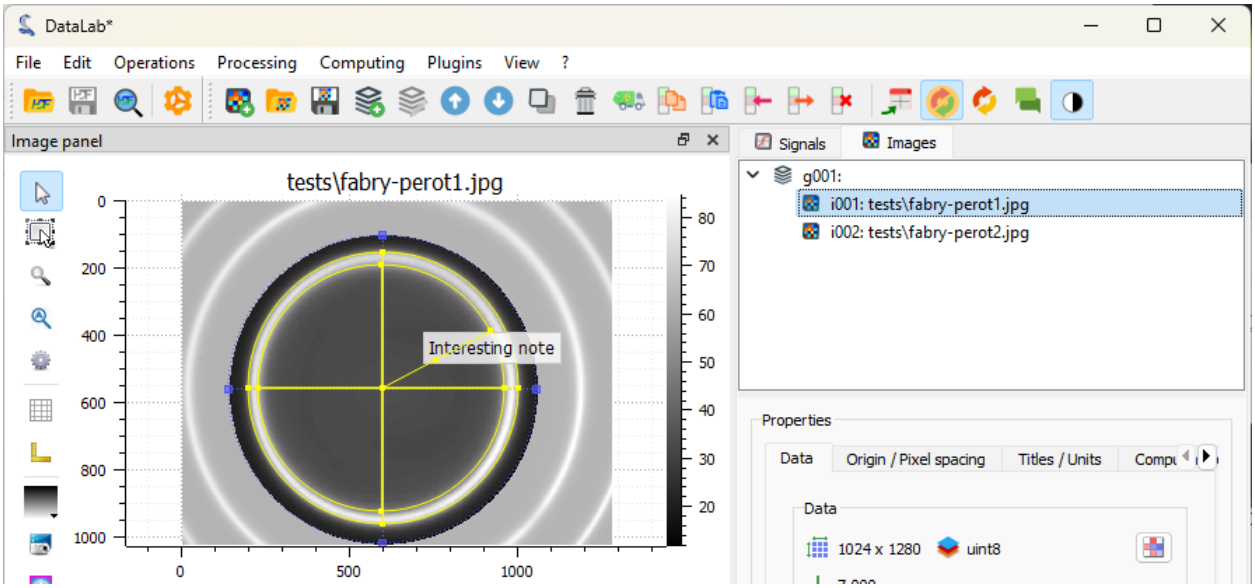


Fig. 61 – L'image est affichée dans la fenêtre principale, avec les annotations.

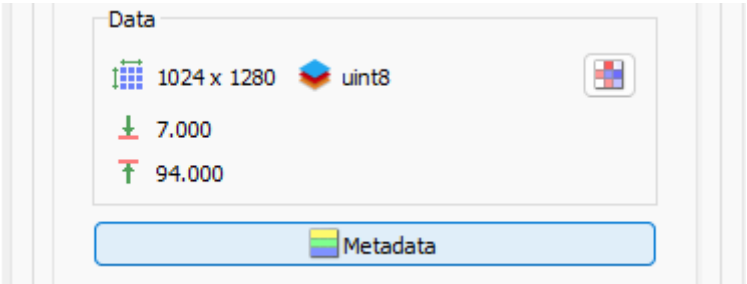


Fig. 62 – Le bouton « Métadonnées » est situé en dessous de la liste des images.

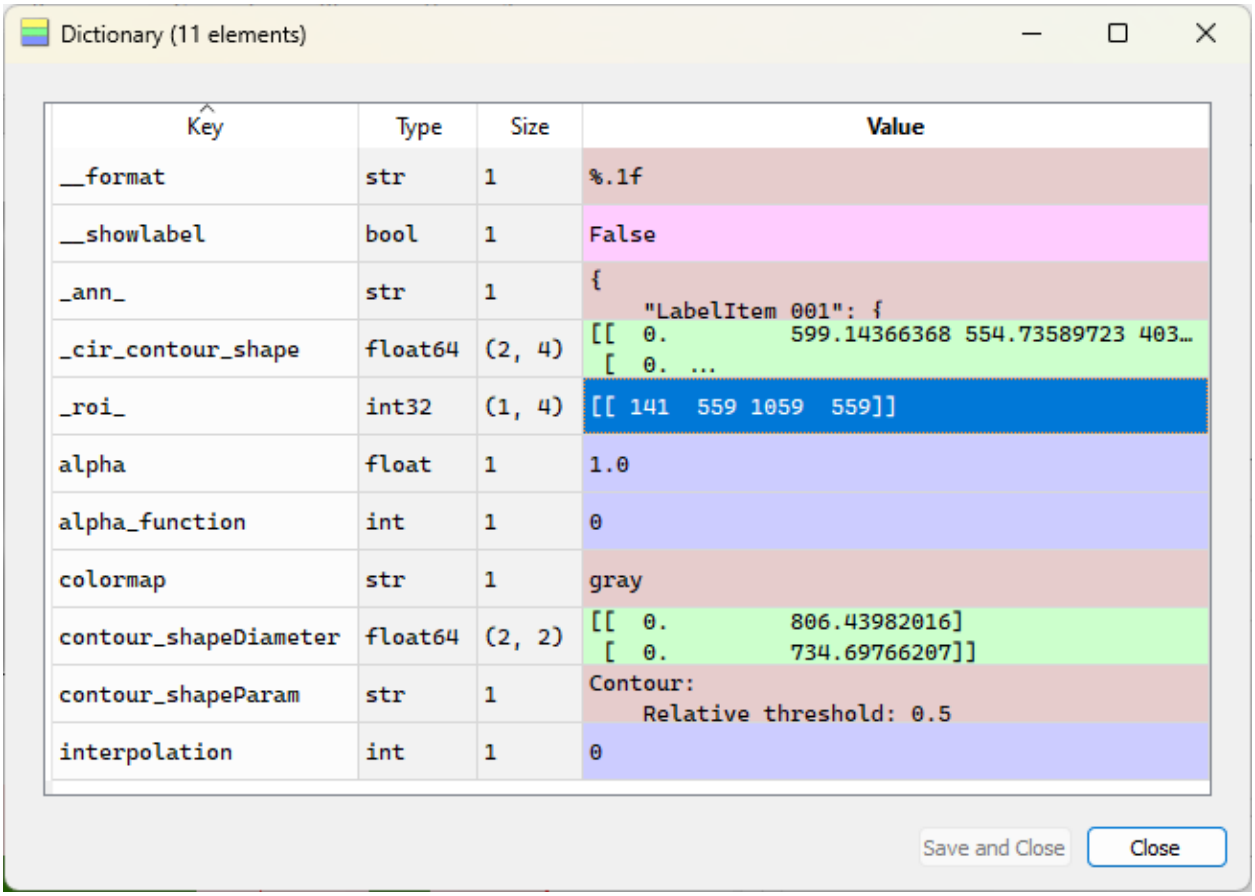


Fig. 63 – La boîte de dialogue « Métadonnées » s'ouvre. Parmi d'autres informations, elle affiche les annotations (dans un format JSON), des informations de style (par exemple la palette de couleurs), et la ROI.

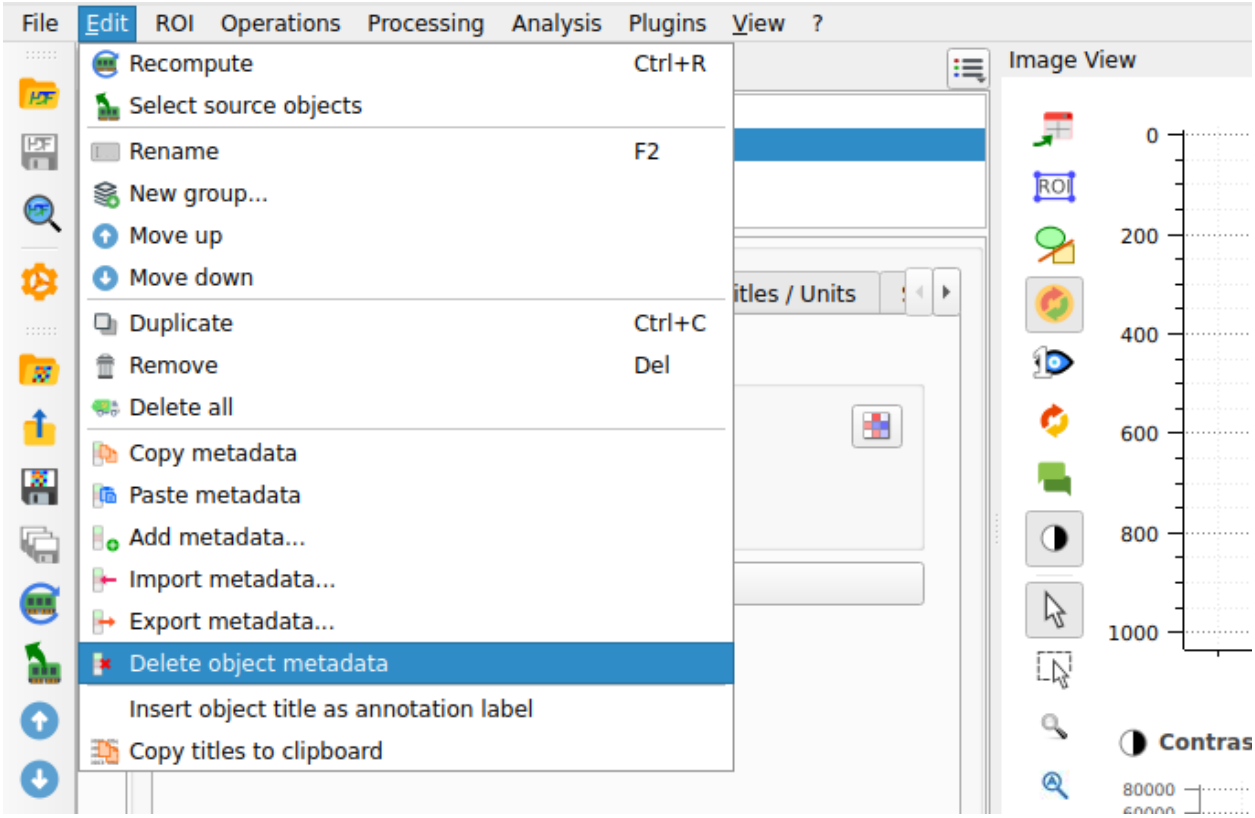


Fig. 64 – Sélectionnez l'entrée « Supprimer les métadonnées de l'objet » dans le menu « Édition ».

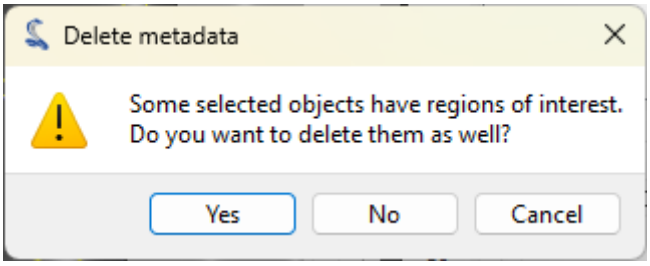


Fig. 65 – La boîte de dialogue « Supprimer les métadonnées ». Cliquez sur « Non » pour conserver la ROI et supprimer le reste des métadonnées.

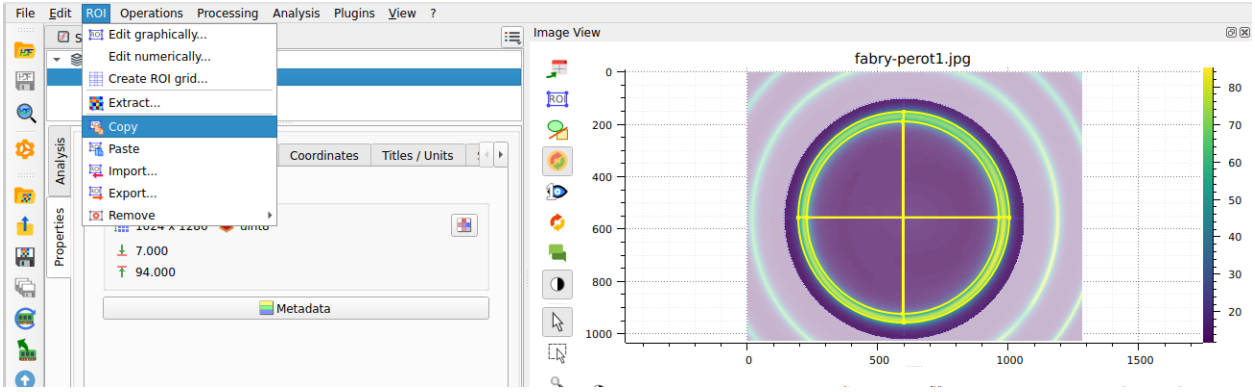


Fig. 66 – L'entrée « Copier » dans le menu « ROI ».

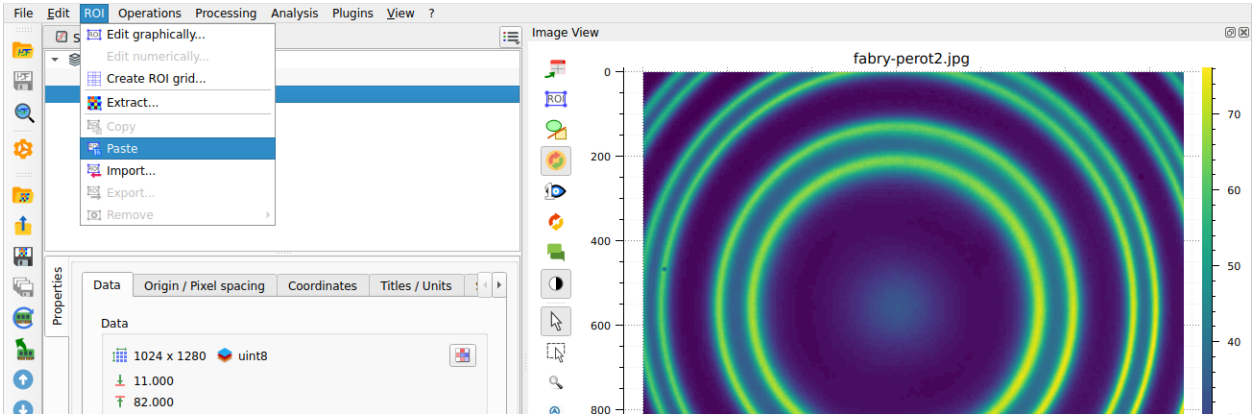


Fig. 67 – L'entrée « Coller » dans le menu « ROI », ou le bouton dans la barre d'outils.

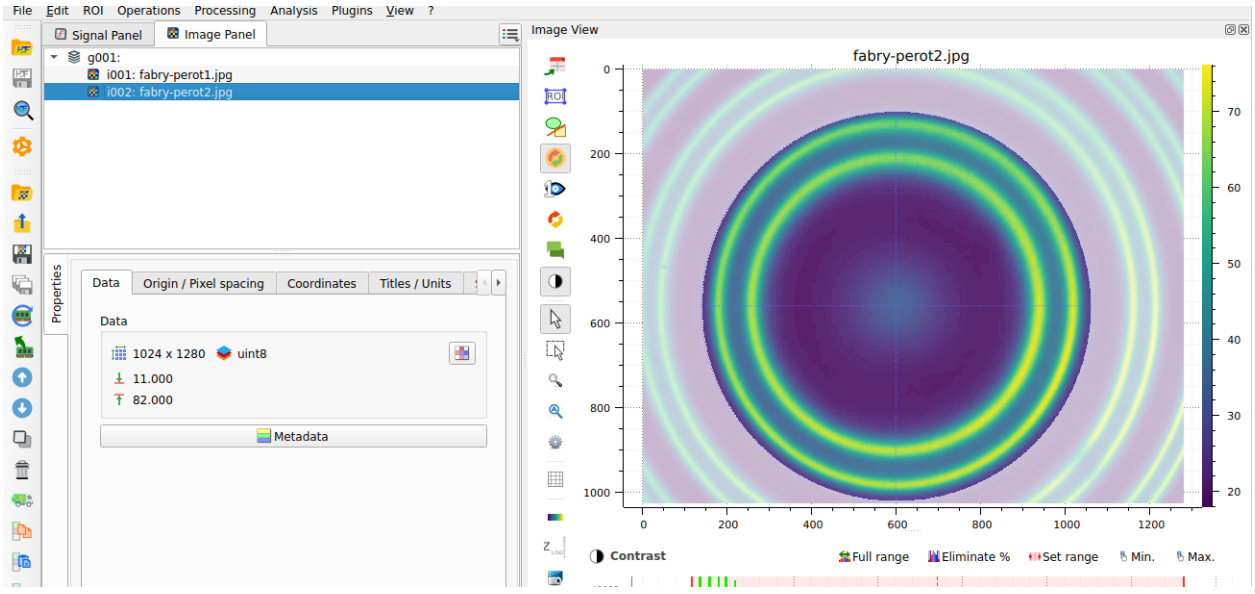


Fig. 68 – La ROI est ajoutée à la deuxième image.

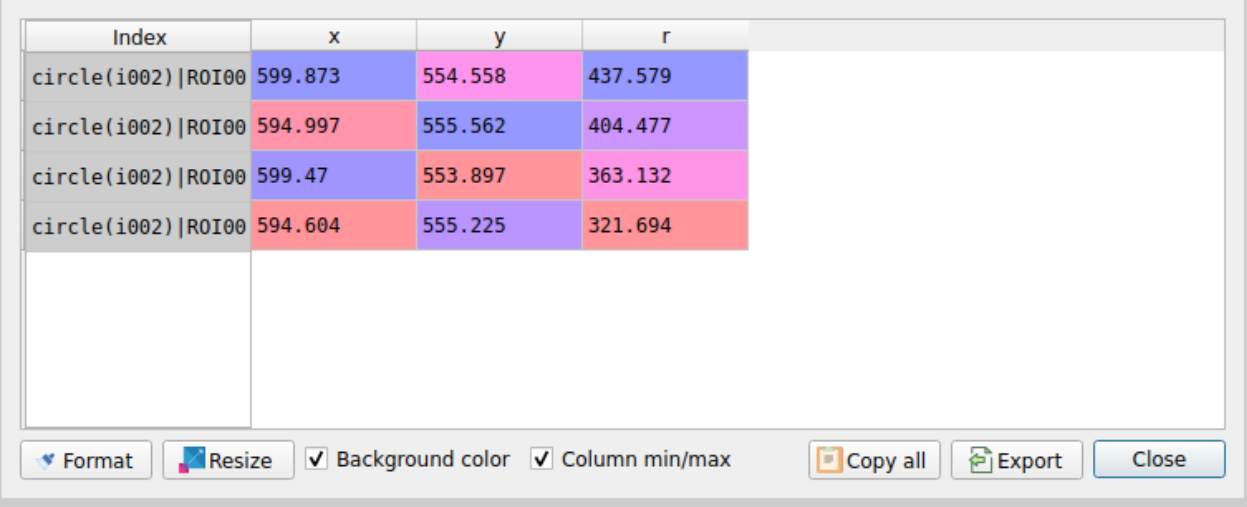


Fig. 69 – La boîte de dialogue « Résultats » pour la deuxième image.

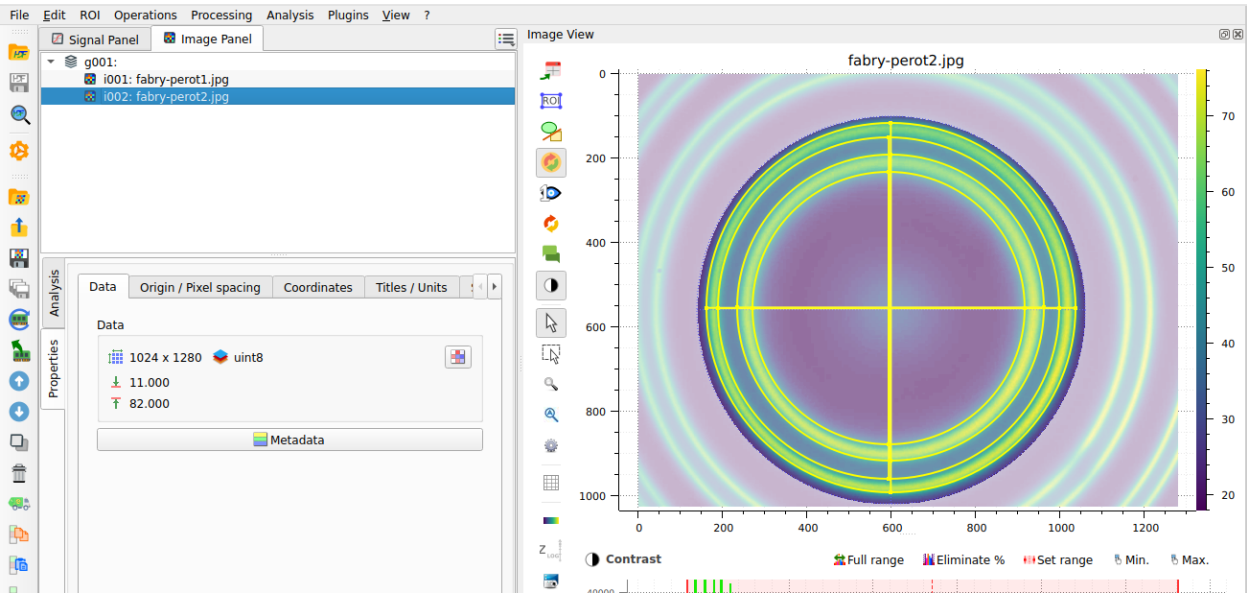


Fig. 70 – Les cercles ajustés sont affichés sur l'image.

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.

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 la coordonnée Y de la ligne (ou la coordonnée X de la colonne) en cliquant sur « paramètres… » dans la boîte de dialogue « Profil rectiligne ».

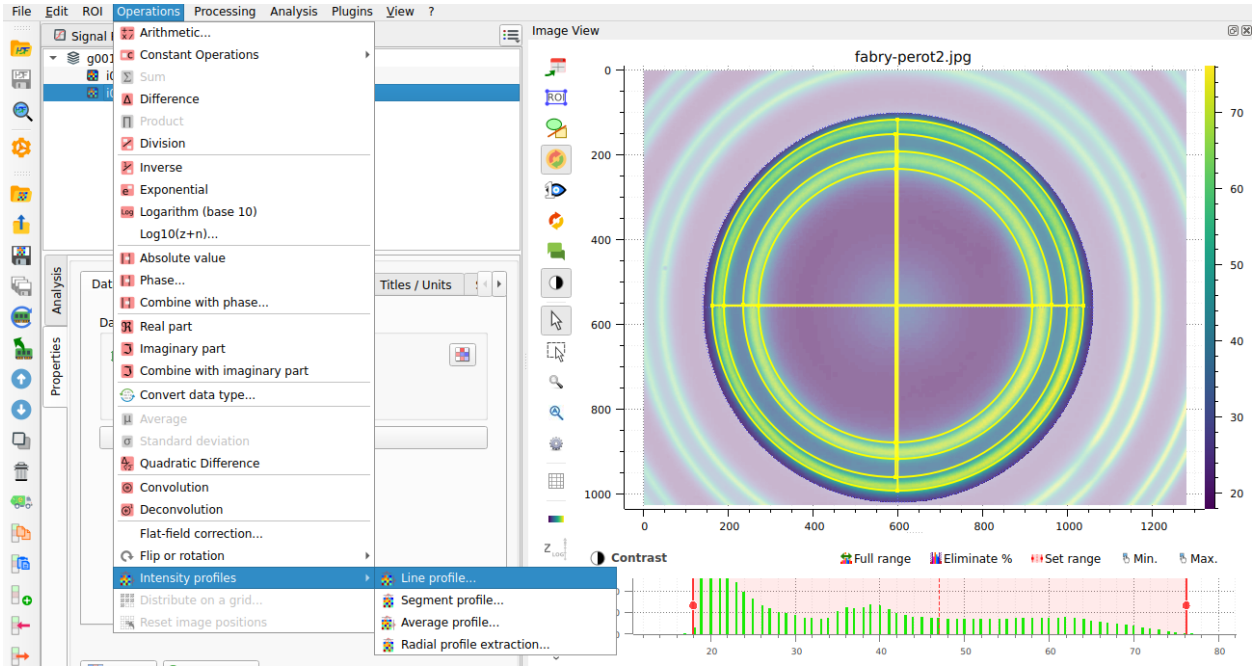


Fig. 71 – Sélectionnez l'entrée « Profil rectiligne… » dans le menu « Opérations ».

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

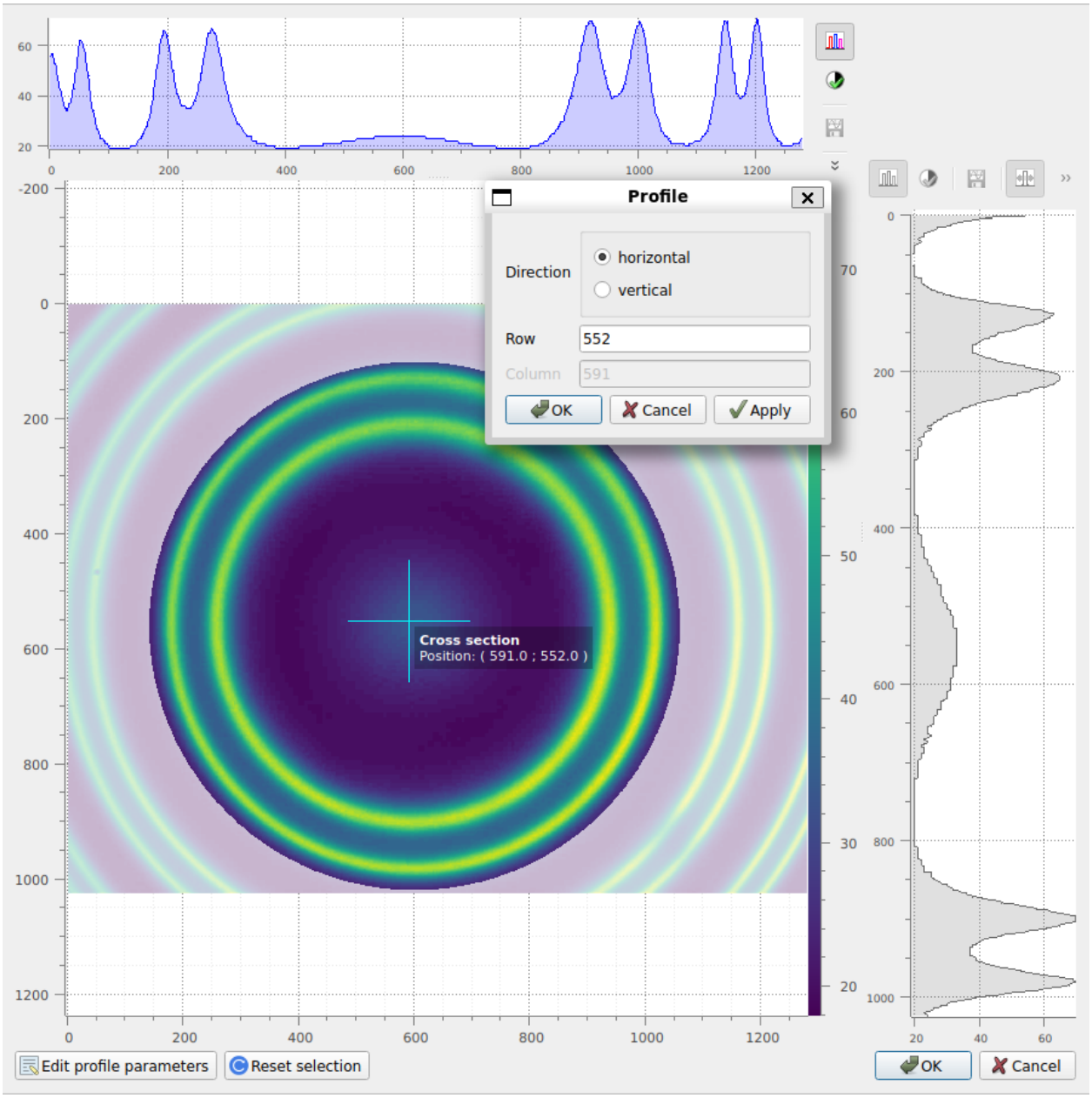


Fig. 72 – La boîte de dialogue « Profil » s'ouvre. Entrez la coordonnée Y du profil horizontal (ou la coordonnée X du profil vertical) dans la boîte de dialogue qui s'ouvre. Cliquez sur « OK ».

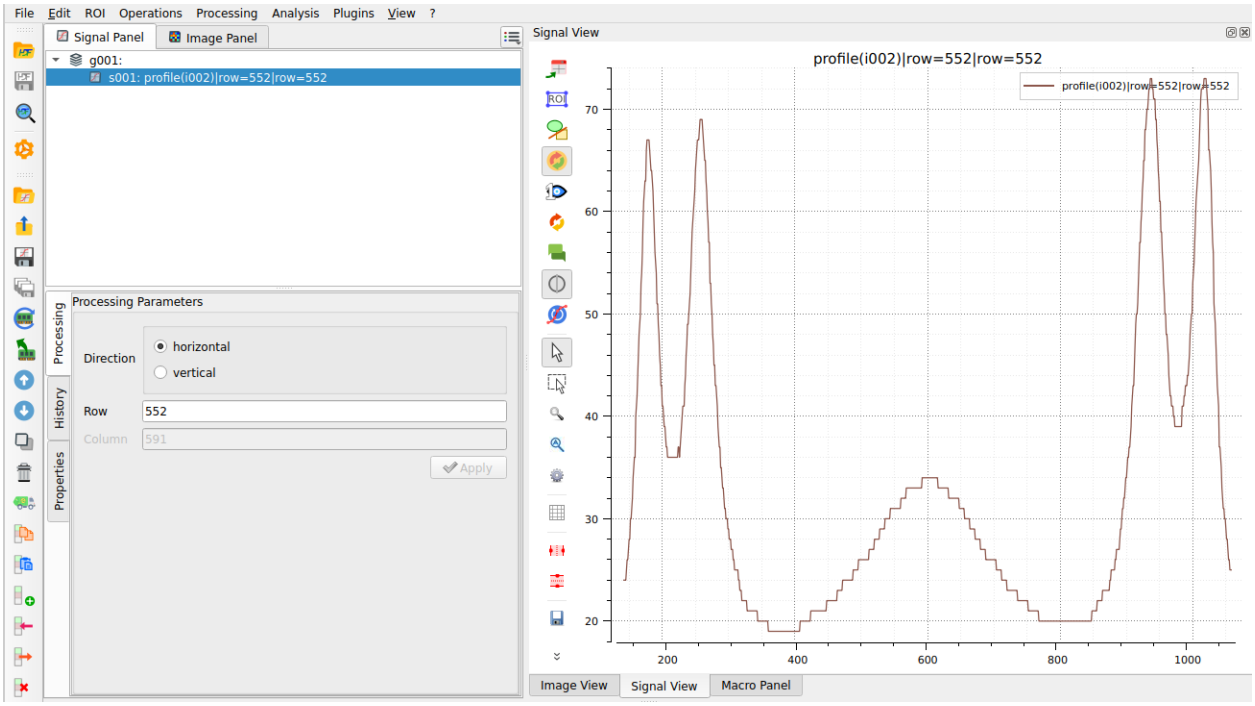


Fig. 73 – Le profil d'intensité est ajouté au panneau « Signaux », et DataLab bascule vers ce panneau pour afficher le profil.

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.

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.

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

Profilométrie de faisceau laser : mesurer la taille du faisceau le long de l'axe de propagation

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

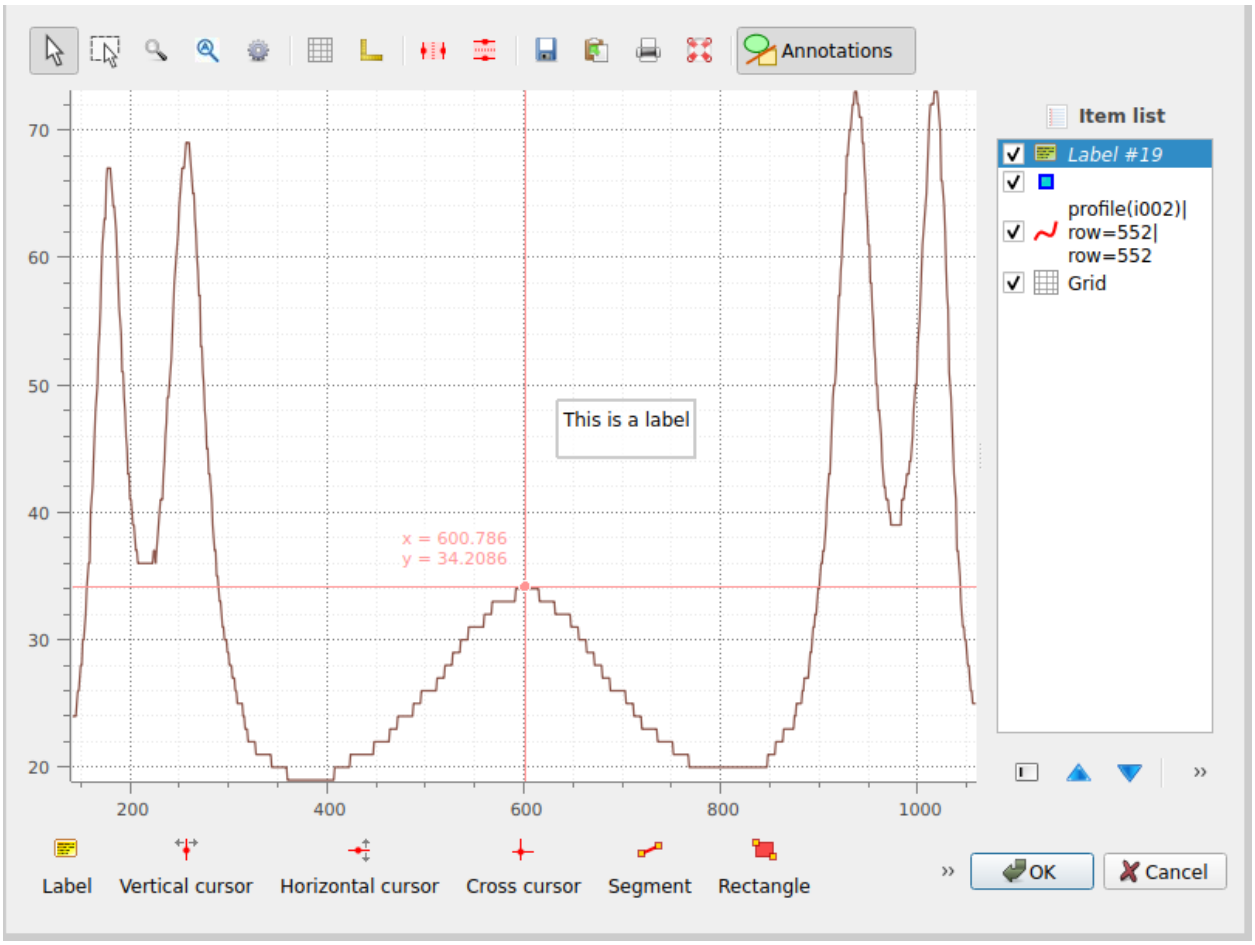


Fig. 74 – Le signal est affiché dans une fenêtre séparée. Ici, nous avons ajouté des curseurs verticaux et une étiquette de texte très intéressante. Comme pour les images, les annotations sont stockées dans les métadonnées du signal, et avec les données du signal lorsque l'espace de travail est sauvegardé. Cliquez sur « OK » pour fermer la fenêtre.

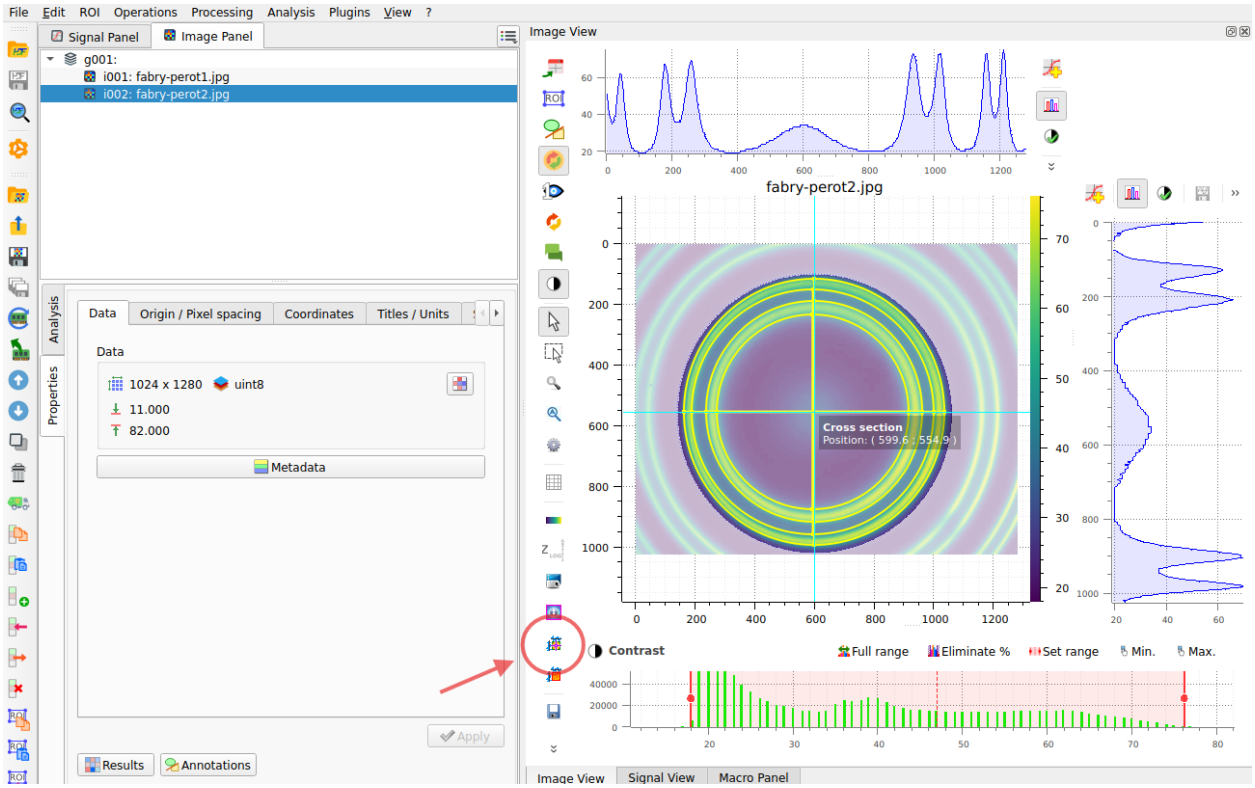


Fig. 75 – L'outil de profil rectiligne dans le panneau de visualisation. Dans le cercle rouge, le bouton pour l'activer. Dans le cercle bleu, le bouton pour transférer le profil vers le panneau « Signaux ».

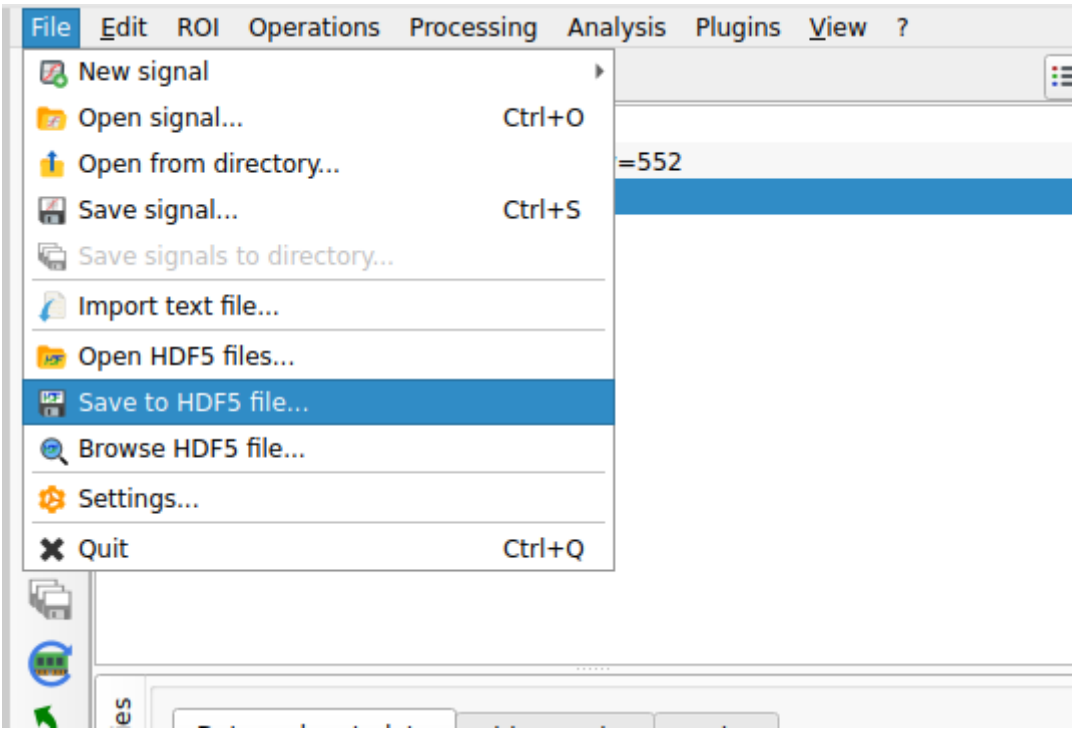


Fig. 76 – Sauvegardez l'espace de travail dans un fichier avec « Fichier > Sauvegarder dans un fichier HDF5… », ou le bouton dans la barre d'outils.

— 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

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.

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.

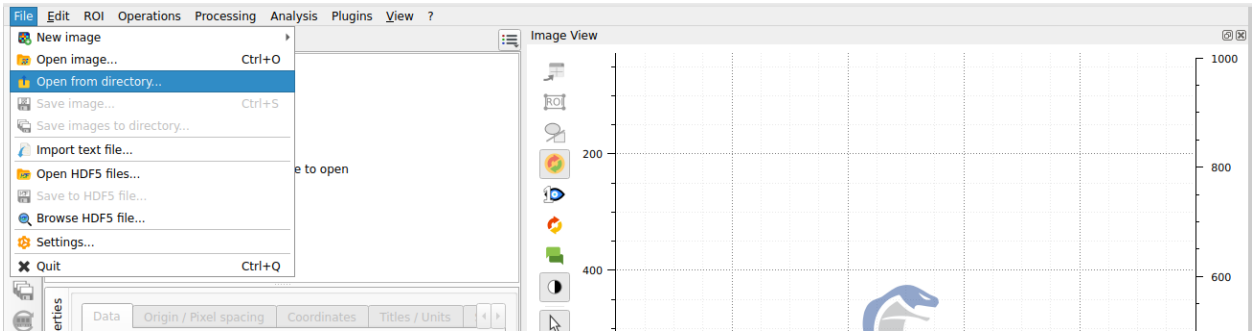


Fig. 77 – Le menu « Fichier > Ouvrir une image… »

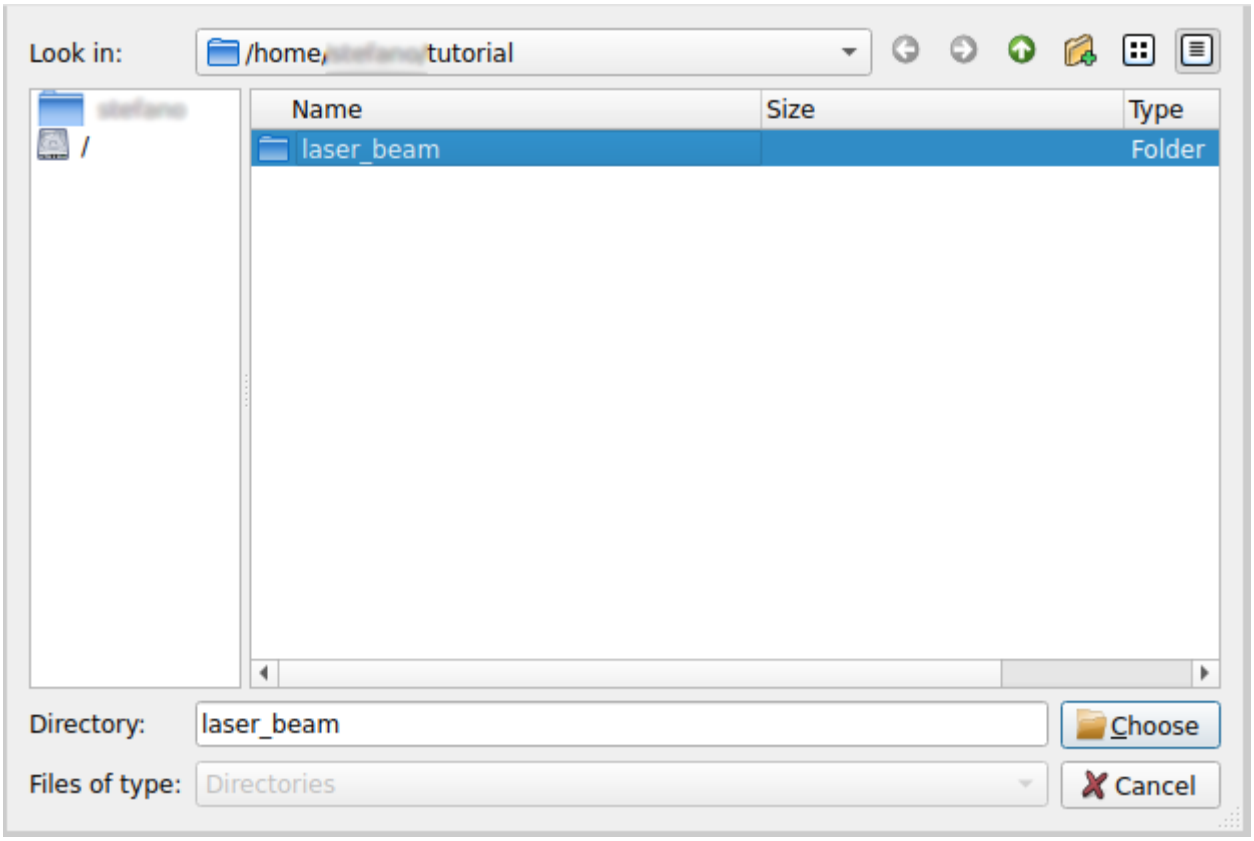


Fig. 78 – Sélectionnez le dossier contenant les images et cliquez sur « Choisir ».

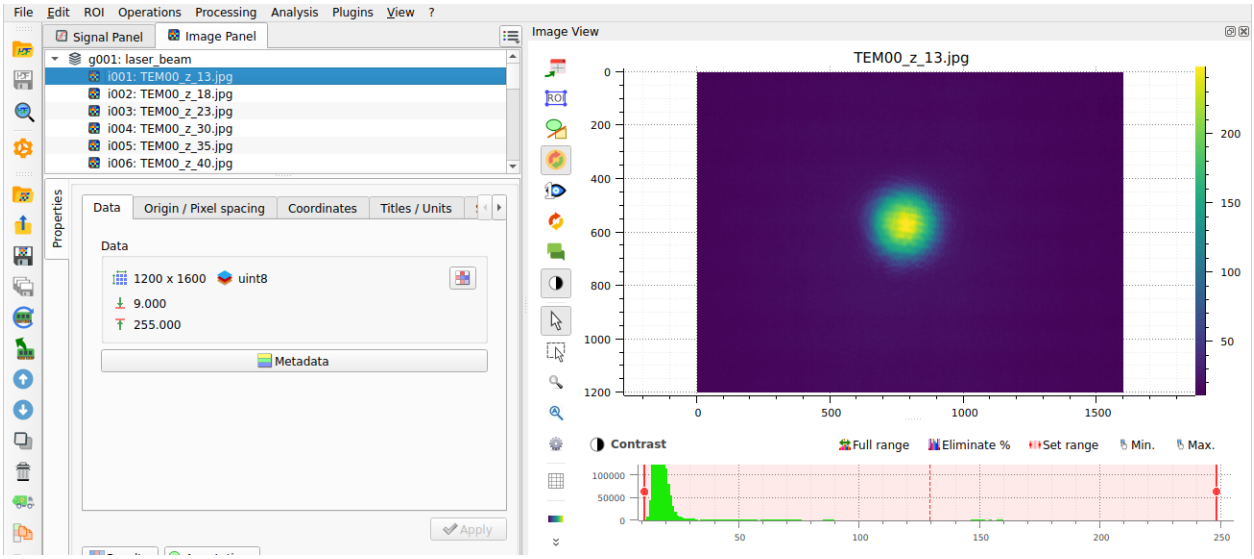


Fig. 79 – Zoomer avec le bouton droit de la souris. Déplacer l'image avec le bouton du milieu de la souris.

Note

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

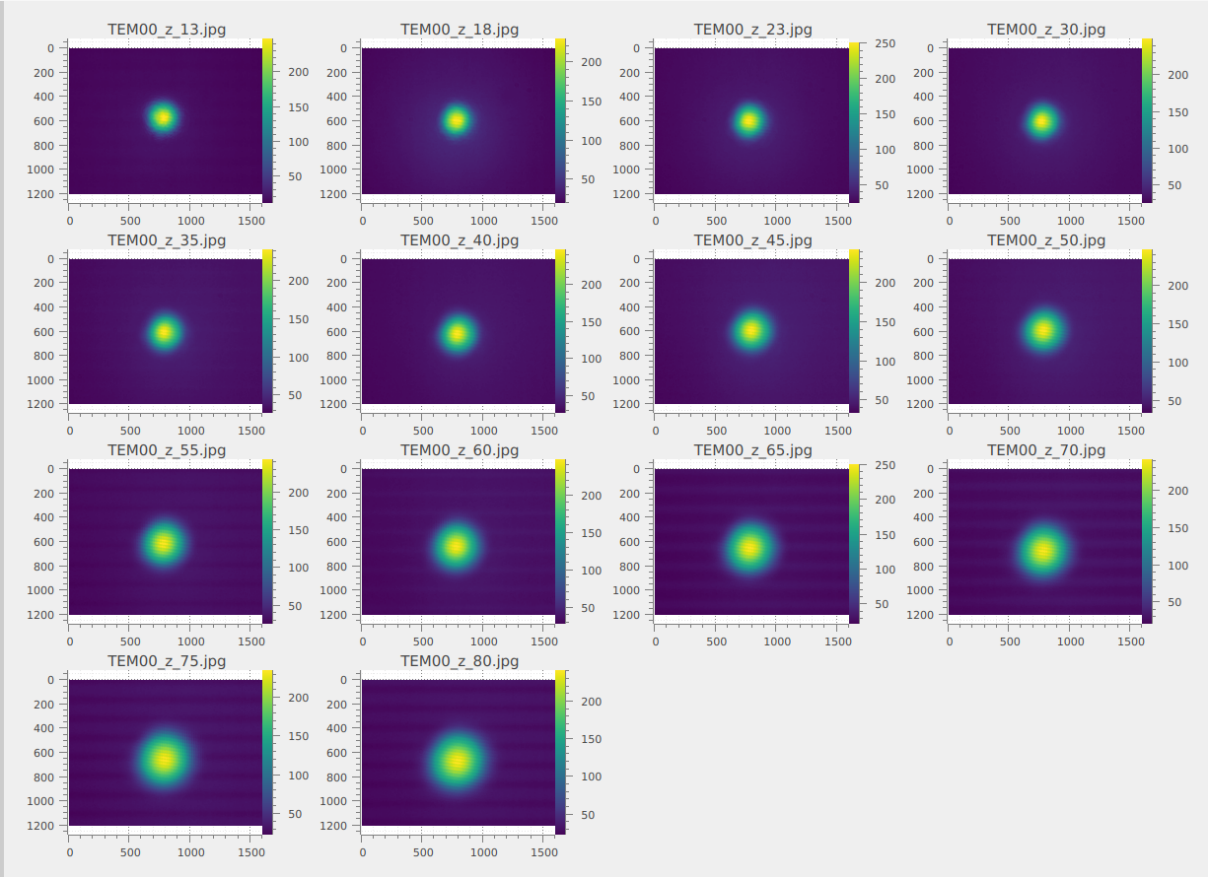
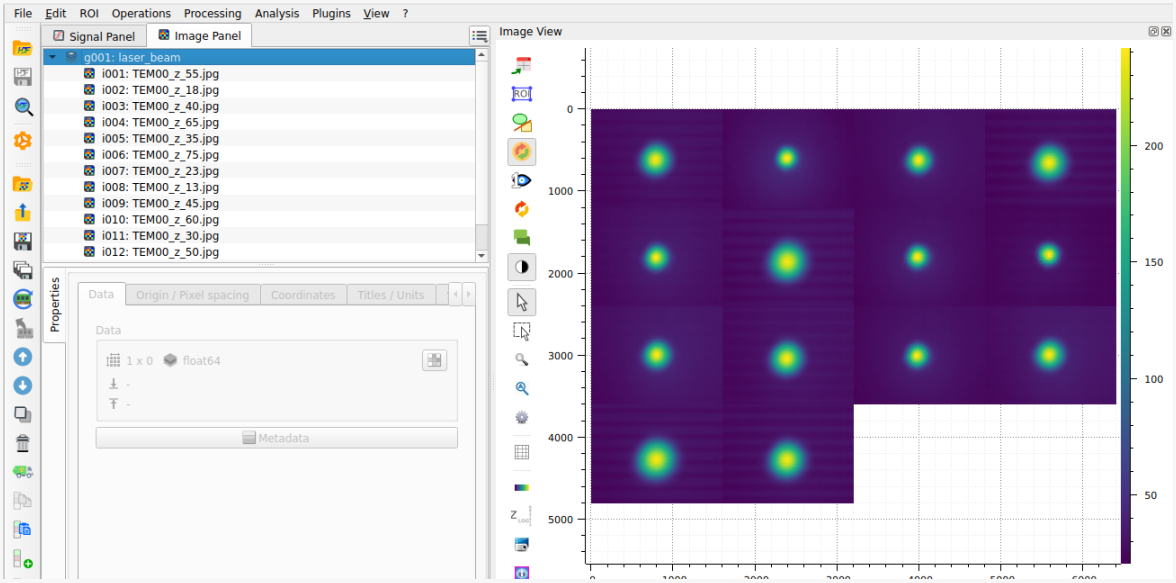


Fig. 80 – Affichage des images côte à côte.

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.
Les options « Distribuer sur une grille » et « Réinitialiser les positions des images » modifient les images sur place, sans créer de nouvelles images. Pour cette raison, ces opérations ne sont pas enregistrées dans le panneau d'historique.

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 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, 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.

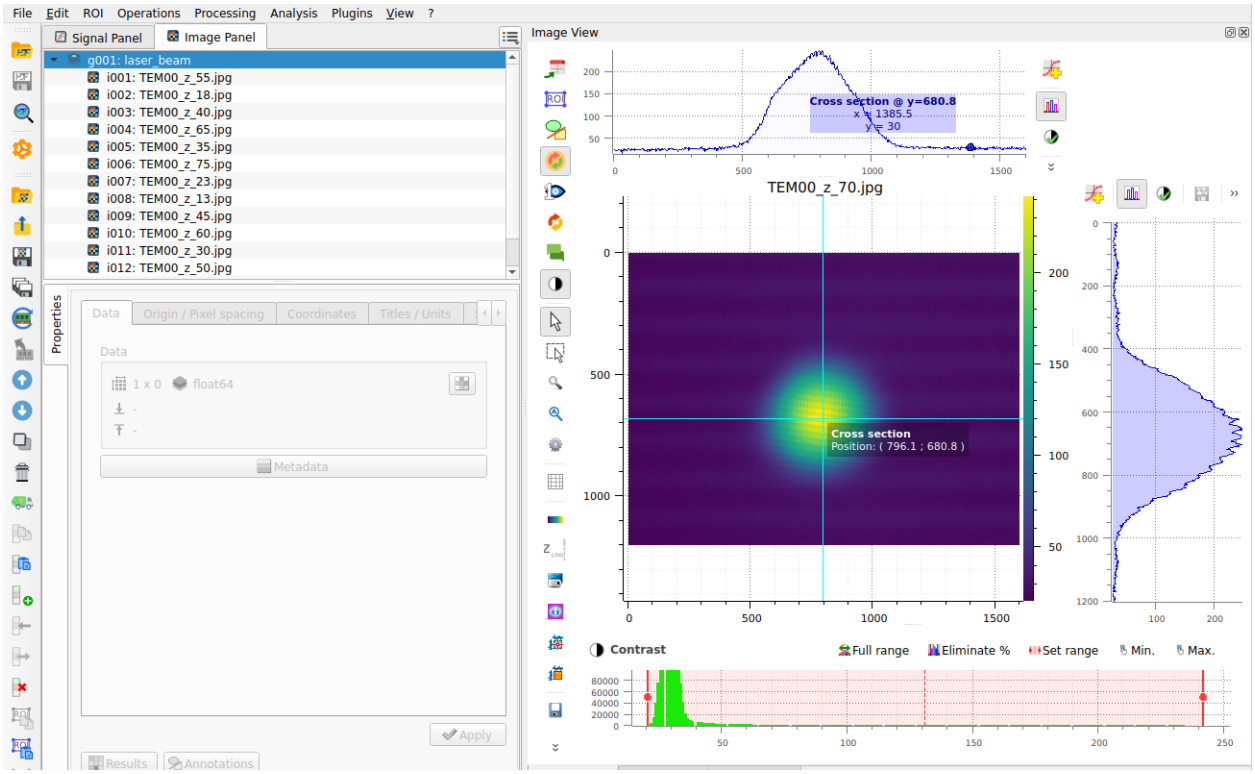


Fig. 81 – Une image du panneau « Images ». Pour afficher le marqueur de courbe, sélectionnez la courbe de profil et cliquez avec le bouton droit pour ouvrir le menu contextuel, puis choisissez « Marqueurs > Lié à l'élément actif ».

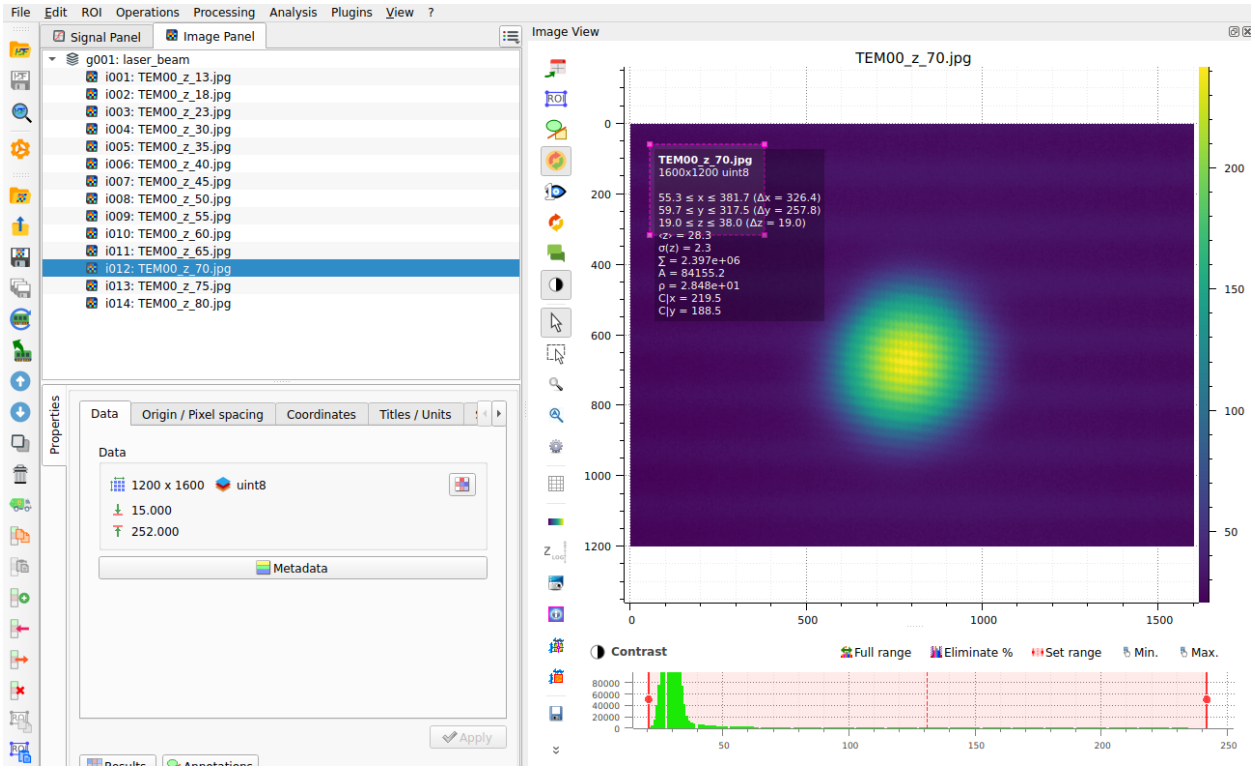


Fig. 82 – L'outil « Statistiques de l'image » dans la barre d'outils verticale.

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

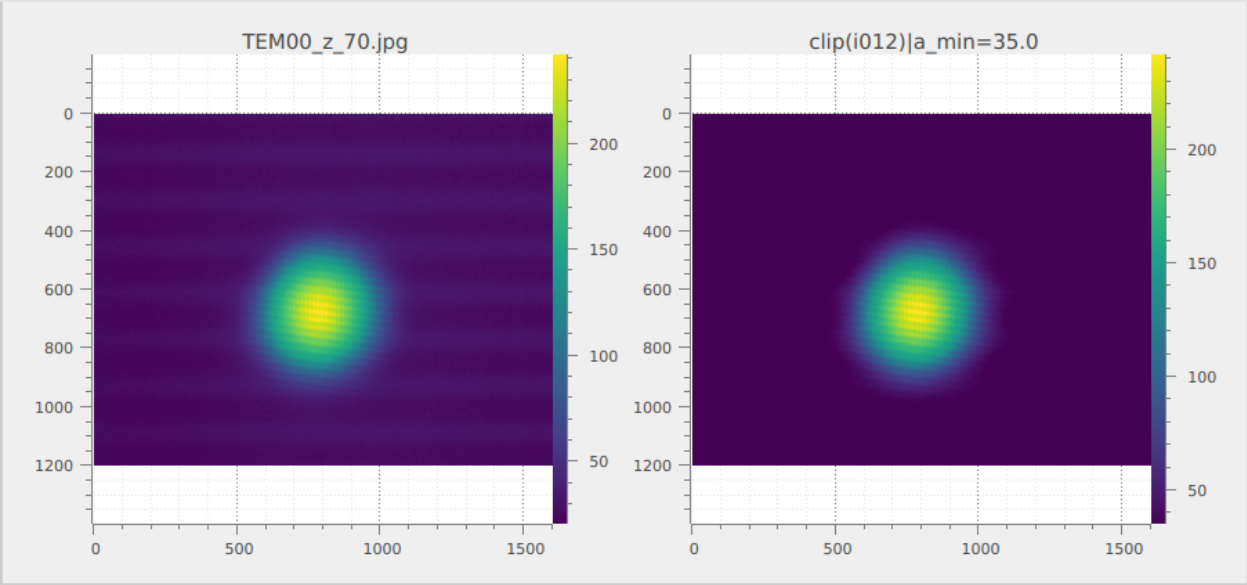


Fig. 83 – Les deux images d'origine et l'image écrêtée sont affichées côte à côte dans la « Vue Image » (en utilisant la fonction « Distribuer sur une grille » vue précédemment).

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 « Analyse > Profils d'intensité > Profil rectiligne ». Définissez la coordonnée Y de la ligne sur la coordonnée Y du centroïde calculé précédemment (c'est-à-dire 668) en utilisant le bouton « Définir les paramètres ». Voir Analyse d'interférogrammes Fabry-Perot : détection d'anneaux et ajustement de cercles 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.

> **Note**
>
> Si l'enregistrement de l'historique est activé, les actions de traitement d'image, le profil extrait et l'ajustement gaussien sont enregistrés dans la même session d'historique active. Consultez Enregistrer et rejouer une chaîne de traitement pour un exemple de flux entre les panneaux.

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'inten-

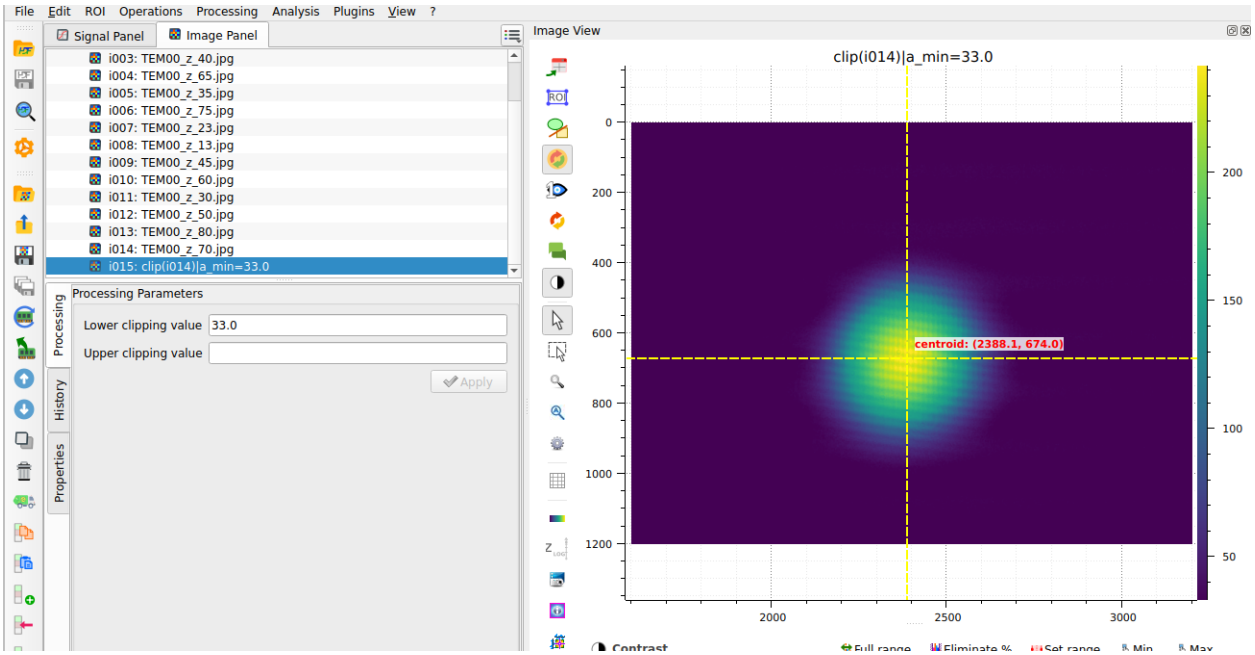


Fig. 84 – La position du centroïde est affichée sur l'image.

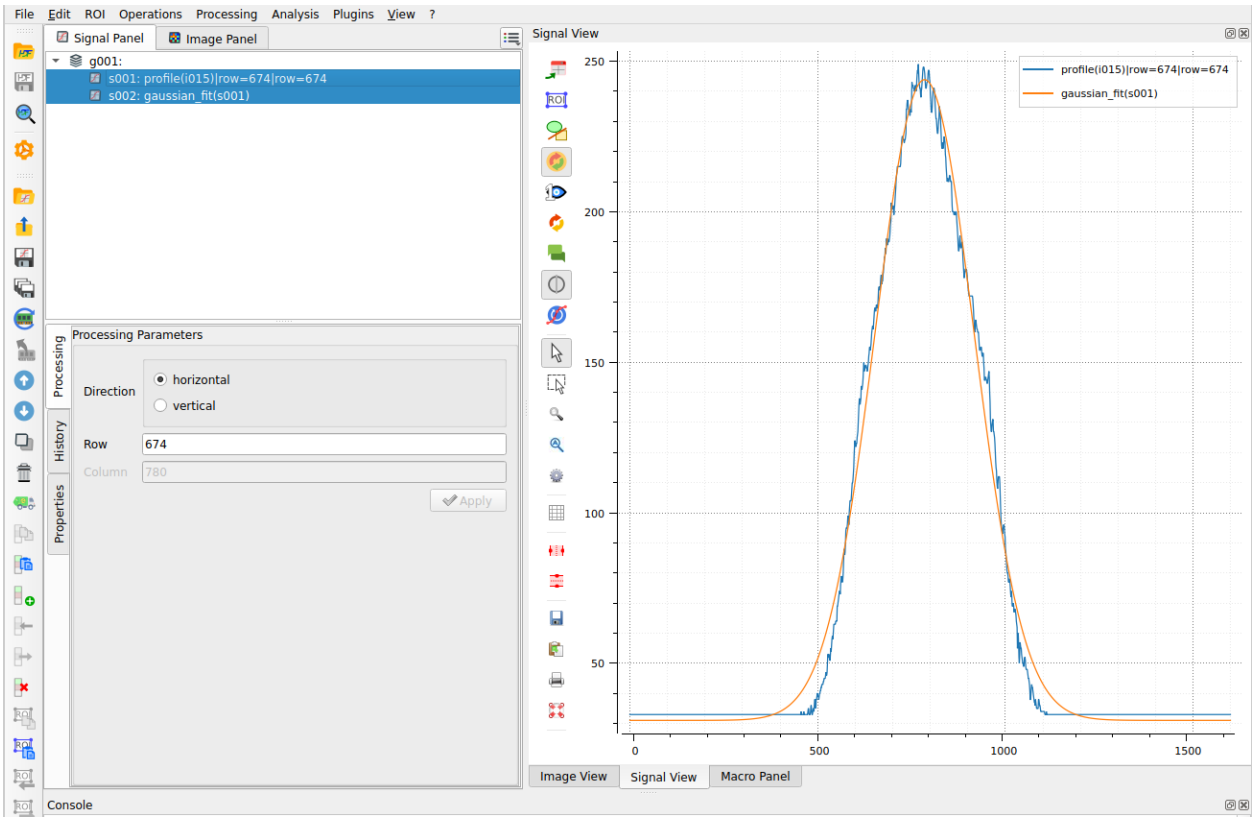


Fig. 85 – Le profil d'intensité ajusté à une fonction gaussienne. Ici, les deux signaux sont sélectionnés.

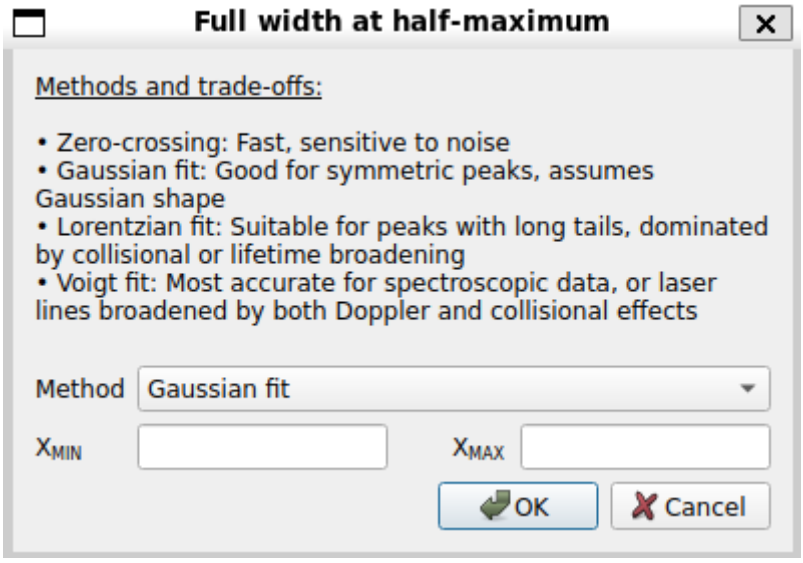


Fig. 86 – La fenêtre contextuelle qui permet de choisir la méthode pour estimer la FWHM.

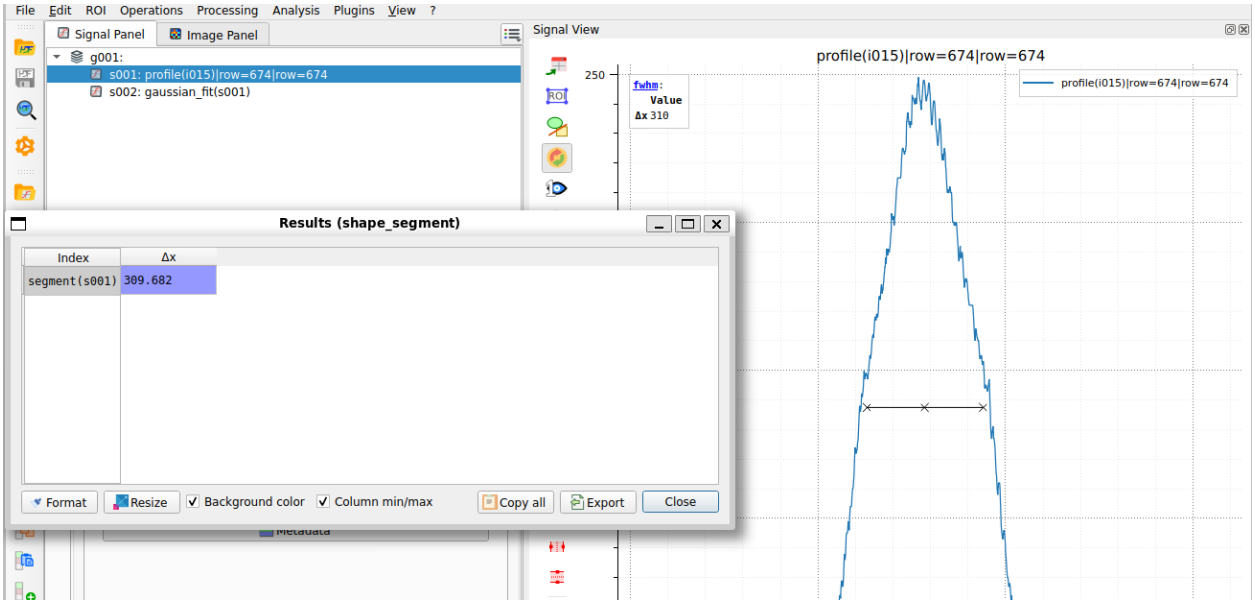


Fig. 87 – Le résultat de la FWHM affiché dans la fenêtre contextuelle et sur la courbe.

sité > 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.

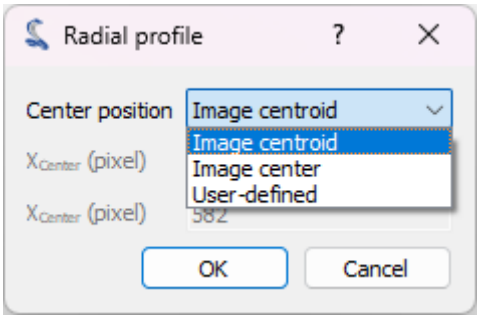


Fig. 88 – Les options disponibles pour le calcul du profil radial.

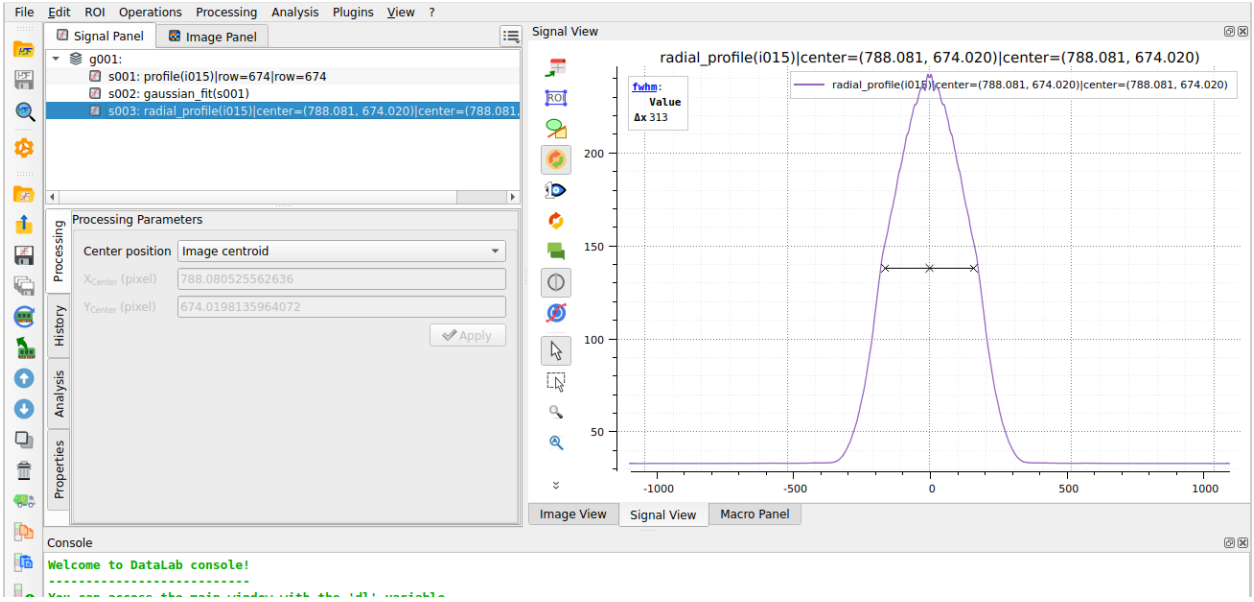


Fig. 89 – Le profil d'intensité radial affiché dans le panneau « Signaux ». Il est plus lisse que le profil de ligne, car il est calculé à partir d'un plus grand nombre de pixels, ce qui permet de moyenner le bruit.

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).

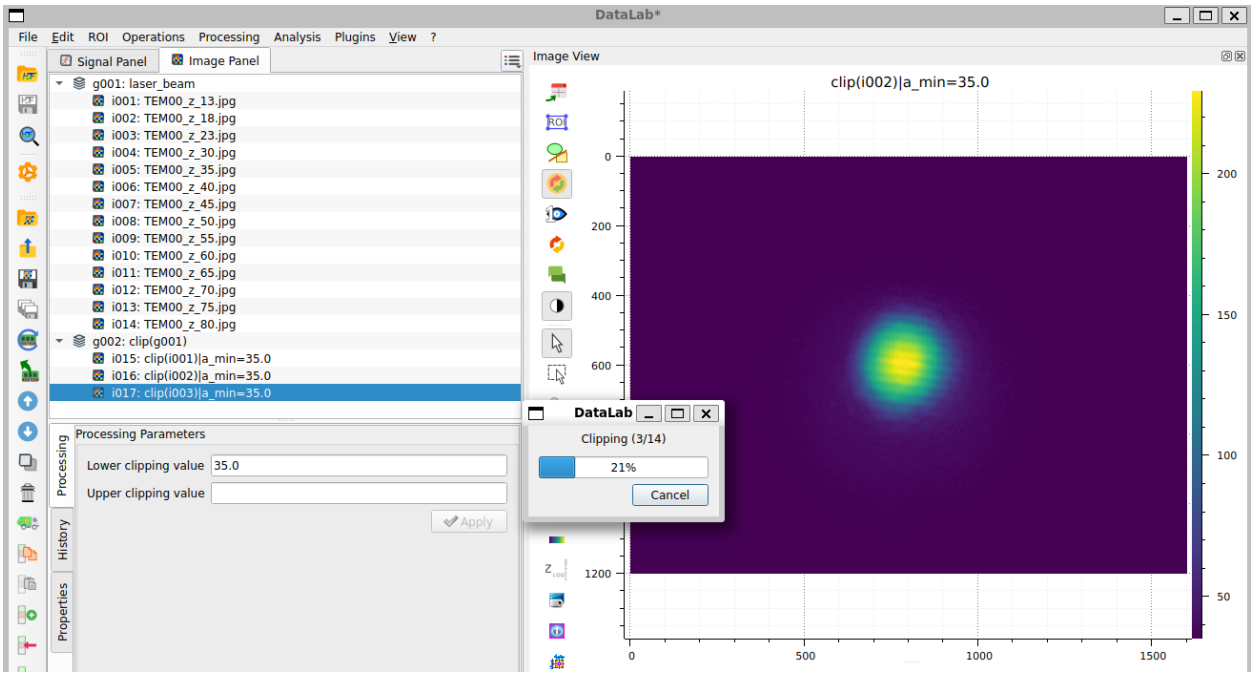


Fig. 90 – L'écrêtage s'applique à toutes les images du groupe.

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 :



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

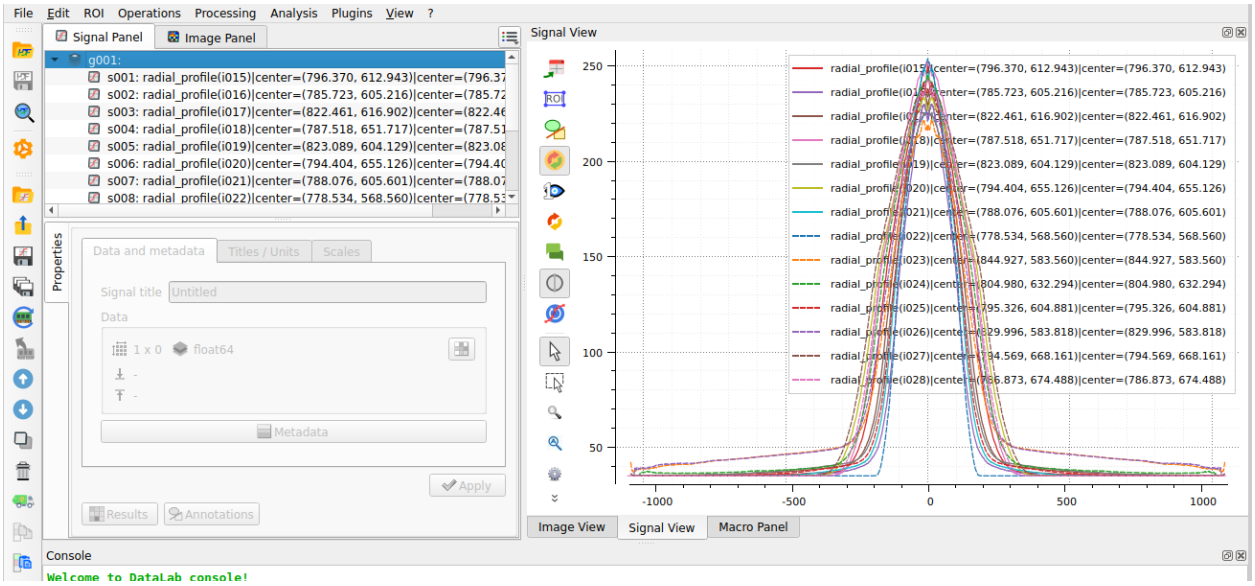


Fig. 91 – Le panneau « Signaux » contient maintenant tous les profils d'intensité radiaux.

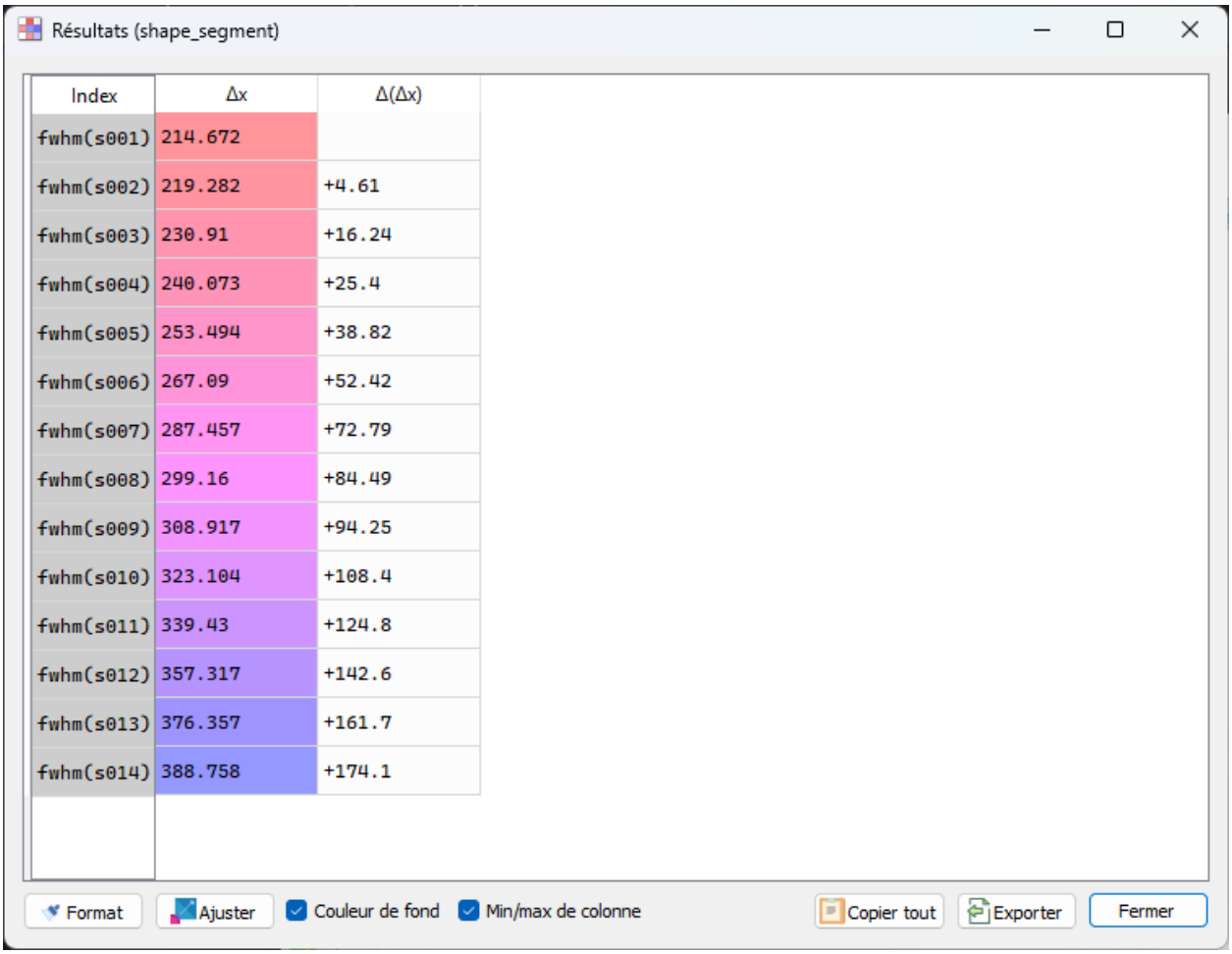


Fig. 92 – La boîte de dialogue « Résultats » affiche les valeurs FWHM pour tous les profils.

FWHM (L) en fonction de l'index de l'image (Indices).

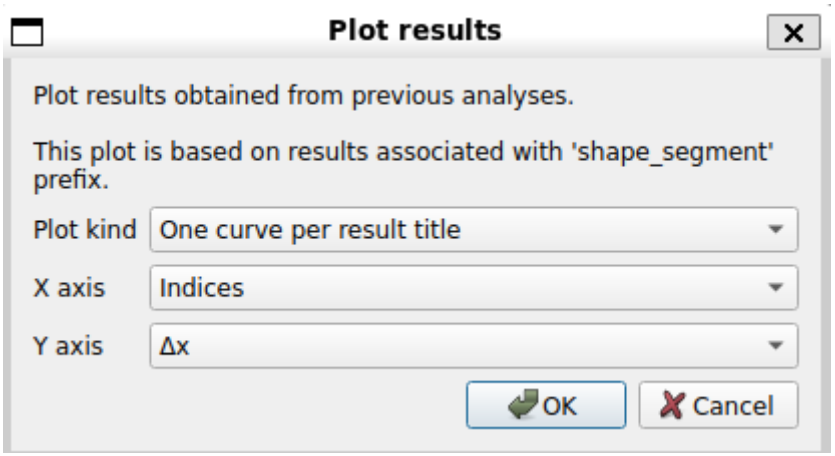


Fig. 93 – La fonction « Tracer les résultats » dans le menu « Analyse ».

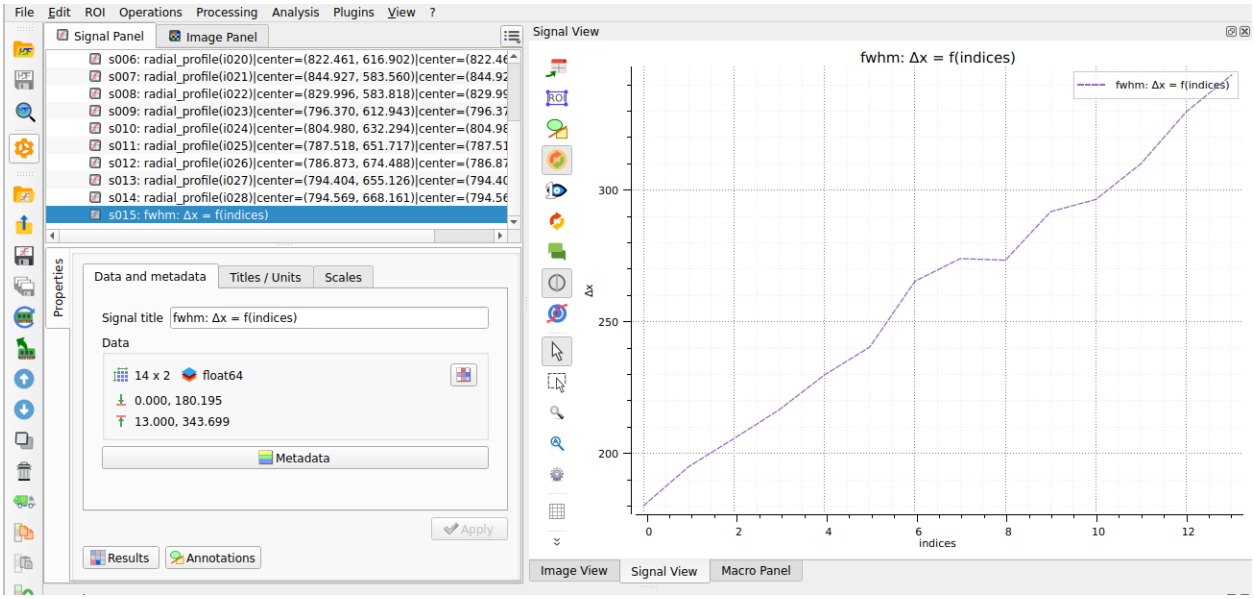


Fig. 94 – La taille du faisceau en fonction de la position le long de l'axe de propagation (la position est en unités arbitraires—l'index de l'image).

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 : $X\,[\mathrm{mm}] = 0.5 \cdot i + 10$ 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.

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

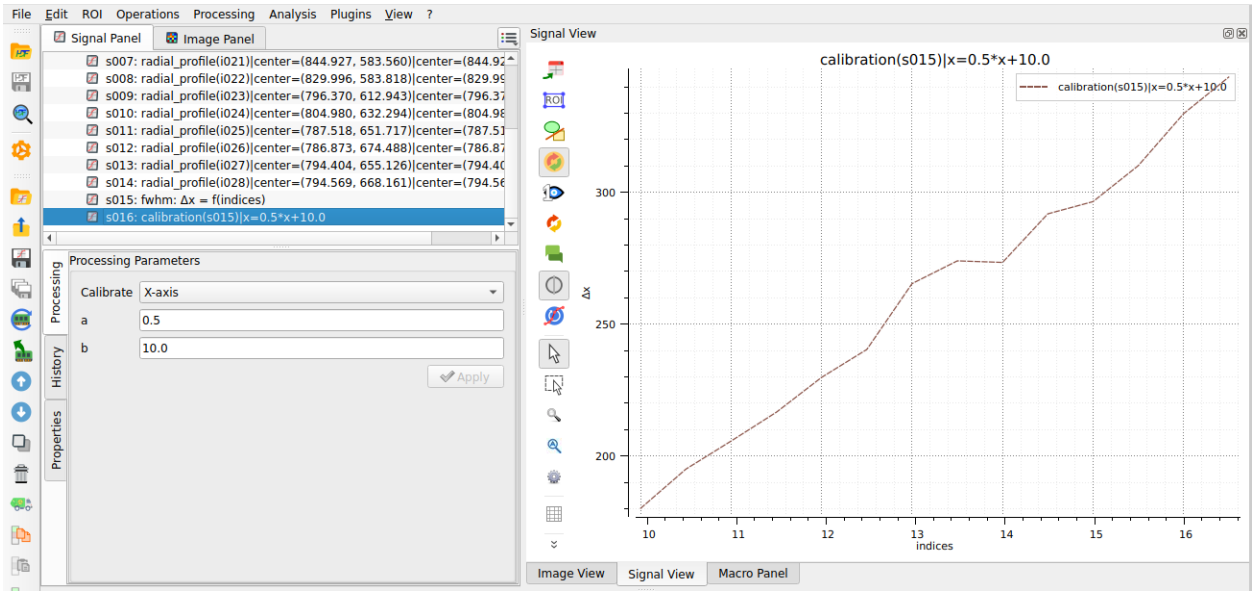


Fig. 95 – La taille du faisceau étalonné en fonction de la position le long de l'axe de propagation (la position est maintenant en mm).

Enregistrer et rejouer une chaîne de traitement

Ce tutoriel explique comment utiliser le panneau d'historique afin de rendre un flux de traitement reproductible :

- Enregistrer une chaîne de traitement complète sur un signal de test
- Inspecter les actions enregistrées et leur état de l'espace de travail
- Rejouer la chaîne, pas à pas et avec des paramètres modifiés
- Dupliquer une session pour comparer deux variantes de traitement

> **Voir aussi**
>
> La documentation de référence du panneau (barre d'outils, vue en arbre, règles de persistance et reconnexion de chaîne) est disponible dans la section Panneau d'historique.

Afficher le panneau d'historique

Le panneau d'historique est visible par défaut, mais il peut être masqué dans votre configuration. Dans ce cas, cochez l'entrée « Panneau d'historique » dans le menu « Affichage ».

Le panneau est vide lors de sa première ouverture. Il comprend trois parties : une barre d'outils en haut, l'arbre des sessions et actions enregistrées au centre (colonnes « Titre », « Date et heure » et « Description »), et les tableaux d'état de l'espace de travail en bas (« Signal » / « Forme » à gauche, « Image » / « Dimensions » à droite).

Comme les autres panneaux de DataLab, le panneau d'historique est ancrable : il peut être déplacé sur n'importe quel côté de la fenêtre principale ou détaché dans une fenêtre flottante. Ici, il est ancré à gauche, près du panneau des signaux, afin que les actions enregistrées restent visibles pendant le traitement des données. Cette configuration est utilisée dans la suite du tutoriel, mais vous pouvez choisir celle qui convient à votre flux.

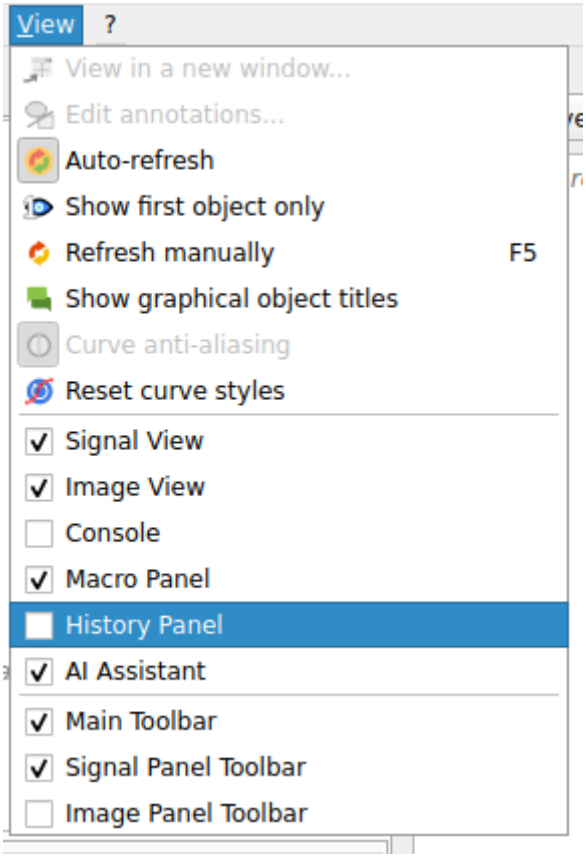


Fig. 96 – L'entrée « Affichage > Panneau d'historique », qui affiche ou masque le panneau.

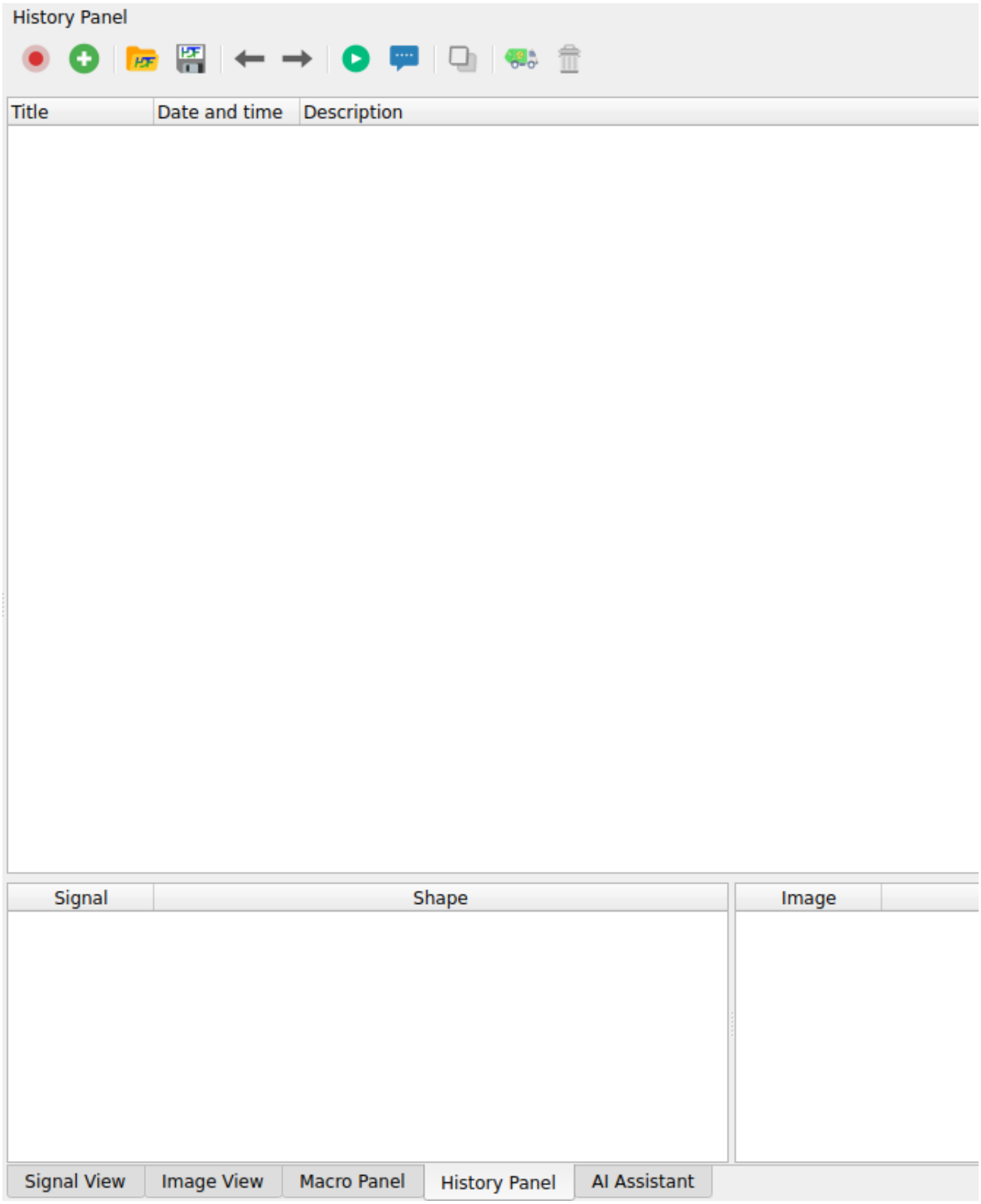


Fig. 97 – Le panneau d'historique vide : barre d'outils, arbre des actions et tableaux d'état de l'espace de travail.

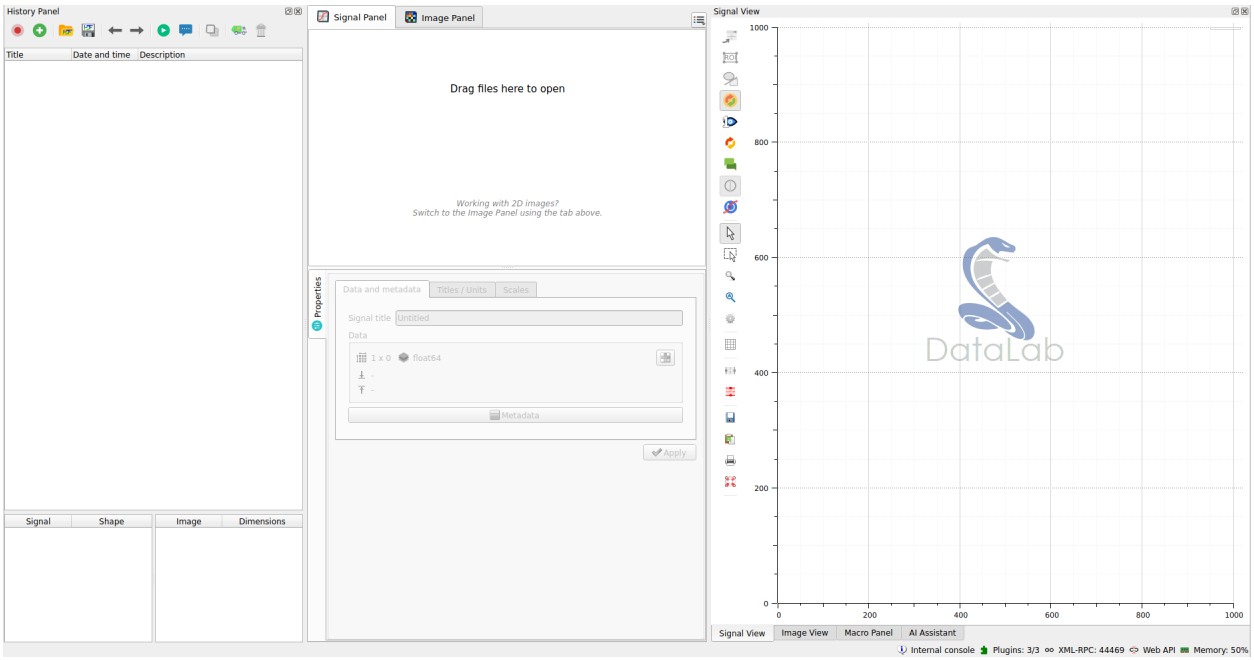


Fig. 98 – Le panneau d'historique ancré à gauche de la fenêtre principale.

Enregistrer une chaîne de traitement

L'enregistrement est désactivé par défaut : le panneau d'historique ne commence à collecter les actions qu'après l'activation du bouton Mode d'enregistrement de sa barre d'outils.

Le premier objet créé après l'activation de l'enregistrement ouvre une nouvelle session, qui contient toute la chaîne de traitement. Dès lors, chaque action produisant ou modifiant un objet – création d'un signal, exécution d'un calcul, suppression d'un objet – est ajoutée à cette session.

Lorsqu'un nouvel objet est ensuite créé, DataLab demande si la session courante doit être fermée et si une nouvelle doit être ouverte. Comme nous voulons enregistrer une unique chaîne de traitement continue, nous répondons Non afin de continuer à ajouter les actions à la session en cours.

Pour illustrer l'enregistrement d'une chaîne de traitement, créons un signal de test avec l'entrée « Créer > Gaussienne », puis ajoutons-lui du bruit avec « Traitement > Ajout de bruit > Bruit gaussien ».

L'arbre contient maintenant une session Traitement avec les deux actions enregistrées : la création du signal gaussien et l'ajout de bruit gaussien. L'action de bruit sélectionnée expose ses paramètres dans la colonne de description. Le panneau des signaux affiche les signaux d'origine et résultant, tandis que le panneau des paramètres de traitement affiche la graine, la moyenne et l'écart type utilisés par l'action sélectionnée.

Pour créer une chaîne plus longue, nous pouvons appliquer quelques étapes de traitement supplémentaires, par exemple un ajustement. Une fois terminé, l'action est enregistrée dans le panneau d'historique et le signal résultant est affiché dans le panneau des signaux.

Découvrir les commandes du panneau d'historique

Nous disposons maintenant d'une chaîne de traitement très simple, suffisante pour illustrer les principales fonctionnalités du panneau d'historique. La barre d'outils propose les commandes suivantes :

— Le Mode d'enregistrement active ou désactive l'enregistrement des actions suivantes.
— Nouvelle session ouvre une session d'historique distincte.

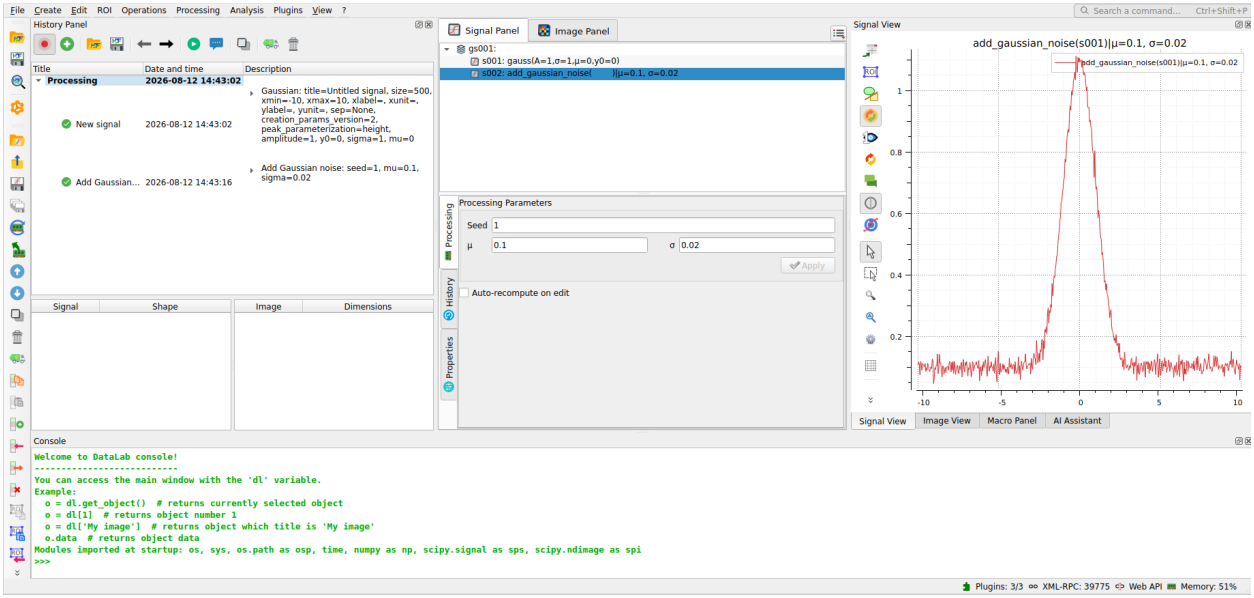


Fig. 99 – Le panneau d'historique après la création d'un signal gaussien et l'ajout de bruit.

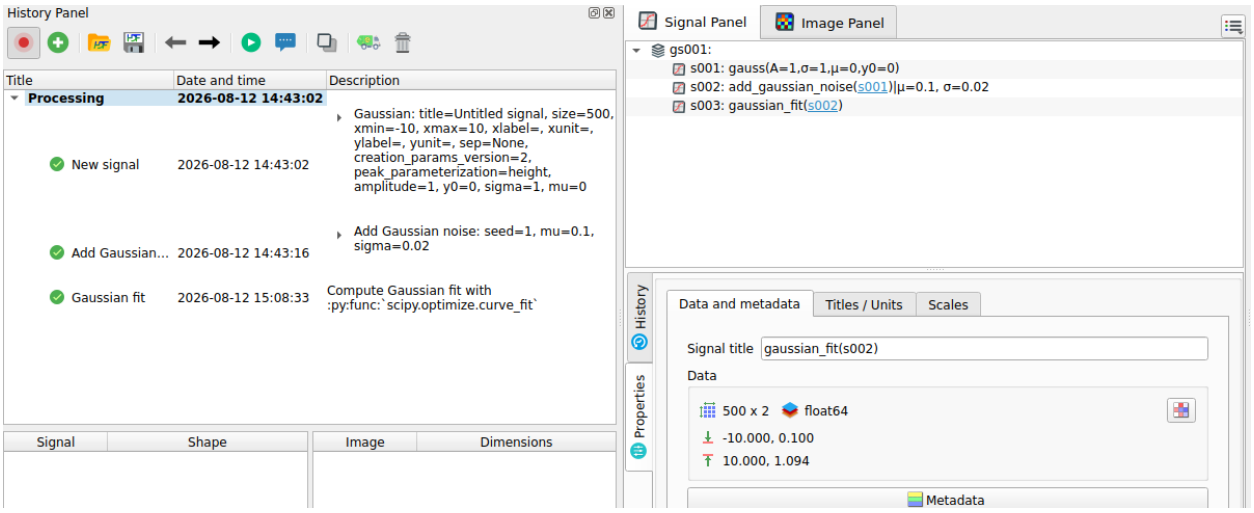


Fig. 100 – Le panneau d'historique après l'ajustement du signal bruité.

— Ouvrir un fichier d'historique et Enregistrer un fichier d'historique chargent ou enregistrent un fichier d'historique autonome `.dlhist`.
— Étape précédente et Étape suivante sélectionnent l'action voisine dans la session courante.
— Rejouer applique la session ou l'action sélectionnée sans afficher de dialogue de paramètres, tandis que Pas à pas la rejoue en permettant de modifier les paramètres à chaque étape.
— Dupliquer copie la session sélectionnée, ou la session complète contenant l'action sélectionnée, afin de comparer une variante de traitement.
— Supprimer les incompatibles supprime les actions qui ne peuvent pas être appliquées directement dans l'espace de travail courant (par exemple parce qu'un signal a été supprimé), et Supprimer efface l'entrée d'historique sélectionnée.

Lorsqu'une action est sélectionnée dans l'arbre, le signal résultant correspondant est affiché dans le panneau des signaux.

Sélectionnez maintenant le signal ajusté dans le panneau des signaux et supprimez-le avec le bouton « Supprimer » de la barre d'outils ou la touche « Supprimer ». L'action d'ajustement reste disponible dans le panneau d'historique, car elle peut être rejouée avec le bouton Rejouer. L'action de suppression est également enregistrée dans l'historique, mais elle est grisée : le signal ajusté a été supprimé, l'action ne peut donc pas être appliquée directement dans l'espace de travail courant.

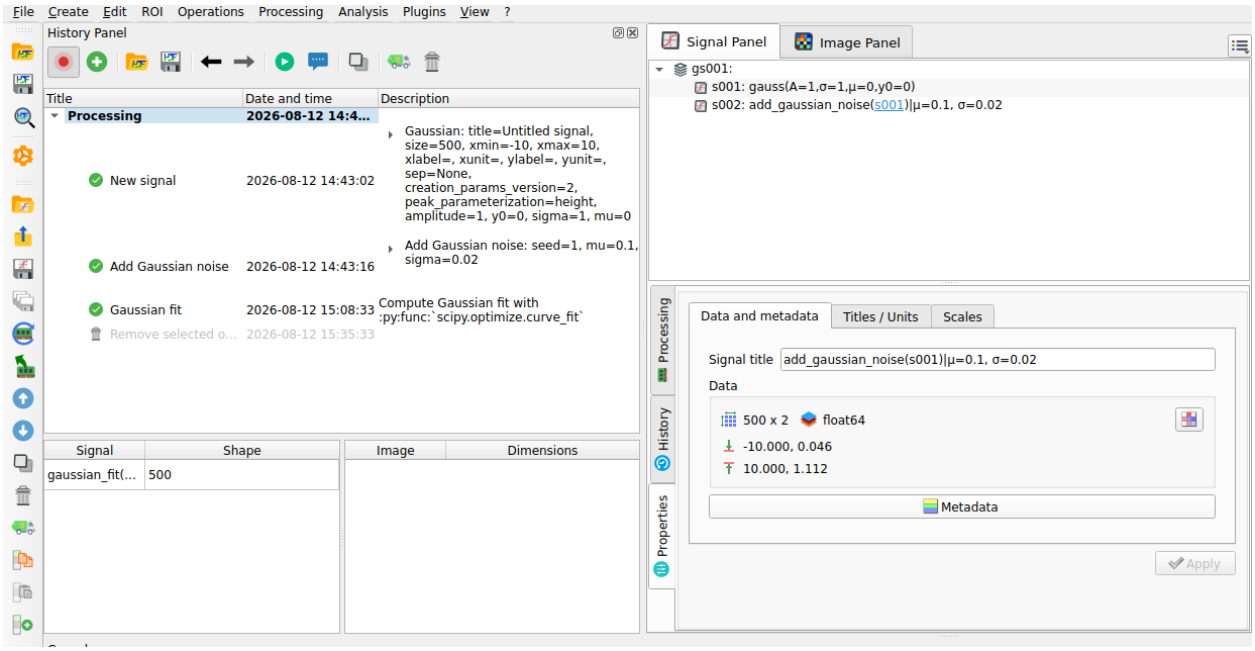


Fig. 101 – Le signal ajusté supprimé ne figure plus dans le panneau des signaux. La suppression enregistrée est grisée dans le panneau d'historique, car elle ne peut pas être rejouée dans l'espace de travail courant.

Nous pouvons maintenant rejouer l'action d'ajustement : le signal résultant est recréé dans le panneau des signaux et l'action « supprimer » n'est plus grisée, car elle peut désormais être appliquée au signal nouvellement créé.

Dupliquer la chaîne pour comparer des variantes

Nous pouvons maintenant dupliquer la chaîne pour comparer deux variantes de traitement. Commencez par sélectionner l'entrée « Supprimer les objets sélectionnés » dans le panneau d'historique, puis appuyez sur la touche Supprimer. Sélectionnez ensuite la session Traitement et cliquez sur Dupliquer. DataLab crée une nouvelle session Copie de traitement et un groupe `Copie - Traitement` dans le panneau des signaux. Ce groupe contient des copies indépendantes des signaux utilisés et produits par la chaîne d'origine.

Pour créer une variante, sélectionnez la session Copie de traitement et cliquez sur Pas à pas. DataLab affiche un dialogue de paramètres pour chaque action. Dans le dialogue Ajouter du bruit gaussien, modifiez le niveau de bruit, par exemple

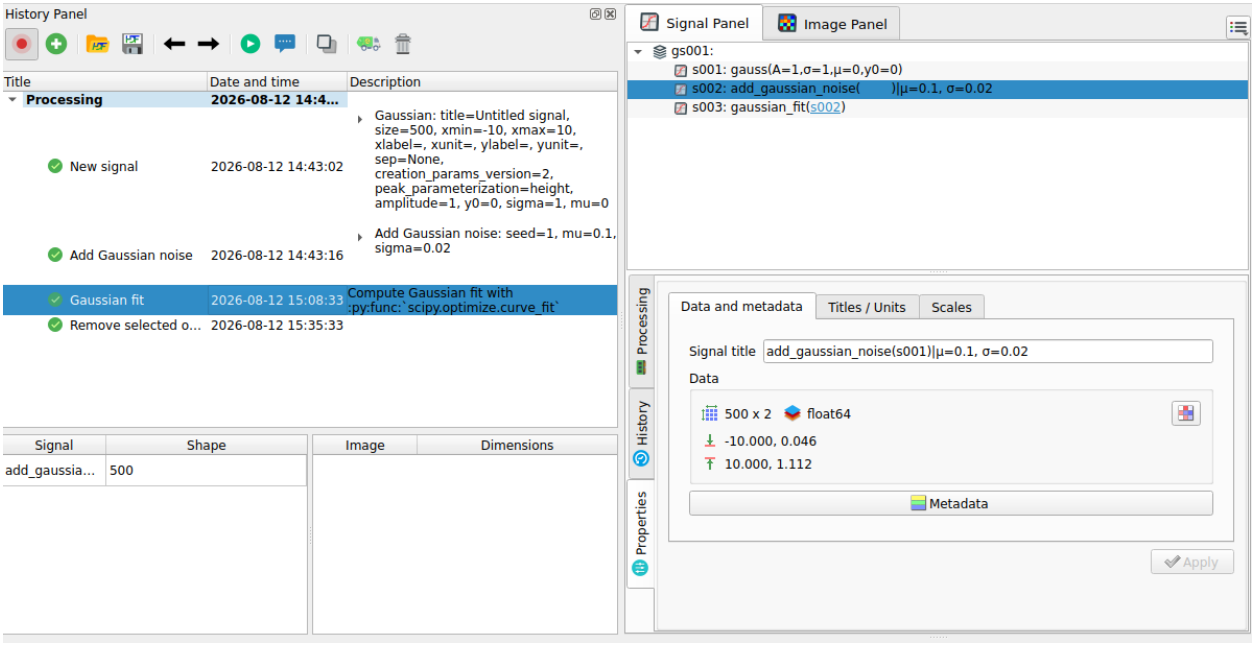


Fig. 102 – Le rejeu de l'ajustement recrée le signal ajusté. Toutes les actions enregistrées sont de nouveau compatibles avec l'espace de travail courant.

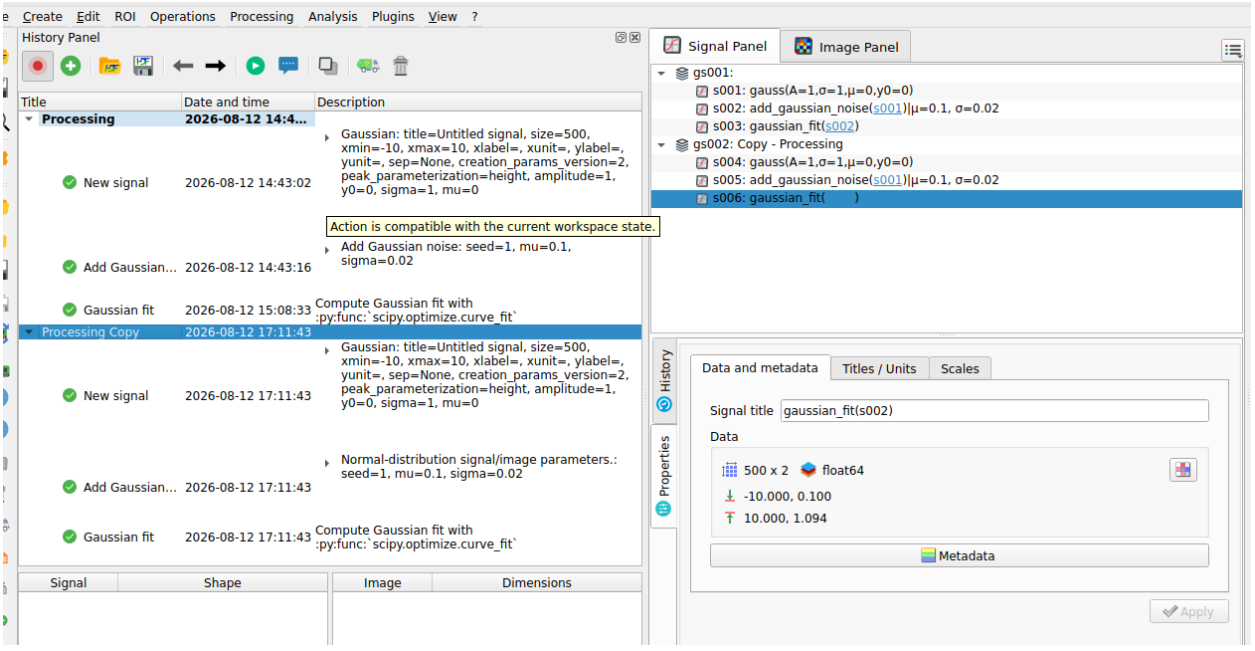


Fig. 103 – La session Copie de traitement contient les actions dupliquées. Le groupe `Copie - Traitement` contient les copies indépendantes des signaux.

en réglant `sigma` à `0.05`, puis validez les dialogues restants. La chaîne de signaux copiée est recalculée sur place : son signal bruité et son ajustement utilisent alors le nouveau niveau de bruit, tandis que la session Traitement d'origine reste inchangée.

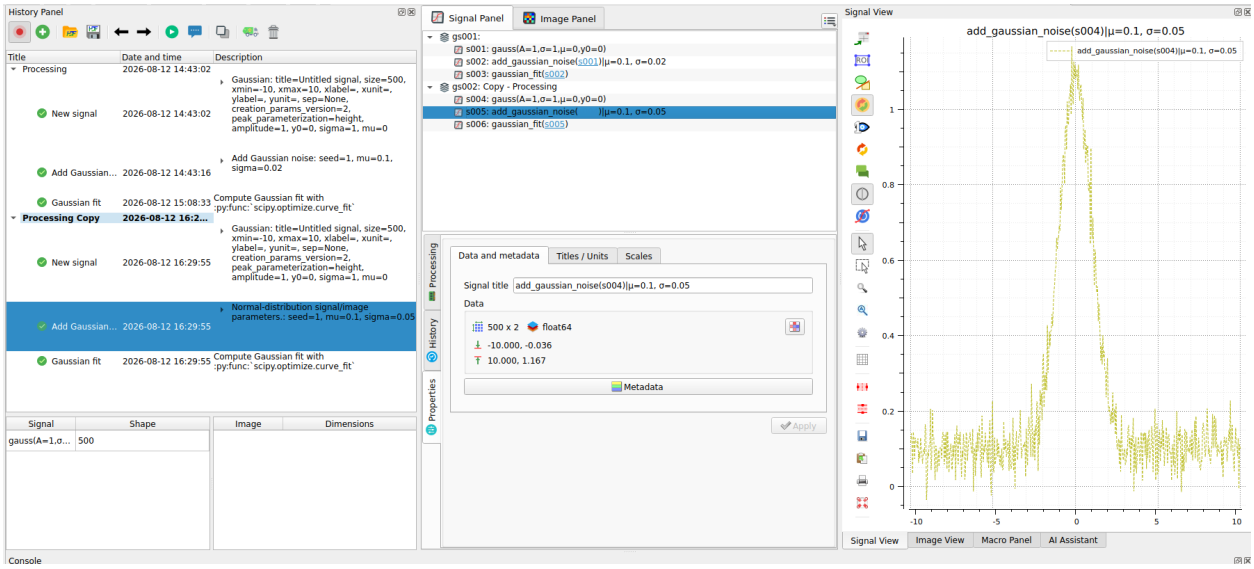


Fig. 104 – L'action de bruit sélectionnée dans Copie de traitement utilise `sigma = 0.05`. La chaîne d'origine et la copie modifiée restent disponibles côte à côte.

Cela peut servir à comparer l'effet du niveau de bruit sur le résultat de l'ajustement, ou à explorer d'autres variantes de traitement. Dans cet exemple, sélectionner les signaux ajustés d'origine et copié dans le panneau des signaux montre une différence, mais elle est faible.

Enregistrer un flux de traitement d'image vers signal

Le panneau d'historique enregistre les flux de travail des deux panneaux de données dans une même session. Contrairement aux panneaux des signaux et des images, il n'est pas séparé par type de données : son arbre chronologique affiche ensemble les opérations sur les images et les signaux qu'elles produisent.

Pour illustrer ce comportement, nous partirons d'un espace de travail vide. Vous pouvez aussi cliquer sur Nouvelle session dans le panneau d'historique afin de l'enregistrer dans une nouvelle session de l'espace de travail courant.

Dans les deux cas, activez l'enregistrement de l'historique et basculez vers le panneau des images.

Créez ou ouvrez une image de test et appliquez une première opération de réduction du bruit, par exemple Traitement > Réduction du bruit > Moyenne mobile. Comme vous le constatez, l'historique des images est enregistré de la même façon que celui des signaux.

Sélectionnez l'image filtrée et choisissez Analyse > Profils d'intensité > Profil linéaire…. Définissez la ligne horizontale dans le dialogue de profil, puis validez. DataLab ajoute le profil extrait au panneau des signaux, tandis que le panneau d'historique ajoute l'action Profil linéaire à la même session, après les actions de traitement d'image.

Cette unique session d'historique capture donc tout le flux de travail : création d'image, réduction du bruit et conversion des données d'image en signal. La sélection d'une action sur une image dans le panneau d'historique active le panneau des images ; celle de l'action de profil active le panneau des signaux et sélectionne le signal résultant.

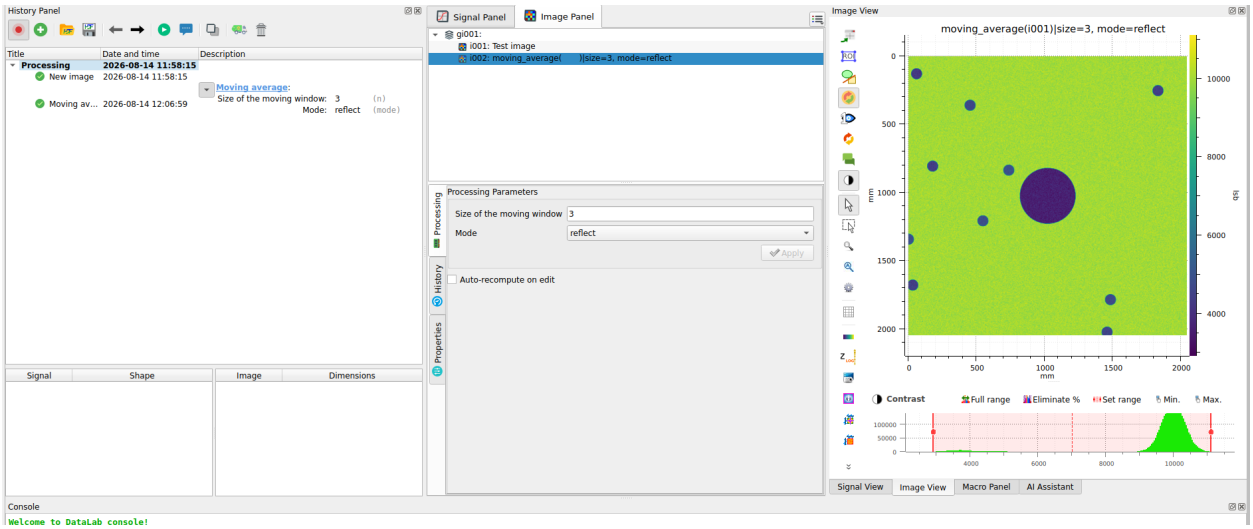


Fig. 105 – La nouvelle image et le résultat de sa moyenne mobile sont enregistrés dans la même session du panneau d'historique.

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.
    """
```

(suite sur la page suivante)

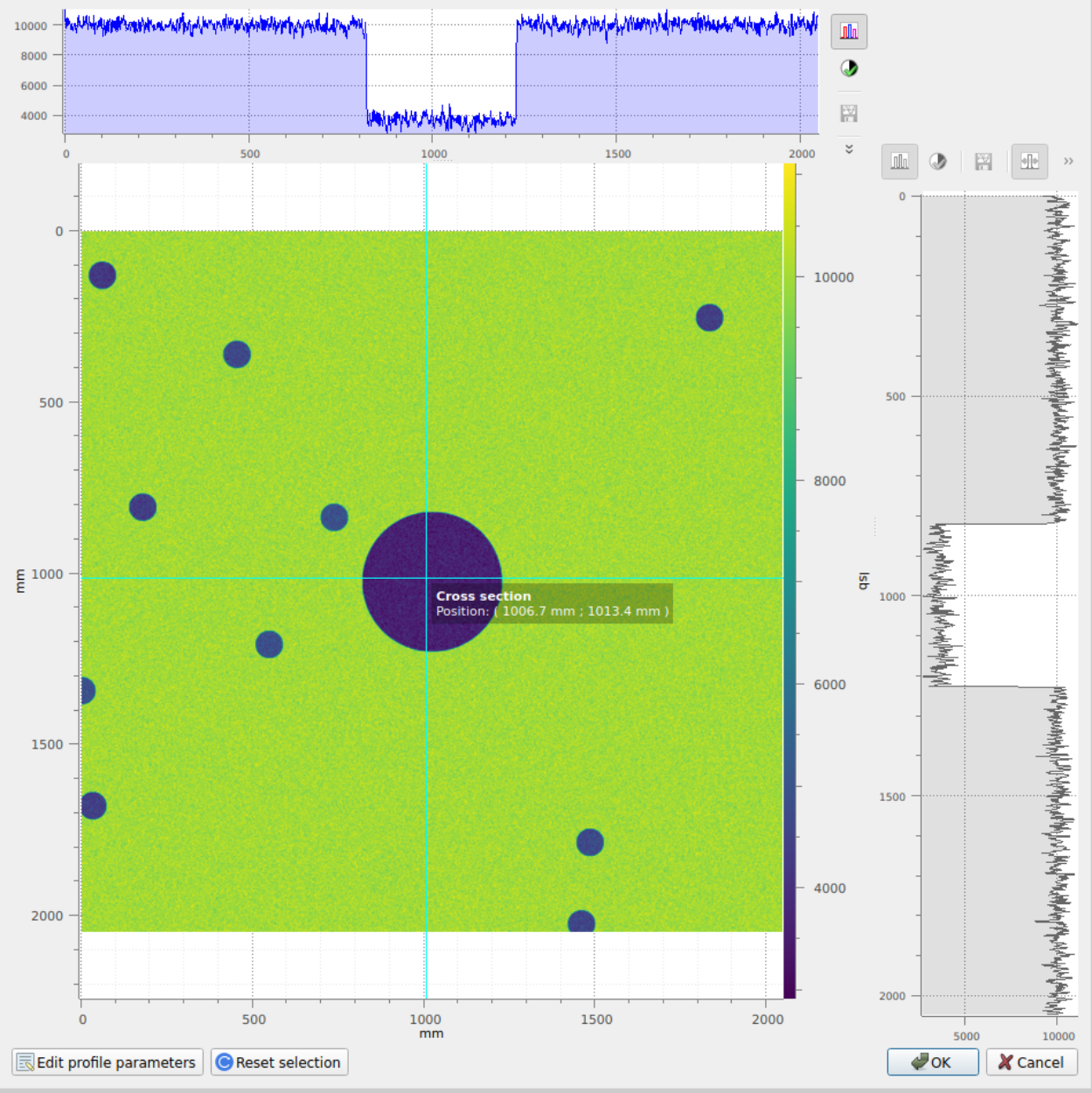


Fig. 106 – Choisissez la ligne horizontale utilisée pour extraire le profil de l'image filtrée.

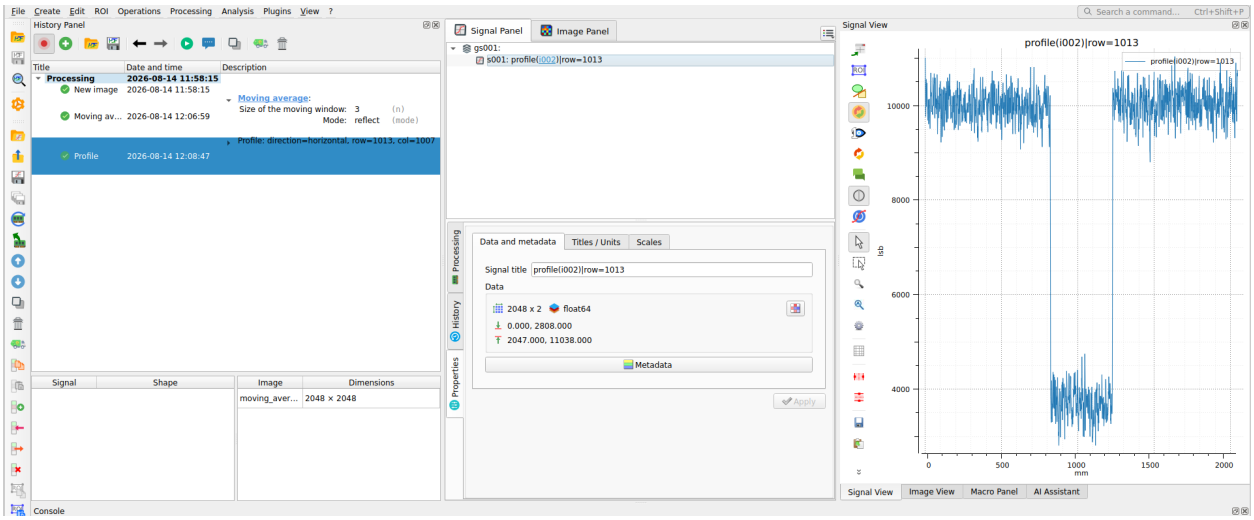


Fig. 107 – Le panneau d'historique enregistre les opérations sur l'image et le signal résultant dans une unique session chronologique.

(suite de la page précédente)

```
    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 automatisée de défauts et de particules sur des images 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 ».

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).

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

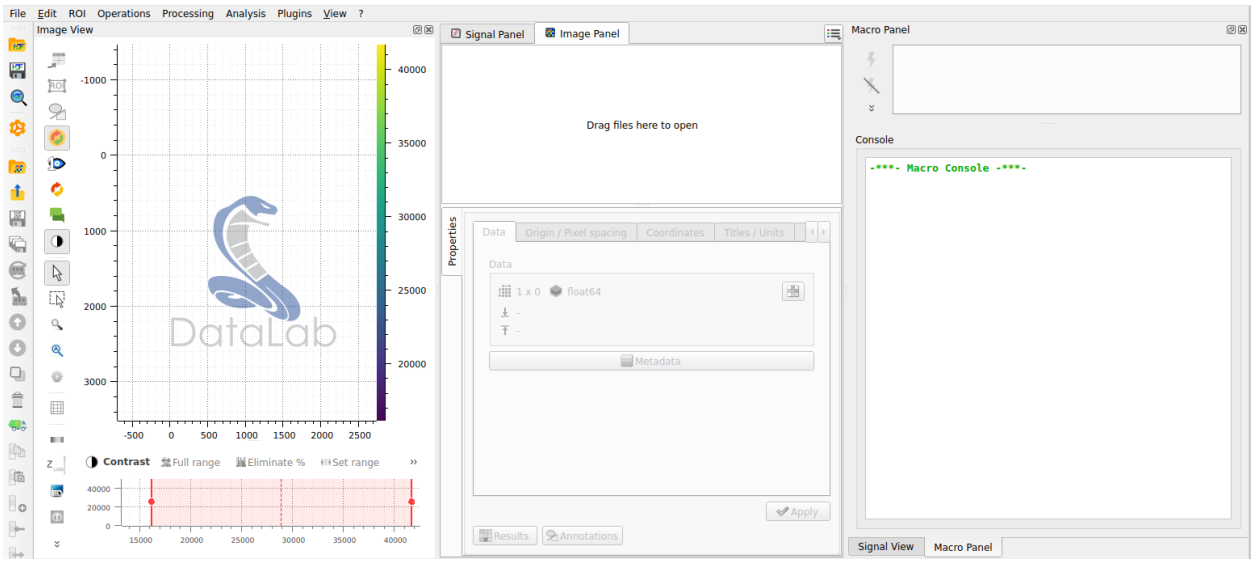


Fig. 108 – La disposition de la fenêtre de DataLab réorganisée pour avoir le « Panneau Image » à gauche et le « Gestionnaire de Macros » à droite.

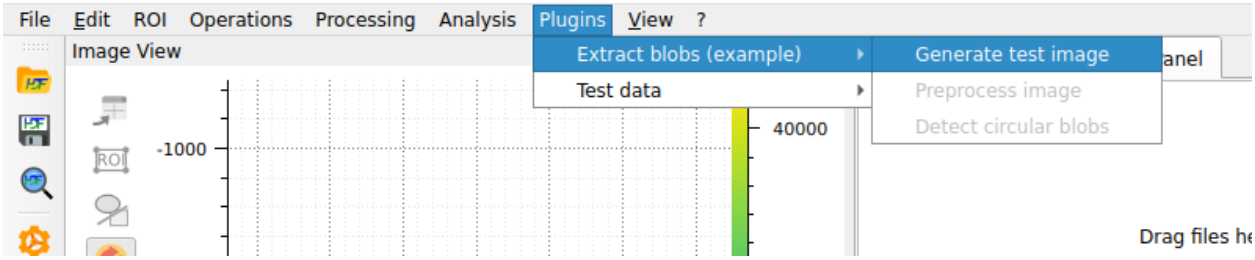


Fig. 109 – Nous générons une nouvelle image en utilisant le menu « Plugins > Extract blobs (example) > Generate test image ».

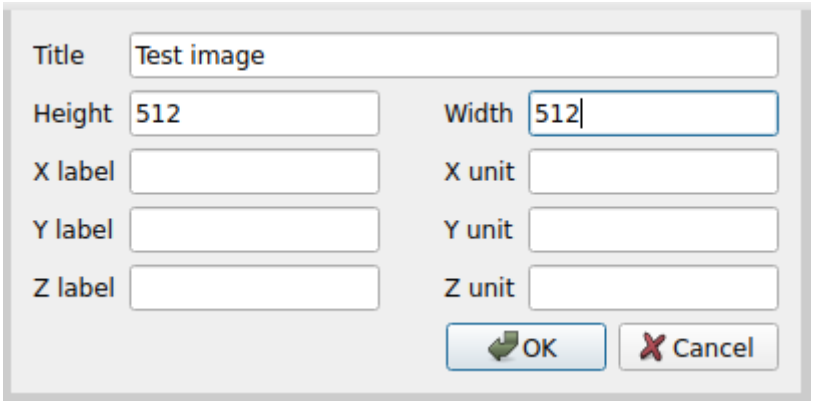


Fig. 110 – Nous sélectionnons la taille de l'image et cliquons sur « OK ».

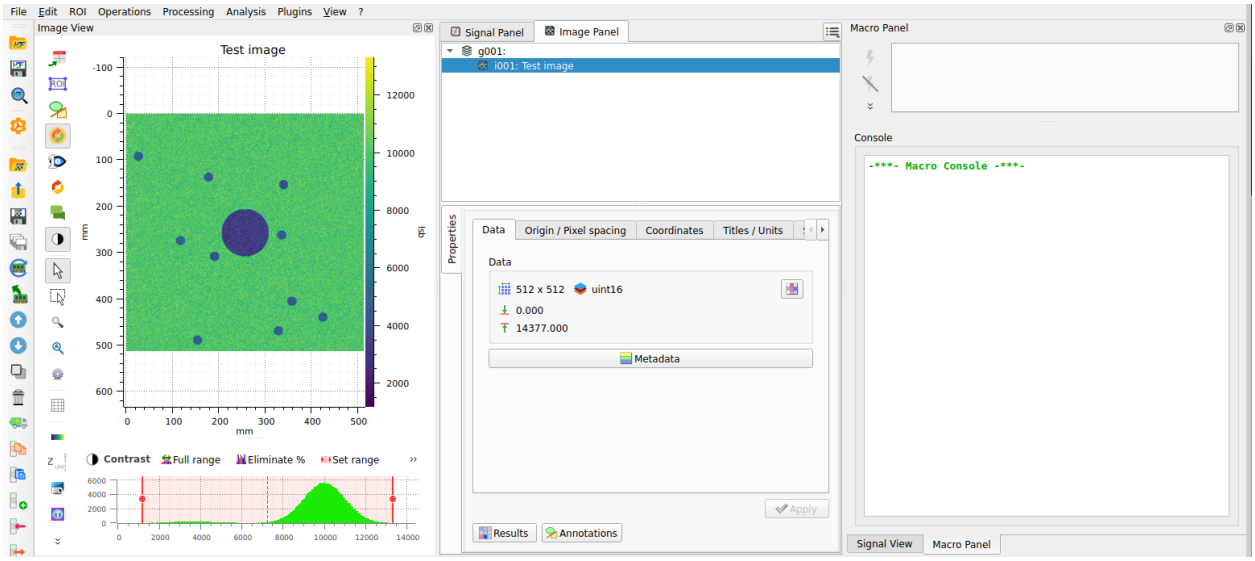


Fig. 111 – L'image générée dans le « Panneau Image ».

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.

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]
    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!")
```

(suite sur la page suivante)

(suite de la page précédente)

```
# 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

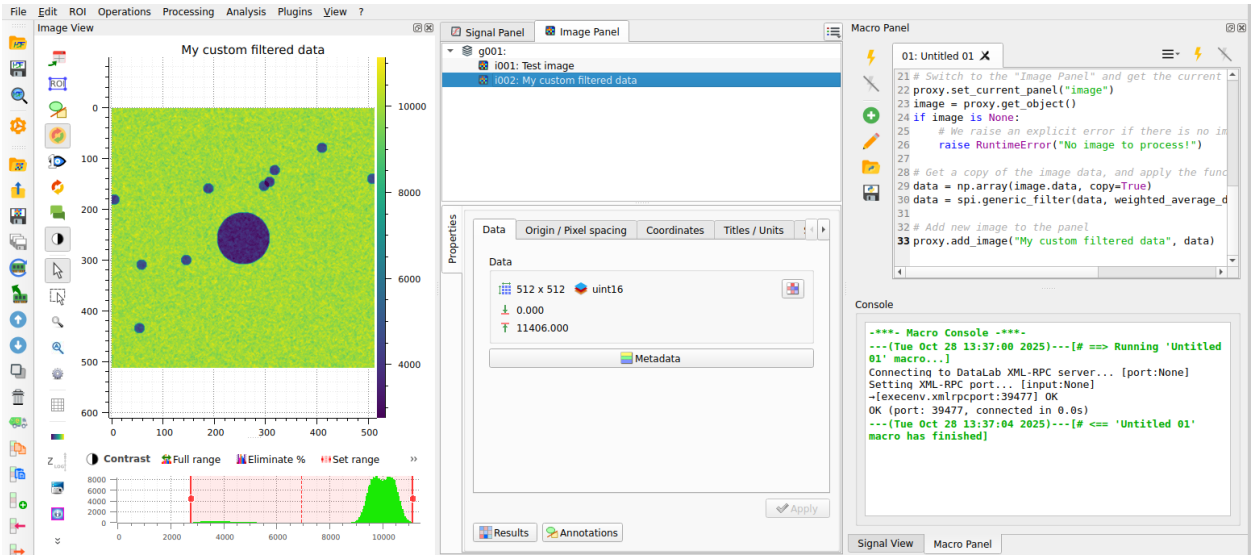


Fig. 112 – Lorsque la macro-commande est terminée, nous pouvons voir la nouvelle image dans le « Panneau Image ». Notre filtre a été appliqué à l'image, et nous pouvons voir que le bruit a été réduit.

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
    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)

```

(suite sur la page suivante)

(suite de la page précédente)

```
42
43  # Switch to the "Image Panel" and get the current image
44  proxy.set_current_panel("image")
45  image = proxy.get_object()
46  if image is None:
47      # We raise an explicit error if there is no image to process
48      raise RuntimeError("No image to process!")
49
50  # Get a copy of the image data, and apply the function to it
51  data = np.array(image.data, copy=True)
52  data = weighted_average_denoise(data)
53
54  # Add new image to the panel
55  proxy.add_image("Filtered using Spyder", data)
```

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

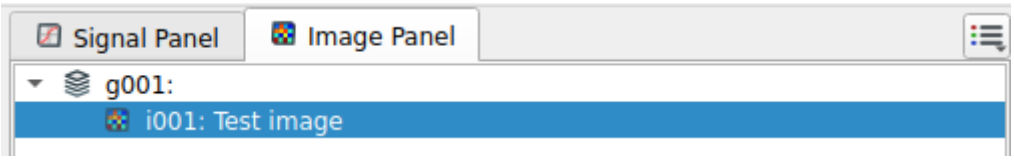


Fig. 113 – La première image dans le « Panneau Image » sélectionnée.

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.

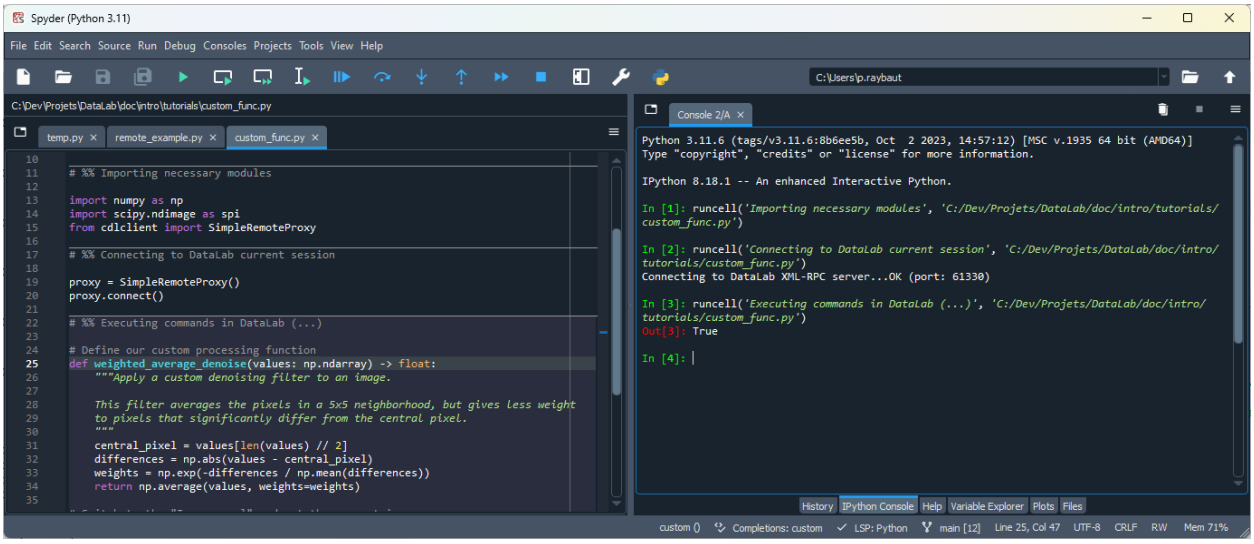


Fig. 114 – L'environnement de l'éditeur Spyder avec le script de traitement personnalisé.

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.

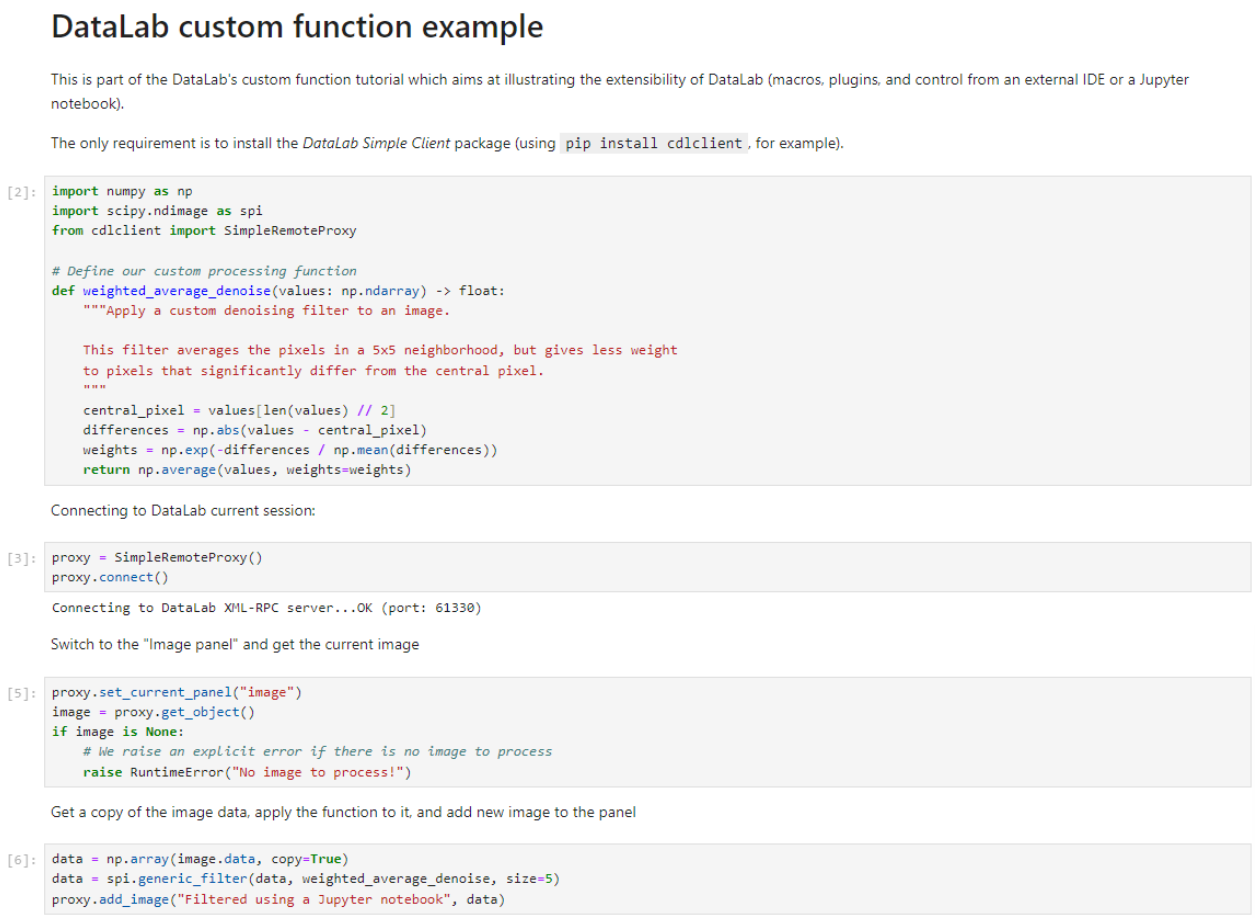
Ici, nous avons utilisé le script sans aucune modification, mais nous aurions pu le modifier pour tester de nouvelles idées, puis utiliser le script modifié dans DataLab.

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.

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.



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

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.

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 :

```python
# -*- 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.
```

(suite sur la page suivante)

(suite de la page précédente)

```
    """

    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"""

    FEATURE_ID = "org.datalab.examples.custom-filters:weighted-average-denoise"
    PLUGIN_INFO = datalab.plugins.PluginInfo(
        id="org.datalab.examples.custom-filters",
        name="My custom filters",
        version="1.0.0",
        description="This is an example plugin",
        capabilities=(datalab.plugins.PluginCapability.PROCESSING,),
    )

    def register_computations(self) -> None:
        """Register computations owned by this plugin."""
        wrapped_func = sipi.Wrap1to1Func(weighted_average_denoise)
        self.imagepanel.processor.register_1_to_1(
            wrapped_func,
            "Weighted average denoise",
            feature_id=self.FEATURE_ID,
            owner_plugin_id=self.plugin_id,
        )

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

Pour le tester, nous devons ajouter le script du plugin dans l'un des répertoires de plugins découverts par DataLab au démarrage : vous pouvez le trouver via « Fichier > Paramètres… » (voir la section Plugins pour plus de détails, ou Détection automatisée de défauts et de particules sur des images pour un exemple). Nous rechargeons ensuite les plugins depuis le menu « Plugins » (ou redémarrons DataLab) et nous pouvons voir que le plugin a été chargé.

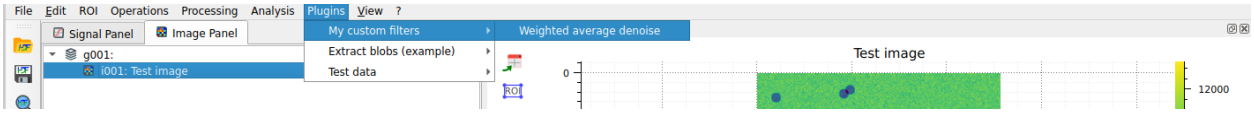


Fig. 116 – Après le rechargement des plugins (ou le redémarrage de DataLab), nous pouvons voir que le plugin a été chargé.

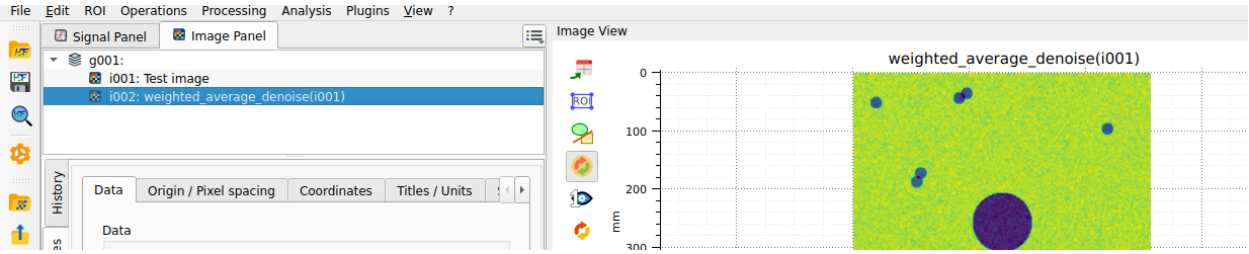
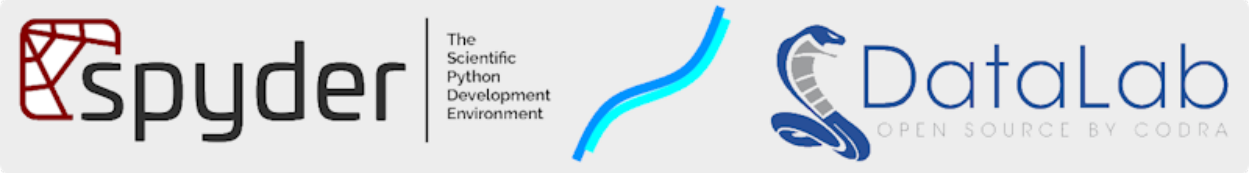


Fig. 117 – Nous générons à nouveau notre image de test en utilisant (voir les premières étapes du tutoriel), et nous la traitons en utilisant le plugin : « Plugins > My custom filters > Weighted average denoise ».

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
```

(suite sur la page suivante)

(suite de la page précédente)

```
# %% 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
```

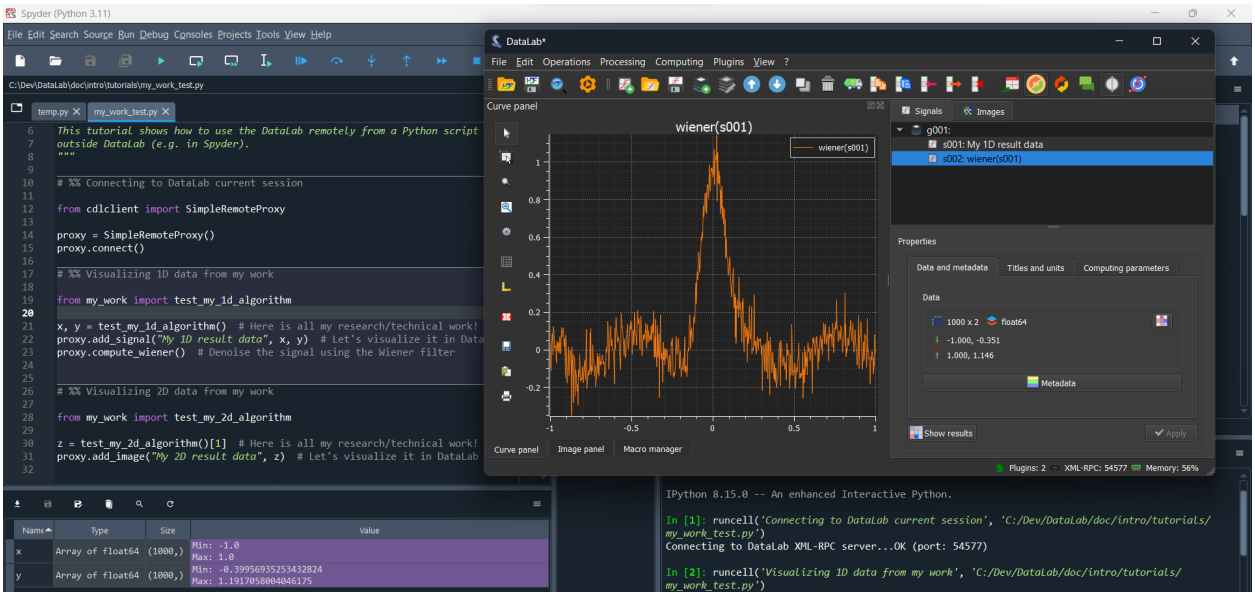


Fig. 118 – Dans cette capture d'écran, nous pouvons voir le résultat de l'évaluation des deux premières cellules : la première cellule se connecte à DataLab, et la seconde cellule visualise les données 1D retournées par la fonction `test_my_1d_algorithm`.

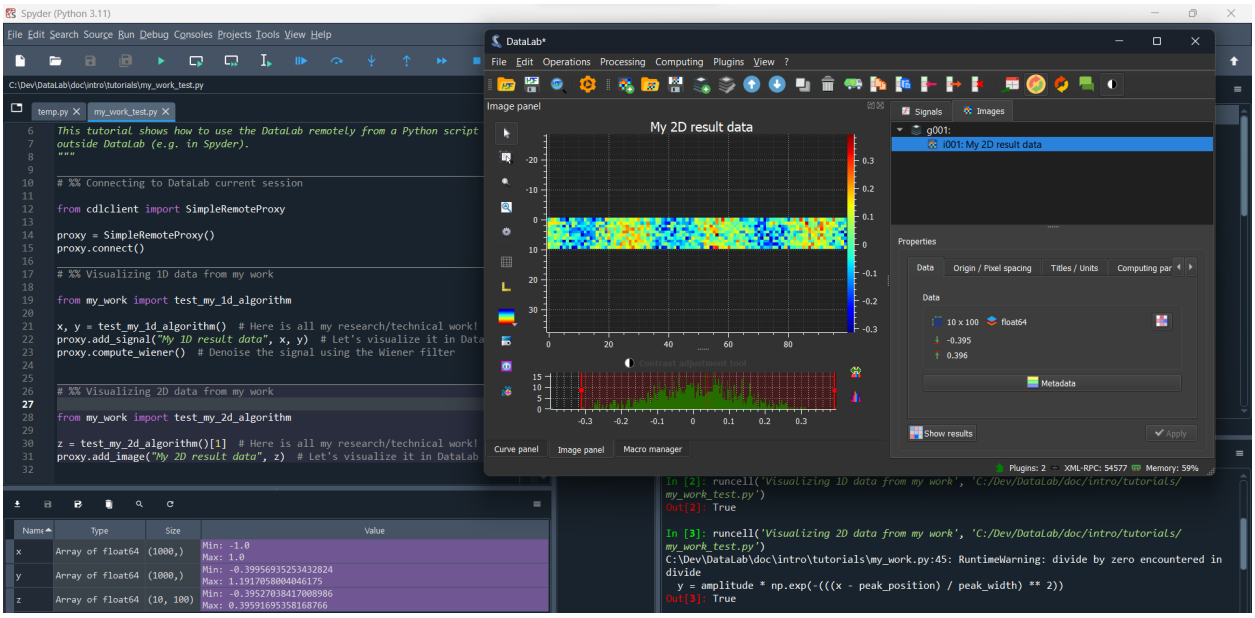


Fig. 119 – Dans cette capture d'écran, nous pouvons voir le résultat de l'évaluation de la troisième cellule : la fonction `test_my_2d_algorithm` retourne un tableau 2D, que nous pouvons visualiser directement dans DataLab.

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).
```

(suite sur la page suivante)

(suite de la page précédente)

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

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

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

```
# %% Debugging my work with DataLab

from my_work import generate_2d_data

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.

> **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.

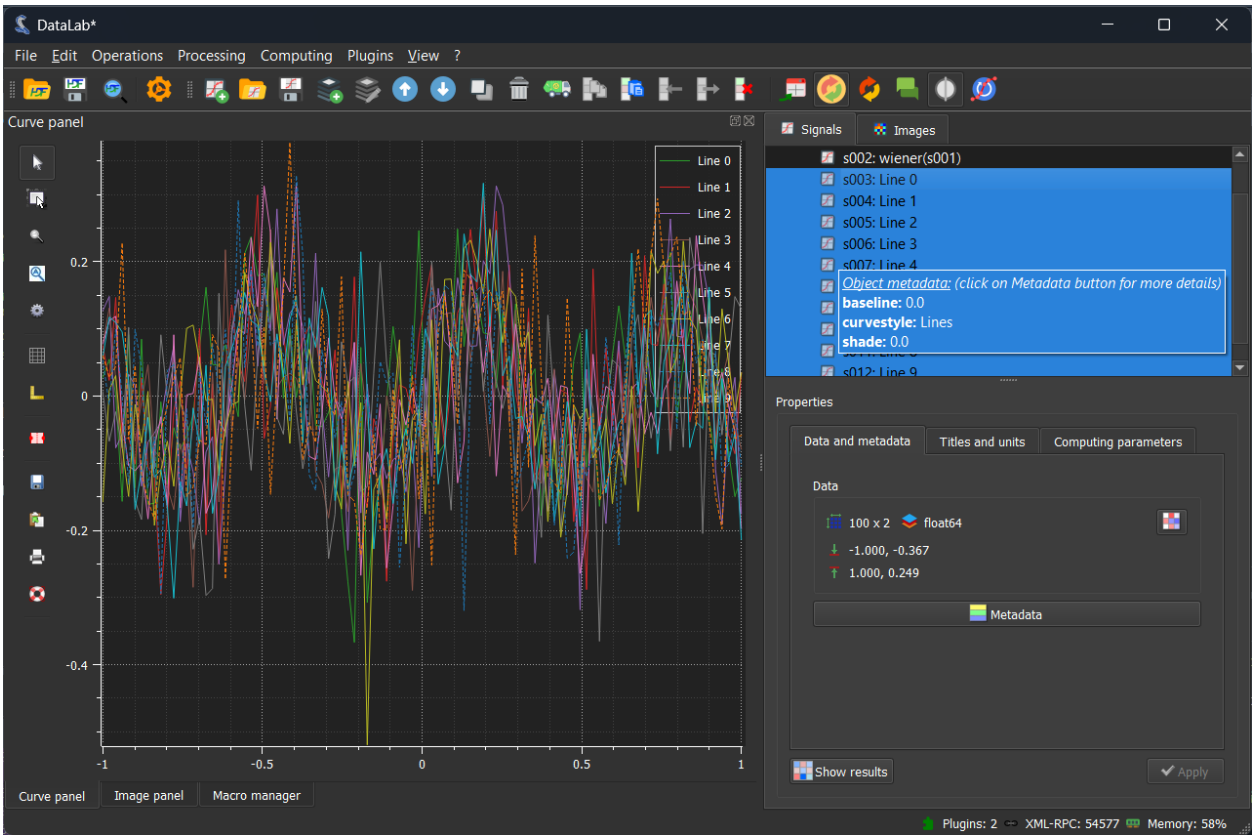


Fig. 120 – Dans cette capture d'écran, nous pouvons voir le résultat de l'évaluation de la première cellule : la fonction `test_my_2d_algorithm` est appelée avec le paramètre `debug_with_datalab` défini à `True` : 10 tableaux 1D sont générés et visualisés dans DataLab.

CHAPITRE 2

Cas d'usage

DataLab est utilisé en recherche et dans l'industrie pour analyser les données produites par les instruments scientifiques – caméras, numériseurs, spectromètres – dans un espace de travail unique, sans avoir à développer une application sur mesure. Créé à l'origine pour les diagnostics plasma du Laser Mégajoule du CEA, où il est utilisé en production, il a depuis été appliqué à de nombreux autres domaines.

Les pages ci-dessous montrent comment DataLab répond à un problème concret dans chaque domaine, avec des données réelles et un tutoriel pas à pas.

2.1 DataLab pour la spectroscopie

2.1.1 Le problème

Les spectres produits par un instrument – Raman, FTIR, XRF ou tout balayage en énergie ou longueur d'onde – sortent rarement propres : bruit, dérive de la ligne de base, pics qui se recouvrent. Les options habituelles sont le logiciel du fabricant (fermé, lié à un instrument), le tableur (source d'erreurs, sans traçabilité) ou les scripts maison (des semaines de travail, difficiles à partager avec des non-programmeurs).

2.1.2 Ce que fait DataLab

DataLab permet d'enchaîner toute l'analyse de manière interactive, en gardant chaque étape de traitement enregistrée avec ses paramètres :

- filtrer le bruit (Wiener, moyenne mobile, etc.),
- corriger la ligne de base, y compris par détendance linéaire restreinte aux régions sans pics – une approche robuste quand les pics dominent le signal,
- sélectionner une région d'intérêt et ajuster un modèle (gaussien, lorentzien, Voigt, multi-pics…),
- enregistrer tout l'espace de travail (données + historique de traitement + métadonnées) dans un unique fichier HDF5 que vos collègues peuvent rouvrir – dans l'application de bureau ou dans le navigateur.

Chaque opération est aussi accessible depuis Python (scripts, Jupyter, macros) : une analyse interactive peut ainsi devenir un pipeline automatisé traitant une série entière de spectres.

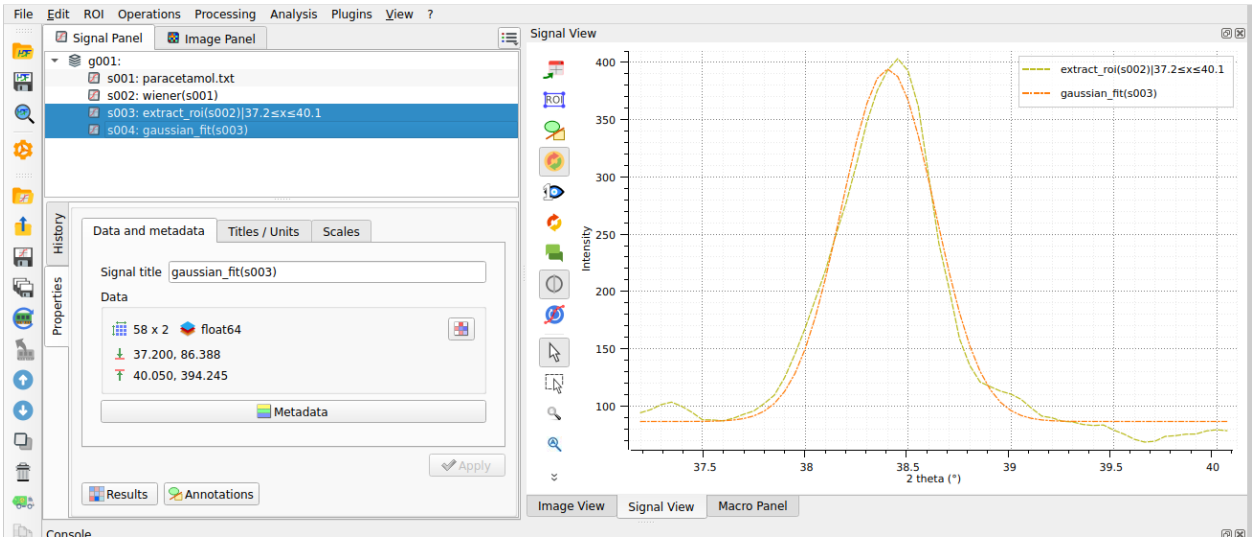


Fig. 1 – Ajustement gaussien sur une région d'intérêt d'un spectre dans DataLab.

2.1.3 Éprouvé en production

DataLab a été conçu pour le Laser Mégajoule du CEA, où il est utilisé en production pour traiter et visualiser les données des spectromètres dans le cadre des diagnostics plasma.

2.1.4 Essayer

— Sans installation : ouvrez DataLab-Web avec un spectre de démonstration déjà chargé dans votre navigateur – vos données ne quittent jamais votre machine.
— Tutoriel pas à pas : Analyse de spectre : débruitage, correction de ligne de base et ajustement de pics.
— Ou installez l'application de bureau.

2.2 DataLab pour la photonique et les lasers

2.2.1 Le problème

Aligner un banc laser ou caractériser un montage optique consiste à extraire des mesures quantitatives d'images caméra : taille du faisceau à plusieurs positions le long de l'axe de propagation, rayons des franges d'un interférogramme, profils d'intensité. Le faire avec le logiciel de profilométrie du fabricant vous lie à une caméra ; le faire avec des scripts oblige à reconstruire de zéro la gestion des ROI, la calibration et la visualisation.

2.2.2 Ce que fait DataLab

DataLab travaille sur les images et sur les signaux qui en sont extraits, dans le même espace de travail :
— charger d'un coup un dossier entier d'images caméra,
— seuiller, définir des régions d'intérêt, appliquer une calibration linéaire,
— extraire des profils d'intensité rectilignes ou radiaux et les ajuster (gaussienne et autres modèles) pour obtenir la FWHM et la taille du faisceau,
— détecter des contours et les ajuster par des cercles pour mesurer les rayons des franges d'un interférogramme,
— répéter la même analyse sur une pile d'images et tracer, par exemple, la taille du faisceau en fonction de la position le long de l'axe de propagation.

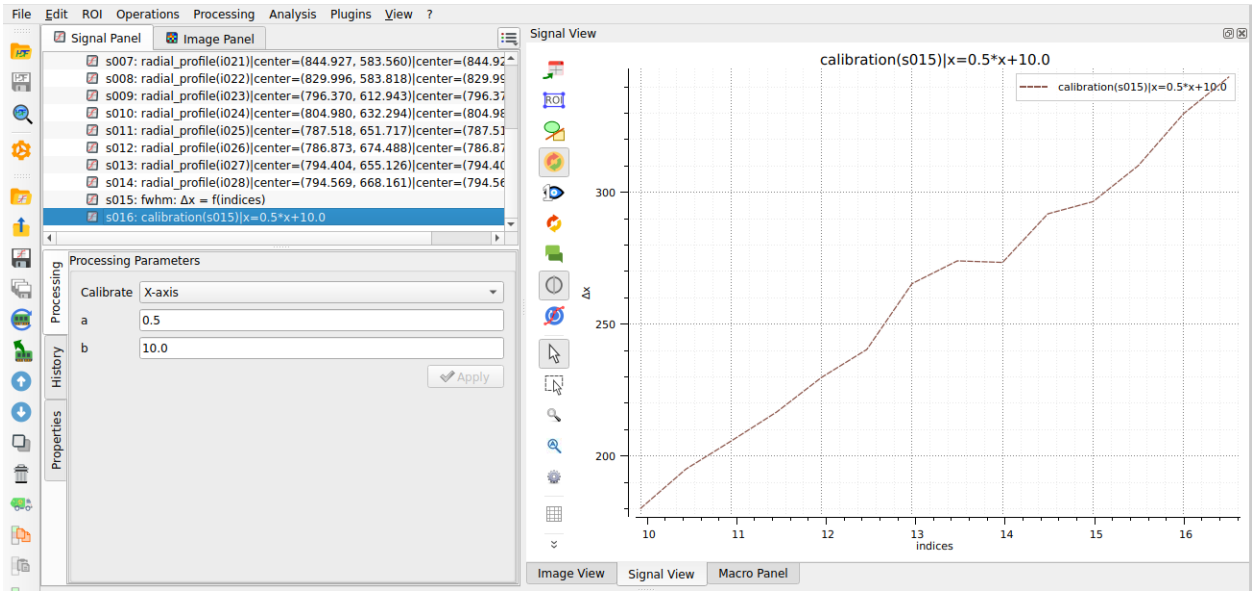


Fig. 2 – Taille du faisceau le long de l'axe de propagation, calculée à partir d'une pile d'images caméra dans DataLab.

2.2.3 Éprouvé en production

DataLab a été créé pour les diagnostics plasma du Laser Mégajoule du CEA – l'une des plus grandes installations laser au monde – où il traite des images et des signaux issus de caméras et de numériseurs ; il est également utilisé pour l'alignement laser et les activités de R&D en métrologie autour de l'installation.

2.2.4 Essayer

— Sans installation : ouvrez DataLab-Web avec des images de faisceau et d'interférogramme de démonstration déjà chargées dans votre navigateur – vos données ne quittent jamais votre machine.
— Tutoriels pas à pas : Profilométrie de faisceau laser et Analyse d'interférogrammes Fabry-Perot.
— Ou installez l'application de bureau.

2.3 DataLab pour le contrôle non destructif

2.3.1 Le problème

Le contrôle produit des images – radiographies, thermogrammes, C-scans ultrasonores, photographies de surface – sur lesquelles il faut détecter, compter et mesurer des défauts ou des indications, souvent sur des séries entières de pièces. La revue manuelle ne passe pas à l'échelle et est difficile à rendre reproductible ; un logiciel entièrement sur mesure est coûteux à développer et à maintenir pour chaque banc de contrôle.

2.3.2 Ce que fait DataLab

DataLab fournit les briques d'analyse d'images et garde toute la chaîne traçable :
— débruiter les images de contrôle (filtre médian, binning et autres filtres),
— détecter automatiquement les éléments de type tache – défauts, particules, points – avec plusieurs algorithmes de détection,
— restreindre l'analyse à des régions d'intérêt et obtenir des résultats mesurables (positions, tailles) exportables vers vos rapports,
— enregistrer l'espace de travail (images + historique de traitement + résultats) dans un unique fichier HDF5, pour la traçabilité,

— et quand les algorithmes intégrés ne suffisent pas, brancher vos propres traitements Python grâce au système de plugins – vos opérateurs gagnent une nouvelle entrée de menu, pas un nouveau logiciel.

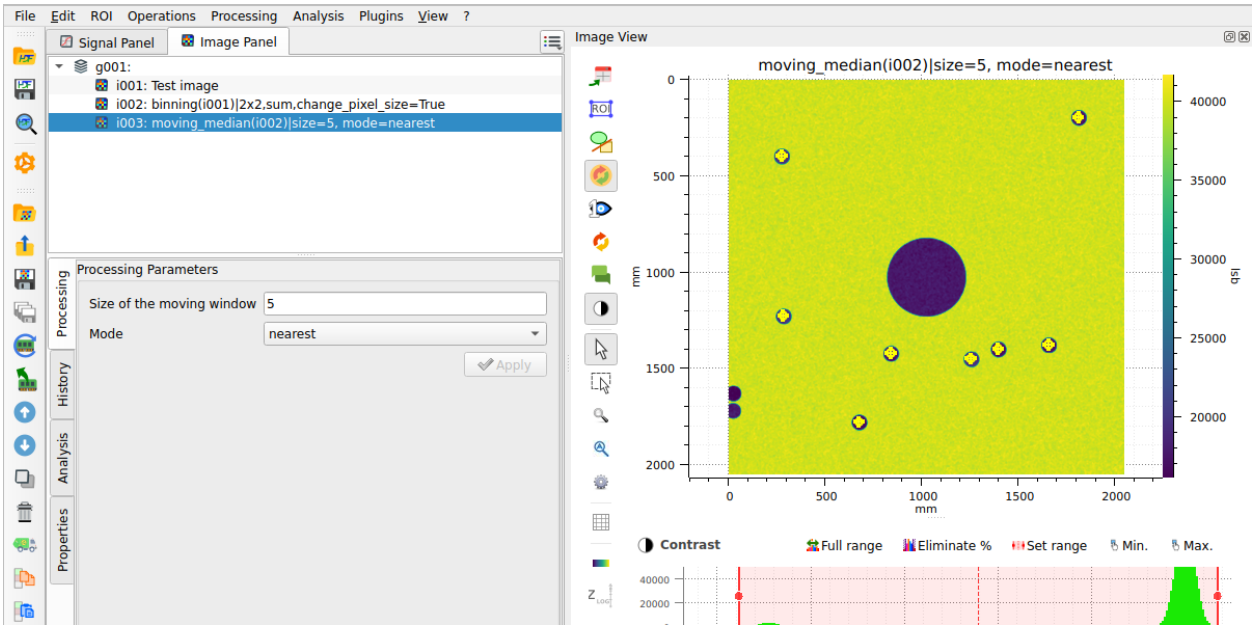


Fig. 3 – Détection automatique de taches sur une image de test débruitée dans DataLab.

2.3.3 Éprouvé en production

Dans le domaine du contrôle non destructif, le CEA a confié à CODRA X-GRID, un logiciel de reconstruction automatique de scènes radiographiques à partir d'images de radiographie X partielles – sans aucune métadonnée préalable de position, d'orientation ou de grandissement. Sa chaîne de traitement (débruitage, détection robuste de taches, estimation d'homographie, fusion d'images) a été prototypée de manière interactive avec DataLab avant d'être intégrée dans l'outil de production. Ces travaux ont été présentés à EuroSciPy 2026.

2.3.4 Essayer

— Sans installation : ouvrez DataLab-Web avec une image d'inspection de démonstration déjà chargée dans votre navigateur – vos données ne quittent jamais votre machine.
— Tutoriel pas à pas : Détection automatisée de défauts et de particules sur des images.
— Ou installez l'application de bureau.

CHAPITRE 3

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.

3.1 Vue d'ensemble et fonctionnalités communes

3.1.1 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 contient les onglets de données Signaux et Images, qui répertorient les jeux de données actuellement chargés dans chaque panneau. Cliquer sur un onglet fait basculer 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.

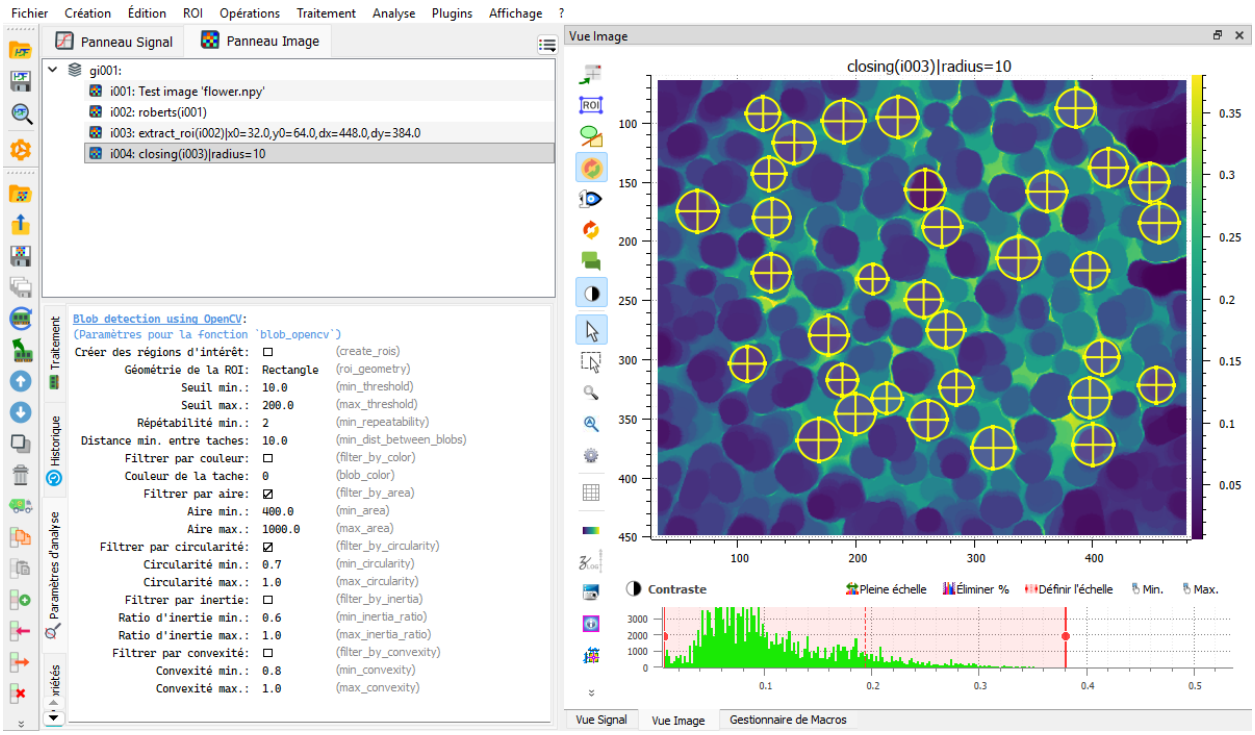


Fig. 1 – Fenêtre principale de DataLab

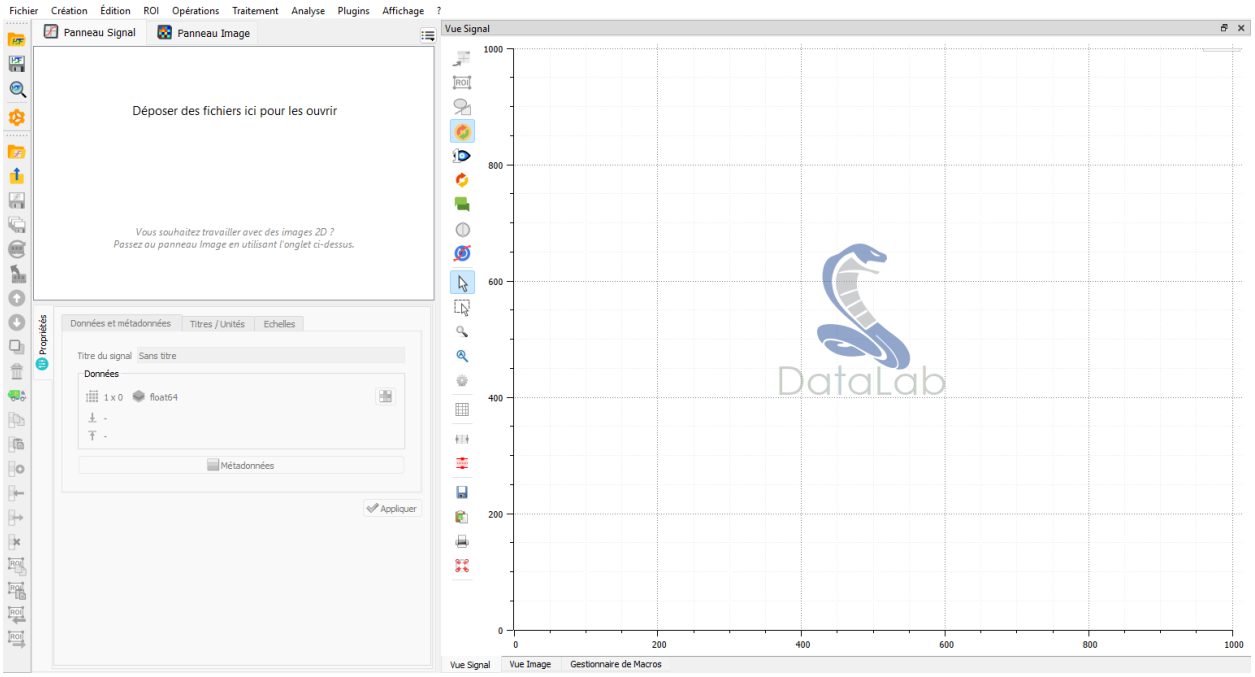


Fig. 2 – Fenêtre principale de DataLab, au démarrage.

Dans la liste des jeux de données, le signal, l'image ou le groupe situé sous le curseur de la souris est mis en évidence, et un bouton de suppression  apparaît à sa droite. Un clic sur ce bouton supprime l'élément, ou toute la sélection si l'élément en fait partie, après confirmation. Supprimer un élément non sélectionné laisse la sélection courante inchangée. Comme l'action « Supprimer » du menu « Édition », la suppression est enregistrée dans le Panneau d'historique. Le bouton peut être masqué dans les Préférences (onglet « Visualisation », sous-onglet « Commun »).

D'autres panneaux sont disponibles sous forme de widgets ancrables, et peuvent être affichés ou masqués depuis le menu « Affichage » : la Page d'accueil, le Panneau Applications, le Panneau d'historique, le panneau de macros (voir Macros) et l'"Assistant IA.

Page d'accueil

Une page d'"Accueil est affichée à côté des vues signal et image au démarrage de DataLab, et chaque fois que le panneau signal ou image courant est vide (par exemple après le passage à un panneau vide ou la suppression de son dernier objet). Elle rassemble les principales actions pour bien démarrer :

- créer un signal ou une image à partir d'un modèle, ouvrir des fichiers de signaux ou d'images, parcourir un fichier HDF5, ouvrir un espace de travail HDF5 précédemment enregistré, ou importer des données texte (CSV, TSV, …) avec l'assistant d'importation. Si nécessaire, un menu permet de choisir entre signaux et images ;
- interroger l'"Assistant IA, suivre la visite guidée, lancer la démo, lire la documentation en ligne, ou consulter les notes de version de la version en cours d'exécution.

Lorsque des plugins d'application sont installés, une section Applications en haut de la page affiche leurs tuiles : un clic sur la tuile principale d'une application affiche sa page dans le Panneau Applications, ou lance l'action principale fournie par le plugin. Une application peut proposer d'autres tuiles, par exemple pour ouvrir directement un exemple : elles sont affichées à côté de sa tuile principale lorsque la section a la place nécessaire, et passent sinon dans le menu de la tuile principale. Ce menu, ouvert depuis le bouton « … » de la tuile principale ou par un clic droit sur celle-ci, permet aussi d'épingler l'application en tête de la section ou de la masquer de la page d'accueil. Les applications épinglées viennent en premier, puis celles utilisées récemment, puis les autres par ordre alphabétique. La section est limitée par défaut à deux lignes de tuiles (ce nombre peut être modifié dans les Préférences) : lorsque davantage d'applications sont installées, la dernière tuile indique le nombre d'applications non affichées et affiche le panneau Applications. Le bouton « Parcourir toutes les applications… » affiche également ce panneau, où le menu « … » de chaque application propose les mêmes options « Afficher sur la page d'accueil » et « Épingler sur la page d'accueil » (voir Plugins).

La page d'accueil est un panneau ancrable comme les autres : dès qu'un signal ou une image est créé ou ouvert, la vue correspondante passe au premier plan, et la page d'accueil reste disponible sous forme d'onglet. Elle peut être fermée comme n'importe quel autre panneau, et rouverte à tout moment depuis l'entrée « Page d'accueil » du menu « ? » (Aide) ou depuis le menu « Affichage » ; elle est aussi rouverte automatiquement lorsque le panneau courant devient vide. La case à cocher « Afficher la page d'accueil lorsque le panneau courant est vide », en bas de la page, détermine cet affichage automatique, y compris au démarrage (cette option est également disponible dans les Préférences).

Panneau Applications

Le panneau Applications présente les méthodes des plugins d'application installés. Il est affiché par l'entrée Applications de la barre de menus, par les tuiles d'applications de la Page d'accueil, ou depuis le menu « Affichage ». La première fois, il s'affiche en fenêtre flottante à côté de la fenêtre principale, sur toute la hauteur de celle-ci, afin que vous gardiez vos données et leurs vues sous les yeux pendant que vous l'utilisez. Vous pouvez ensuite l'ancrer comme n'importe quel autre panneau, par exemple à gauche de la fenêtre principale, à côté de la liste des données dont il analyse la sélection, le regrouper en onglets avec un autre panneau, ou le laisser flottant : DataLab l'affiche à nouveau là où vous l'avez laissé, avec la même taille. Lorsque le panneau est demandé alors qu'il est déjà affiché, sans changer d'application, il clignote brièvement pour indiquer où il se trouve.

Lorsque plusieurs plugins d'application sont installés, leur nombre est indiqué en haut du panneau, au-dessus d'une liste permettant de choisir l'application : cliquez dessus pour afficher toutes les applications, avec la version et la description de chacune. S'il n'y a qu'une application, son nom est affiché à la place. Le bouton « … » situé à côté ouvre un menu donnant accès à la documentation de l'application et à ses options de page d'accueil : « Afficher sur la page d'accueil » et « Épingler sur la page d'accueil ».

Les méthodes de l'application sont listées les unes sous les autres, suivies le cas échéant de ses outils et de ses jeux de données : un clic sur le titre de l'une de ces sections masque son contenu, ou l'affiche à nouveau, et DataLab mémorise ce choix pour toutes les applications. Chaque méthode occupe une ligne. Sa pastille de couleur et le texte placé sous son titre indiquent si la sélection courante peut être analysée ; l'infobulle en donne les raisons. Lorsque des métadonnées manquent, le statut renvoie vers « Édition > Métadonnées > Ajouter des métadonnées… », et il est mis à jour dès que des objets sont modifiés. « Exécuter… » lance la méthode sur la sélection courante, et ne vous demande d'affecter les objets sélectionnés à ses entrées qu'en cas de besoin ; le nombre d'objets créés et les diagnostics de l'exécution s'affichent ensuite sous la ligne.

Sous la liste, dans la zone « Détails », le panneau donne les détails de la méthode sélectionnée : cliquez sur une ligne, ou utilisez les touches `Haut` et `Bas`, pour sélectionner une autre méthode. Les détails indiquent la version et la description de la méthode, si la sélection courante peut être analysée et pourquoi dans le cas contraire, les entrées qu'elle attend (avec leur description, ainsi que la description et la clé des métadonnées requises), et les exemples conçus pour elle : « Essayer avec cet exemple » ouvre un exemple et exécute la méthode avec des paramètres adaptés. Le séparateur entre la liste et les détails règle leurs hauteurs ; DataLab le mémorise, ainsi que la méthode sélectionnée pour chaque application.

Palette de commandes

Plutôt que de naviguer dans des menus imbriqués, n'importe quelle commande peut être atteinte en saisissant une partie de son nom ou de son chemin de menu dans la palette de commandes. Celle-ci répertorie toutes les commandes disponibles pour le panneau actif, identifiées par leur chemin de menu localisé (par exemple « Traitement › Analyse de Fourier › FFT »).

La palette peut être ouverte de trois façons :

- avec le raccourci clavier `Ctrl+Maj+P`,
- en cliquant sur la zone de recherche située dans le coin supérieur droit de la barre de menus,
- depuis l'entrée « Palette de commandes… » en haut du menu « ? » (Aide).

Saisissez quelques caractères pour filtrer la liste : la correspondance est approximative, si bien que « fft » comme « fan » atteignent les entrées « Analyse de Fourier ». Les résultats se parcourent avec `Haut` / `Bas` et se déclenchent avec `Entrée`, exactement comme si la commande avait été sélectionnée dans la barre de menus. Avec une requête vide, toutes les commandes sont listées par ordre alphabétique de chemin.

Panneau d'historique

Le Panneau d'historique est un panneau ancrable supplémentaire qui enregistre les actions sur les signaux et les images dans une session active partagée. Il permet de rejouer les actions, de les rejouer pas à pas, de les dupliquer et de diagnostiquer les incompatibilités. Les sessions peuvent être enregistrées dans des fichiers autonomes `.dlhist` ou avec l'espace de travail au format HDF5.

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 objet signal ou image, DataLab stocke non seulement les données elles-mêmes, mais aussi un ensemble de métadonnées décrivant les données et la manière dont elles doivent être affichées. Ces métadonnées sont stockées dans un dictionnaire accessible par l'attribut `metadata` de l'objet (et peuvent également être consultées dans la vue Propriétés, à l'aide du 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 interactive d'objets

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 objet signal ou image depuis le menu Création
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

Aperçu avant l'application d'un traitement

Les boîtes de dialogue standard des traitements 1-vers-1 proposent une case Aperçu, décochée à chaque ouverture. Cochez-la pour afficher un résultat temporaire pendant le réglage des paramètres. Si plusieurs signaux, images ou groupes sont sélectionnés, choisissez une source dans le sélecteur d'aperçu ; OK traite toujours toute la sélection initiale avec les paramètres communs.

L'aperçu utilise toutes les données sources, y compris leurs régions d'intérêt, sans ajouter d'objet ni d'entrée d'historique. Une saisie invalide suspend les mises à jour. Les avertissements et erreurs s'affichent dans la boîte de dialogue. Les saisies rapprochées sont regroupées et la cadence des curseurs est limitée ; les données volumineuses et les algorithmes coûteux

peuvent prendre davantage de temps. L'aperçu affiche uniquement le résultat, avec ses propres axes et unités, y compris lorsqu'un traitement d'image produit un signal.

Le premier aperçu démarre un processus dédié. Une fois l'aperçu terminé, DataLab garde ce processus prêt pour les boîtes de dialogue d'aperçu suivantes, y compris lors du passage des traitements de signaux aux traitements d'images. Fermer une boîte de dialogue pendant qu'un aperçu est encore en cours arrête ce processus ; l'aperçu suivant effectue alors un nouveau démarrage. Le rechargement des plugins abandonne également le processus d'aperçu mis en cache.

Annuler abandonne les paramètres en cours d'édition et arrête l'aperçu. Avec une seule source, OK réutilise une fois l'aperçu terminé et à jour, puis le publie par le chemin de traitement normal, historique compris. Si l'aperçu est encore en cours ou obsolète, ou si plusieurs objets ou groupes sont sélectionnés, DataLab exécute le traitement normal. Les analyses, les opérations à plusieurs entrées, les opérations sans paramètres et les éditeurs personnalisés non pris en charge conservent leur fonctionnement habituel.

Les champs numériques bornés proposent automatiquement des curseurs dans les formulaires de traitement compatibles et dans l'onglet Traitement. La valeur exacte du champ texte fait foi ; le curseur ne limite pas la précision des valeurs saisies au clavier. Les champs sans deux bornes utilisables restent textuels. Les curseurs n'activent jamais à eux seuls l'aperçu ou le retraitement automatique.

Retraiter un résultat existant

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 retraitement :

1. Appliquer une opération de traitement avec des paramètres (par exemple, Traitement > Réduction de bruit > 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 du sigma du filtre)
6. Cliquer sur Appliquer pour retraiter avec les paramètres mis à jour

Normalement, Appliquer crée un nouveau résultat et une entrée d'historique. En mode édition du panneau Historique, le résultat existant est mis à jour et les actions dépendantes sont recalculées. La case facultative Retraiter automatiquement à la modification suit les mêmes règles, attend le relâchement du curseur et une saisie valide, et n'applique pas les modifications d'un objet précédemment sélectionné. Ce fonctionnement convient notamment pour :

— Ajuster de manière itérative les paramètres du filtre tout en observant les résultats en temps réel
— Ajustement des valeurs de seuil sans créer plusieurs objets intermédiaires
— Tester différentes tailles d'éléments structurants morphologiques
— Démonstrations éducatives des effets des paramètres sur les résultats du traitement

> **Note**
>
> Exigences en matière de métadonnées de traitement :
> — Les traitements avec paramètres peuvent être modifiés ; les traitements sans paramètres (par exemple valeur absolue, moyenne) proposent un bouton Réappliquer
> — Chaque objet d'entrée doit encore exister pour le retraitement (sinon, l'onglet Traitement indique pourquoi il ne peut pas être appliqué)
>
> Lorsque l'espace de travail contient un enregistrement de provenance pour l'objet, l'onglet Traitement restaure l'appel enregistré, y compris pour les traitements 2 vers 1 (source et opérande) et n vers 1 (chacun des objets agrégés, dans l'ordre). Voir Provenance des résultats.

Exemple de flux de travail

Voici un flux de travail typique utilisant le traitement interactif, avec le mode édition de l'historique activé :

1. Créer un signal de test : Création > Signal gaussien
   — Paramètres initiaux : amplitude=1.0, mu=50, sigma=10
2. Ajuster les paramètres de création : Dans l'onglet Création, faire passer sigma à 20, cliquer sur Appliquer
   — Le signal est régénéré avec une nouvelle largeur
3. Appliquer le filtre gaussien : Traitement > Réduction de bruit > Filtre gaussien
   — Sigma initial=2.0
4. Ajuster le filtrage : Dans l'onglet Traitement, essayer sigma=1.0, puis sigma=5.0
   — Chaque clic sur Appliquer 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.

3.1.2 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 six onglets : Général, Traitement, Visualisation, Entrée/sortie, Assistant IA et Console.

Général

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

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

Afficher la page d'accueil lorsque le panneau courant est vide
: Lorsque cette option est activée (par défaut), la page d'accueil est affichée au démarrage de DataLab et chaque fois que le panneau signal ou image courant est vide (voir Page d'accueil).

Lignes d'applications de la page d'accueil
: Nombre maximal de lignes de tuiles d'applications sur la page d'accueil, de 1 à 10 (par défaut : 2). Les autres applications restent accessibles depuis le Panneau Applications.

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.

Contournement localhost de l'API Web
: Lorsque cette option est activée (par défaut), les connexions depuis localhost (127.0.0.1) vers l'API Web ne nécessitent pas d'authentification. Cela simplifie l'intégration des notebooks lorsque DataLab-Kernel s'exécute sur la même machine. Désactiver pour une sécurité plus stricte si nécessaire.

> **Note**
>
> Lorsque cette option est activée, toute application s'exécutant sur votre machine locale peut accéder à l'API Web DataLab sans jeton. Désactivez cette option si vous avez besoin d'une sécurité plus stricte.

Seuil de mémoire
: Définir un seuil (en Mo) en dessous duquel un avertissement est affiché avant de charger de nouvelles données.

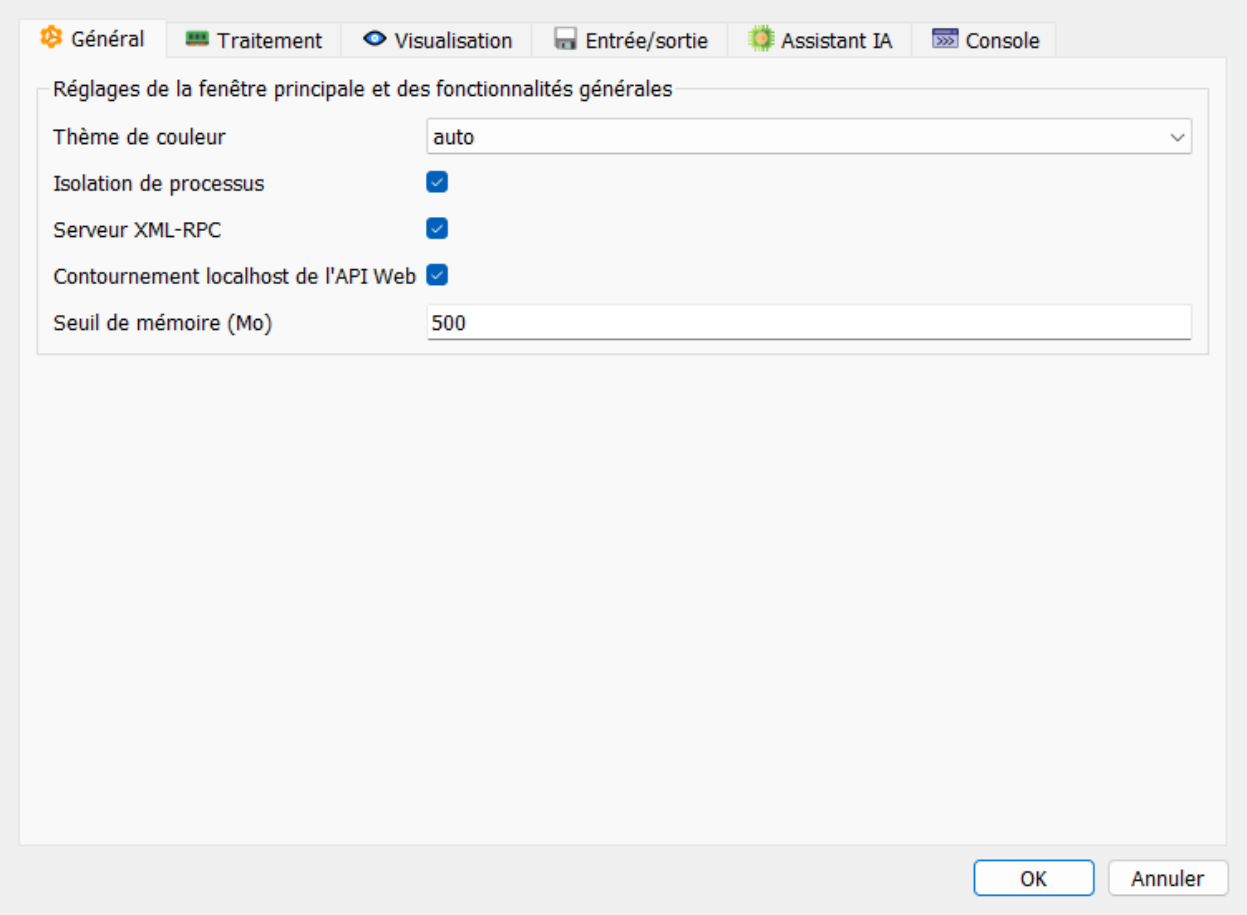


Fig. 3 – Onglet des préférences générales

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.

> **Note**
>
> Paramètres liés aux plugins (activation globale des plugins tiers, activation par plugin et répertoires de plugins supplémentaires) ne font pas partie de cette boîte de dialogue : ils sont regroupés dans la boîte de dialogue de configuration des plugins, accessible depuis l'action de menu `Plugins > Configurer les plugins...` (voir Plugins).

Traitement

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

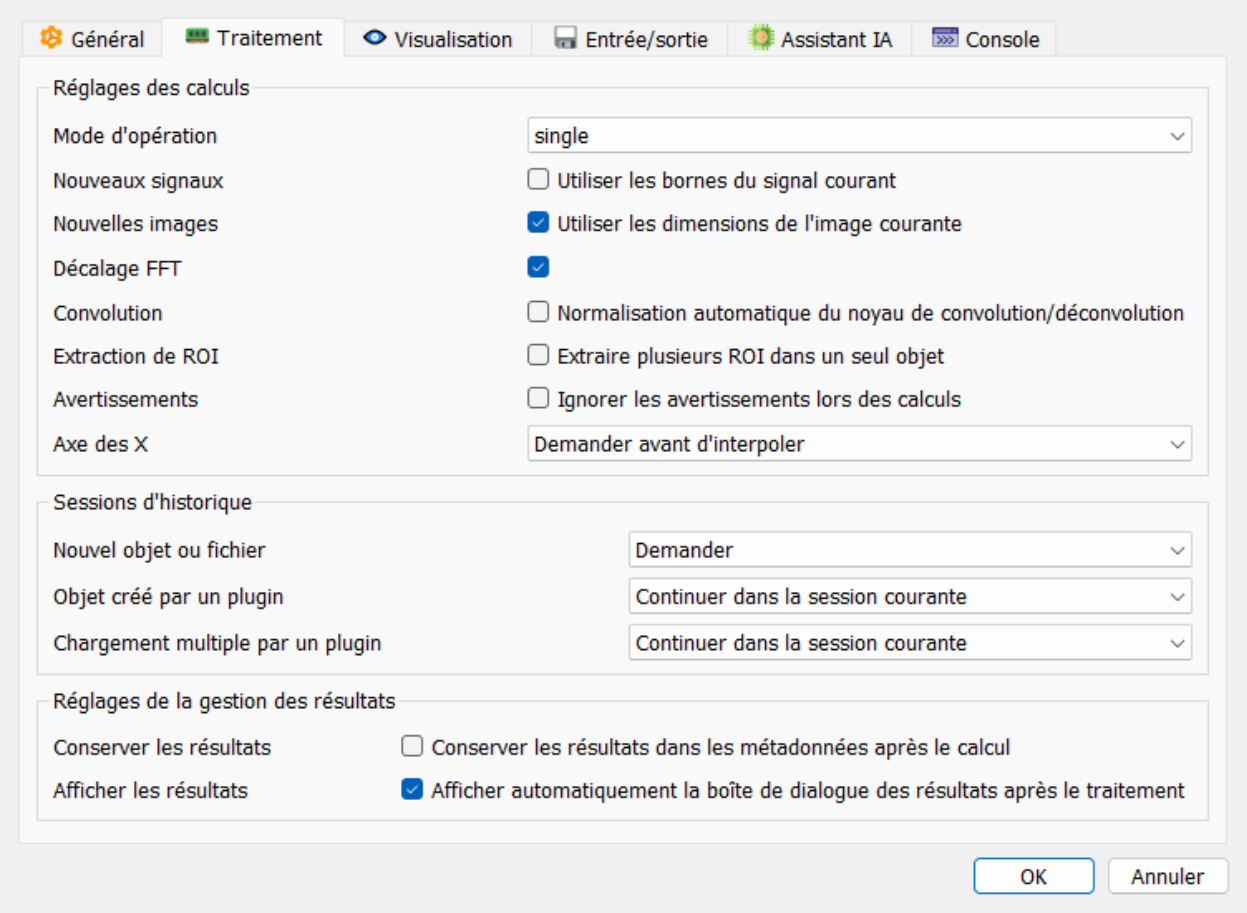


Fig. 4 – Onglet des préférences de traitement

Mode d'opération
: Choisir le mode d'opération pour les calculs prenant N entrées :
— Single : mode d'opération avec un seul opérande
— Pairwise : mode d'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 d'opération avec un seul opérande (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 : Σ(A,B) et Σ(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 d'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

Références dans les titres de résultats
: Choisir comment les objets sources sont représentés dans les titres générés pour les résultats des calculs :
— UUID source (par défaut) : utiliser des références stables de 8 caractères dérivées de l'UUID de chaque objet source.
— Titre de la source : utiliser les titres actuels des objets sources, afin que les références suivent les renommages ultérieurs. Les références imbriquées dans les chaînes de traitement sont également résolues, par exemple `derivative(derivative(Mon signal))`.

Ce réglage modifie uniquement la référence affichée, dans les arborescences d'objets, les légendes et titres des graphiques et les étiquettes ajoutées avec Ajouter le titre de l'objet au graphique. Le titre modifié dans le panneau Propriétés conserve ses références enregistrées. Les titres de résultats stockent des références de 8 caractères (préfixées par `g` pour les groupes), tandis que l'identité de chaque objet reste son UUID complet. Les signaux, images et groupes des arborescences affichent leur propre référence `#UUID8` (ou `#gUUID8`) sous leur titre.

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
: Lorsque cette option est activée, plusieurs régions d'intérêt (ROI) sont extraites dans un seul objet. Lorsqu'elle est désactivée, chaque ROI est extraite dans un objet distinct.

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

Comportement de compatibilité des tableaux des X

Choisir le comportement lorsque les tableaux des X sont incompatibles dans les calculs multi-signaux :
— Demander : afficher une boîte de dialogue de confirmation (par défaut)
— Interpoler : interpoler automatiquement les signaux

Sessions d'historique

Ces paramètres déterminent comment les nouvelles entrées sont affectées aux sessions d'historique. Ils ne sont évalués que lorsque le Mode d'enregistrement est activé et que la session active partagée contient des actions. Aucune décision de politique n'est nécessaire lorsque la session active est vide ; voir Panneau d'historique pour le flux de travail complet.

Nouvel objet ou fichier
: Choisir ce qui se produit lorsqu'un nouvel objet est créé ou qu'un fichier est chargé :
— Demander (par défaut) : demander s'il faut démarrer une nouvelle session
— Toujours démarrer une nouvelle session : démarrer une session avant d'enregistrer l'entrée
— Continuer dans la session courante : ajouter l'entrée à la session active

Objet créé par un plugin
: Choisir ce qui se produit lorsqu'un plugin ajoute un objet :
— Demander : demander s'il faut démarrer une nouvelle session
— Toujours démarrer une nouvelle session : démarrer une session avant d'enregistrer l'objet
— Continuer dans la session courante (par défaut) : ajouter l'objet sans afficher de boîte de dialogue modale, afin de ne pas bloquer l'exécution du plugin

Chargement multiple par un plugin
: Choisir ce qui se produit lorsqu'un plugin regroupe explicitement plusieurs ajouts d'objets dans une même portée de chargement multiple :
— Demander une seule fois : demander une seule fois si l'ensemble du lot doit démarrer une nouvelle session
— Démarrer une nouvelle session : démarrer une seule session pour le lot
— Continuer dans la session courante (par défaut) : ajouter l'ensemble du lot à la session active

Demander une seule fois est le libellé de l'interface correspondant à une décision de session unique et durable qui couvre toute la portée explicite de chargement multiple du plugin, plutôt qu'à une demande par objet. Avec le comportement Demander ordinaire, un mécanisme anti-rebond évite les demandes répétées pour les ajouts synchrones au même panneau pendant le tour courant de la boucle d'événements Qt.

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 :

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).

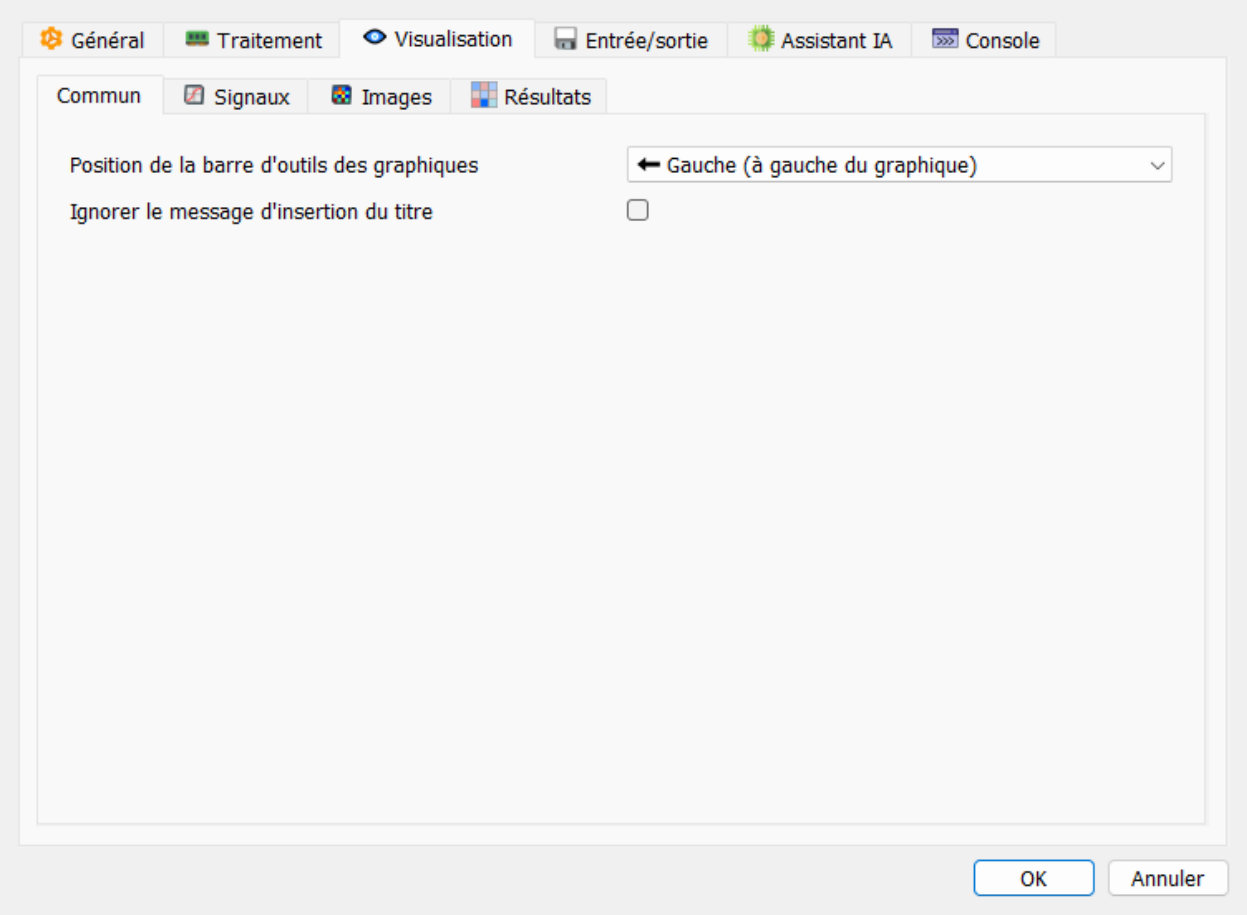


Fig. 5 – Préférences de visualisation - Sous-onglet Commun

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.

Afficher un bouton de suppression au survol des éléments des listes
: Lorsque cette option est activée (par défaut), un bouton de suppression apparaît à droite du signal, de l'image ou du groupe situé sous le curseur de la souris, dans les listes de signaux et d'images (voir Vue d'ensemble). Désactivez cette option pour masquer le bouton : les éléments peuvent toujours être supprimés avec l'action « Supprimer » du menu « Édition », ou avec la touche `Suppr`.

Signaux

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

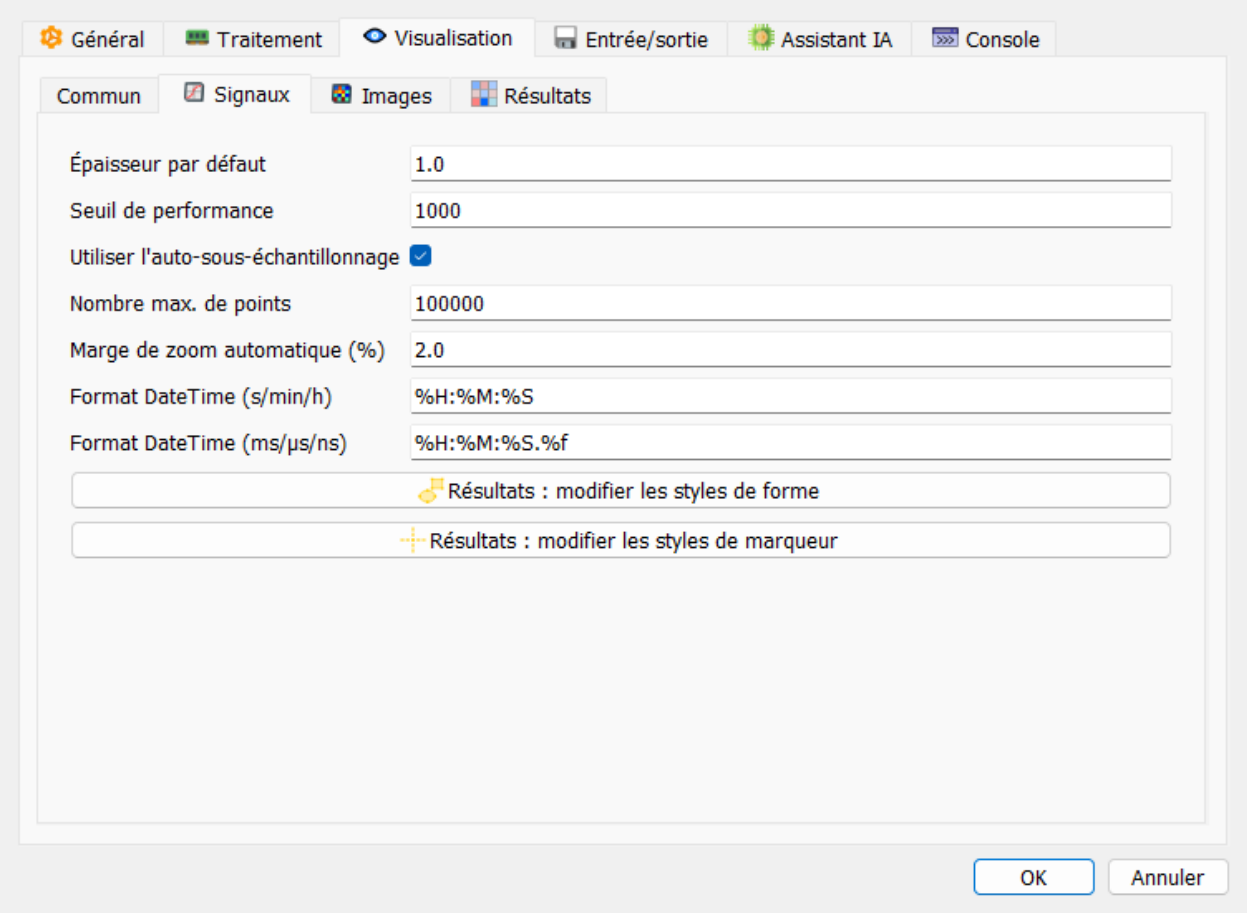


Fig. 6 – Préférences de visualisation - Sous-onglet Signaux

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. Les courbes d'épaisseur 1,0 ou moins sont tracées avec une largeur d'un pixel de l'écran, y compris sur les écrans haute résolution (par exemple avec une mise à l'échelle de l'affichage de 150%), ce qui préserve la rapidité de leur tracé.

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. Le sous-échantillonnage ne concerne que l'affichage : les données des signaux et les traitements ne sont pas modifiés.

Nombre maximum de points de sous-échantillonnage
: Nombre maximal de points affichés pour un signal lorsque le sous-échantillonnage automatique est activé (par défaut : 100 000). Les signaux comportant plus de points sont sous-échantillonnés de sorte qu'au plus ce nombre de points soit affiché.

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/µs/ns)
: Chaîne de format pour les étiquettes de l'axe X datetime lors de l'utilisation d'unités de temps infra-secondes (ms, µs, 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 :

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).

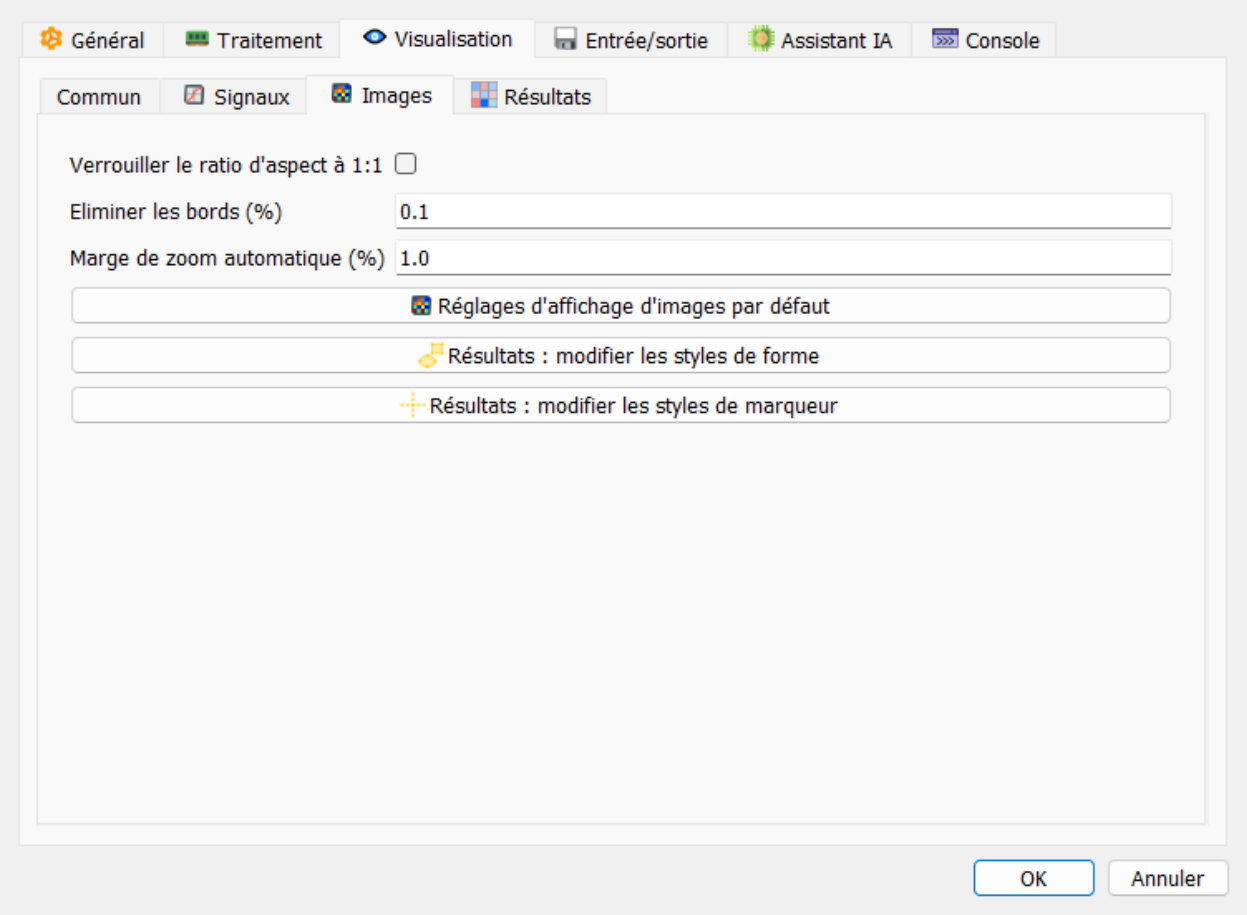


Fig. 7 – Préférences de visualisation - Sous-onglet Images

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 :

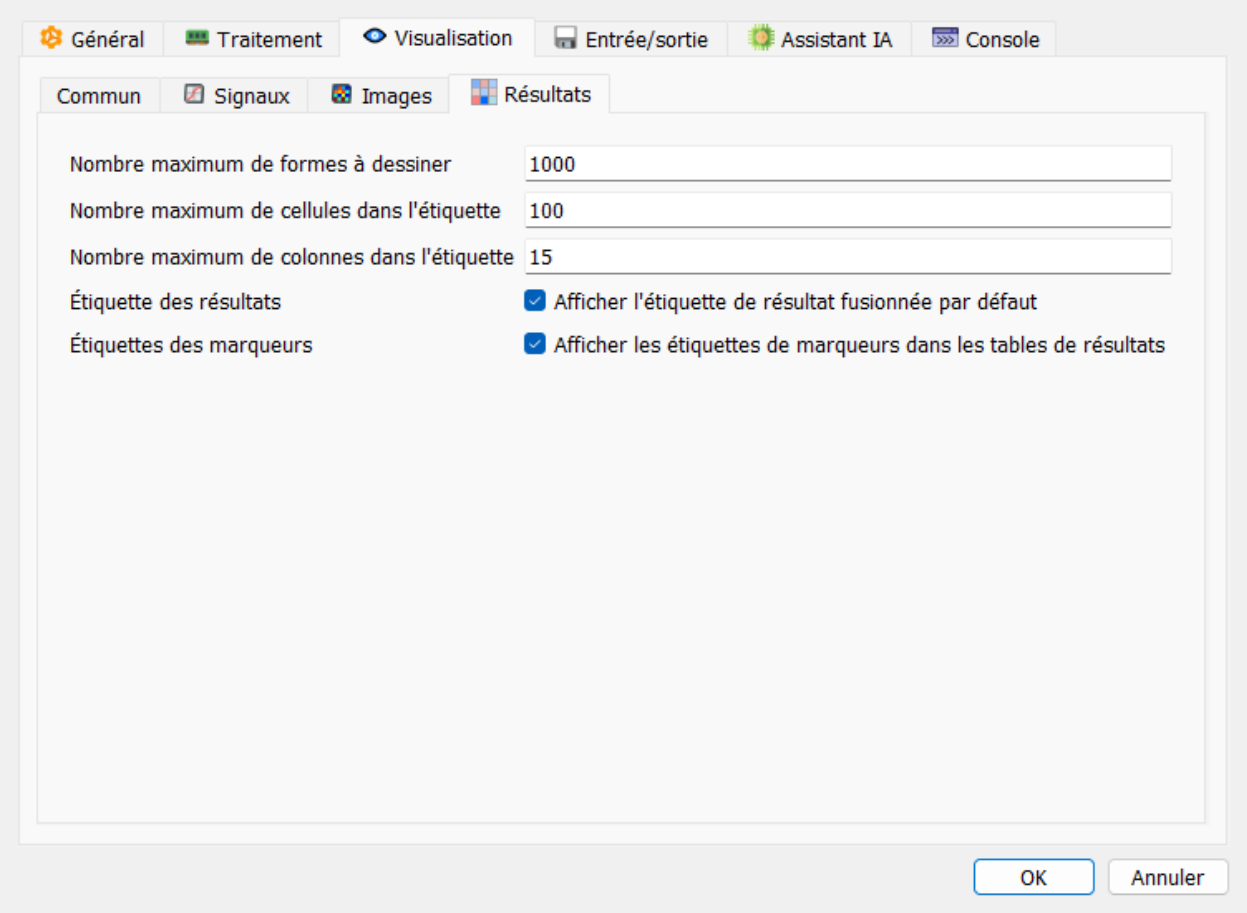


Fig. 8 – Préférences de visualisation - Sous-onglet Résultats

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.

Afficher les étiquettes de marqueurs dans les tableaux de résultats
: Lorsque cette option est activée, une colonne d'étiquettes de marqueurs est ajoutée en tête des tableaux de résultats de type marqueurs (marqueurs XY, X ou Y), afin que chaque ligne puisse être mise en correspondance avec la croix ou le curseur en pointillés correspondant sur le graphique. Les marqueurs XY utilisent des étiquettes numériques (`#1`, `#2`, etc.), les marqueurs d'axe des lettres (`a`, `b`, `c`, etc.).

> **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 :

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.

Assistant IA

L'onglet Assistant IA configure l'assistant conversationnel intégré (voir Assistant IA). Tous les champs sont inactifs tant que l'option Activer l'assistant IA n'est pas cochée.

Activer l'assistant IA
: Active le panneau Assistant IA. Lorsque l'option est désactivée, aucune requête n'est jamais envoyée à un service externe.

Fournisseur
: Service utilisé pour servir le modèle. `openai` couvre tous les points d'accès compatibles OpenAI (OpenAI, GitHub Models, Azure OpenAI, Ollama, LM Studio, llama.cpp, vLLM, etc.). Utilisez `mock` pour essayer l'assistant hors ligne, sans aucune clé d'API : il répond par des réponses scriptées déclenchées par de simples mots-clés.

Modèle
: Nom du modèle à utiliser, tel qu'attendu par le fournisseur (par exemple `gpt-4o-mini`).

Clé d'API
: Clé d'API du fournisseur.

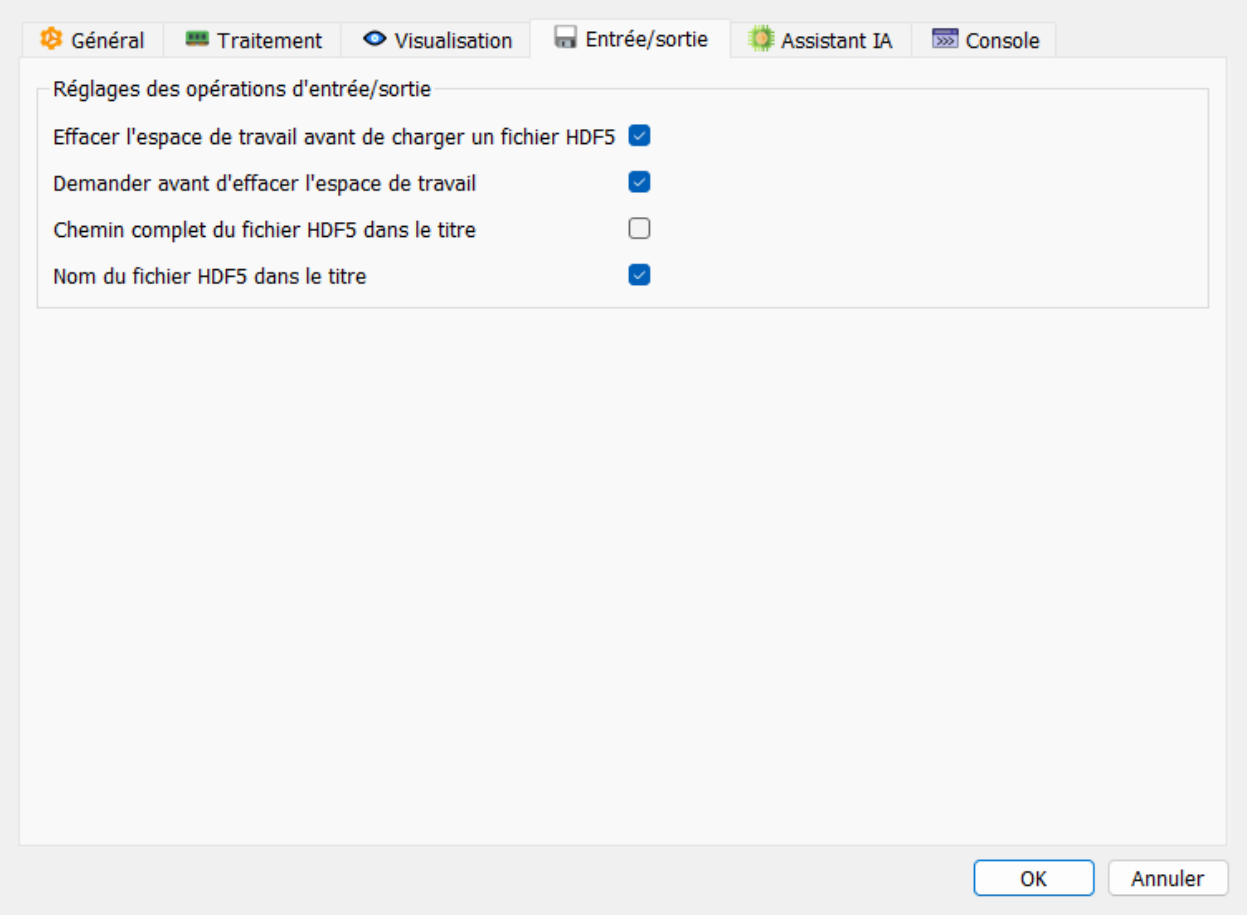


Fig. 9 – Onglet des préférences d'entrée/sortie

> **Avertissement**
>
> Cette valeur est stockée en clair dans le fichier de configuration de DataLab. Il est recommandé de laisser le champ vide et de définir à la place la variable d'environnement standard du fournisseur (par exemple `OPENAI_API_KEY`) : le secret n'est alors jamais écrit sur le disque et peut être partagé avec d'autres outils.

URL de base (optionnel)
: Remplacement du point d'accès pour les services compatibles OpenAI. Laisser vide pour utiliser la valeur par défaut du fournisseur.

Charger un préréglage…
: Pré-remplit l"URL de base et le Modèle avec un point d'accès compatible OpenAI connu (OpenAI cloud, Ollama, LM Studio, llama.cpp, vLLM).

Tester la connexion
: Interroge le point d'accès configuré afin de vérifier l'URL, la clé d'API et l'accessibilité réseau. Cette opération ne consomme aucun jeton.

Température
: Température d'échantillonnage transmise au modèle (0 = déterministe).

Délai d'expiration HTTP (s)
: Durée maximale d'une requête au fournisseur.

Nombre maximum d'itérations d'appels d'outils
: Limite de sécurité sur le nombre d'appels d'outils que l'assistant peut enchaîner pour répondre à une seule question.

Nombre max. de messages d'historique (0 = illimité)
: Limite le nombre de messages passés envoyés au fournisseur à chaque requête. Utile avec les modèles locaux disposant d'une petite fenêtre de contexte, où une conversation trop longue est rejetée par le serveur. L'invite système et le message utilisateur courant sont toujours préservés.

Approuver automatiquement les outils d'inspection en lecture seule
: Lorsque cette option est activée, les outils qui se contentent de lire l'espace de travail (lister les objets, inspecter un objet, lister les opérations disponibles) s'exécutent sans demander de confirmation. Les outils qui modifient l'espace de travail exigent toujours une confirmation explicite.

Autoriser l'IA à créer et exécuter des macros (code Python)
: Lorsque cette option est activée, l'assistant peut proposer des macros Python, exécutées avec un accès complet à DataLab via l'API du proxy — chaque exécution exige néanmoins une confirmation explicite. Désactivez cette option pour masquer complètement l'outil de macro au modèle.

Console

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

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.

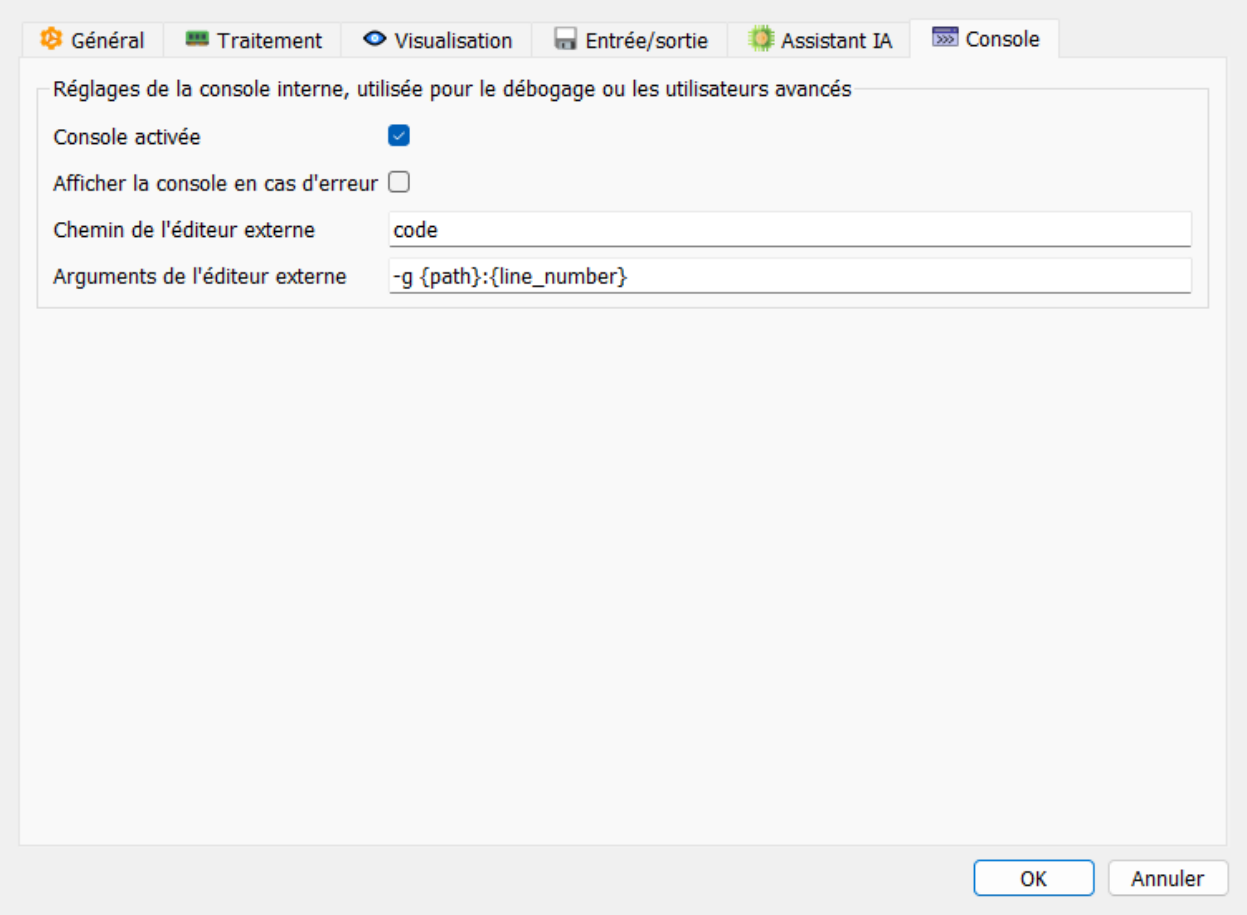
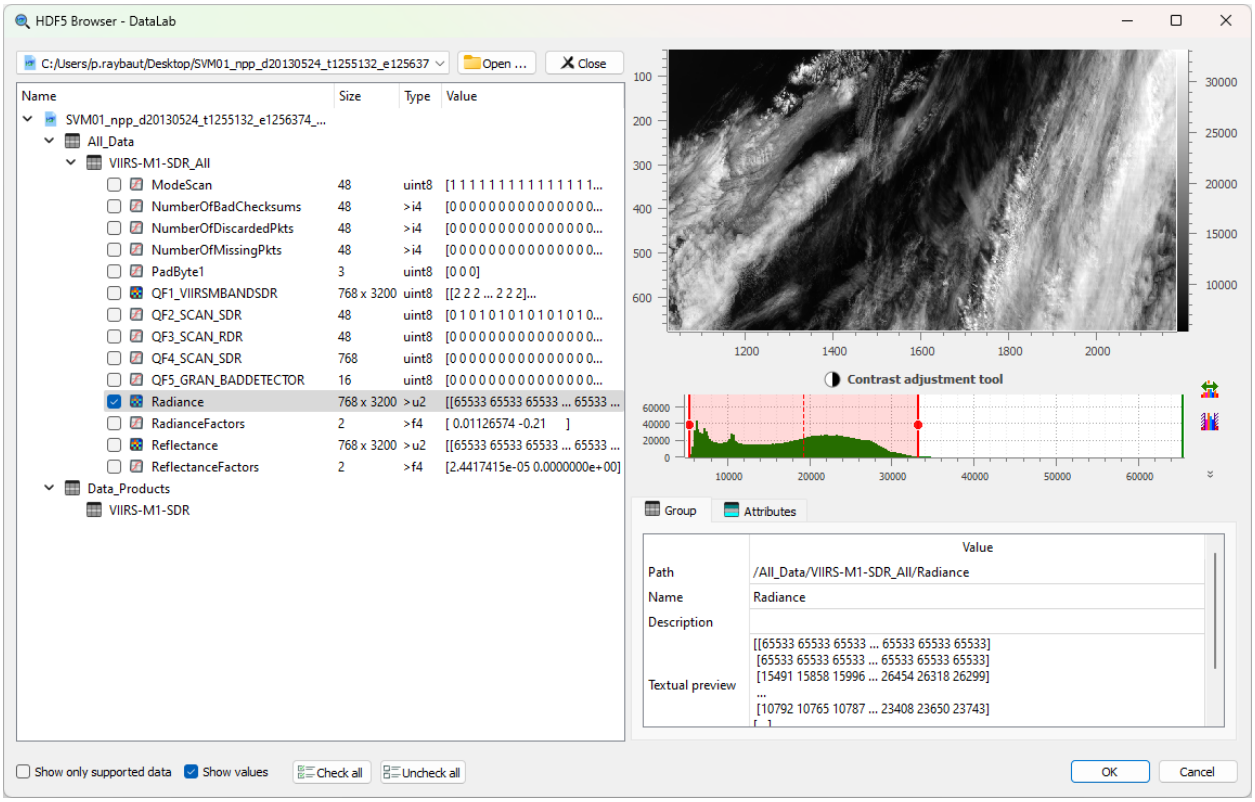


Fig. 10 – Onglet des préférences de console

3.1.3 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.

Fichiers NeXus et axes physiques

Ouvrez les fichiers `.nxs` et `.nx5` avec Parcourir un fichier HDF5, comme les fichiers `.h5`. Un groupe NeXus `NXdata` désigne un jeu de données mesurées et ses axes. Sélectionnez le groupe pour importer l'objet scientifique complet plutôt que ses jeux de données individuels : un groupe 1D produit un signal, et un groupe 2D une image. Les jeux de données bruts restent accessibles séparément dans l'arbre.

L'import conserve les valeurs, libellés et unités des axes, y compris les coordonnées d'image non uniformes. Les incertitudes Y des signaux sont importées lorsqu'elles sont fournies ; les incertitudes des images ne le sont pas. Les axes d'histogramme fournis sous forme de bords de classes sont convertis en centres de classes, et cette conversion est enregistrée.

Un axe dont la forme ou la longueur est invalide, un attribut `axes` qui ne donne pas un axe par dimension ou un axe qui ne contient pas de nombres est signalé dans la description du groupe et empêche son import. Un axe non déclaré utilise les indices des points ou pixels ; l'importeur ne déduit pas d'axe physique à partir des jeux de données voisins. Les images nécessitent des coordonnées croissantes : un axe d'image décroissant est inversé avec l'image, et cette conversion est enregistrée, tandis qu'un axe d'image qui n'est pas strictement monotone empêche l'import.

Les fichiers écrits selon les conventions NeXus anciennes, comme les scans du synchrotron SOLEIL, sont aussi importés avec leurs axes : un jeu de données d'axe peut donner la dimension qu'il décrit avec son attribut `axis`, celui dont l'attribut `primary` est le plus petit étant préféré. Les autres jeux de données mesurés d'un groupe, comme tous les compteurs et détecteurs d'un scan, apparaissent comme des éléments supplémentaires sous le groupe, chacun avec les axes qui correspondent à ses dimensions : un compteur est un signal en fonction du moteur scanné, un détecteur écrivant un spectre à chaque point une image avec le moteur selon Y, et un détecteur écrivant une image à chaque point une pile d'images (voir ci-dessous).

Lorsqu'un fichier NeXus désigne une mesure par défaut (attributs `default`), le navigateur la sélectionne et l'affiche à l'ouverture du fichier. Les données de détecteur compressées, comme les données bitshuffle/LZ4 des détecteurs Eiger, sont lues ; un jeu de données compressé avec un filtre indisponible est signalé dans l'arborescence au lieu de provoquer une erreur.

Les fichiers comportant plusieurs compteurs ou des descriptions NeXus incomplètes peuvent être interprétés par un profil de correspondance fourni par un plugin. Chaque objet décrit apparaît comme un élément supplémentaire avec ses propres axes, libellés et unités ; les axes d'un objet décrit peuvent ne décrire que ses premières dimensions, de sorte qu'un même profil s'applique aux compteurs et aux détecteurs d'un scan. Les profils doivent être enregistrés avant l'ouverture du fichier ; DataLab ne découvre pas automatiquement les fichiers JSON situés à côté. Les auteurs de plugins peuvent suivre le guide d'import HDF5 de SigimaX.

Grands jeux de données et aperçus

La liste des jeux de données numériques utilise leur forme et leur type sans charger toutes leurs valeurs. Les aperçus textuels ne lisent que les premiers éléments, et les attributs contenant plus de 10 000 valeurs ne sont pas copiés dans les métadonnées des objets importés. Les aperçus graphiques sont lus dans un fil d'exécution en arrière-plan ; seul l'aperçu du dernier élément sélectionné est affiché, et les éléments sélectionnés entre-temps ne sont pas lus. La sélection d'un élément qui ne peut pas être prévisualisé, comme un groupe, efface l'aperçu. Les liens rompus, comme les liens externes vers des fichiers de détecteur absents, sont ignorés à l'ouverture du fichier.

Les grands tableaux 1D ordinaires et les images 2D, les objets NeXus et décrits par un profil sont prévisualisés avec un pas d'échantillonnage au-delà de `2**24` éléments (environ 16,8 millions). L'import lit toujours toutes les valeurs, pas l'aperçu échantillonné. Ce n'est pas une limite mémoire universelle : les jeux de données composites et les signaux stockés dans des tableaux 2D utilisent encore des objets complets pour leurs aperçus graphiques. La fermeture d'un fichier ou de l'explorateur attend la fin de la lecture de son aperçu.

Jeux de données multidimensionnels

Les jeux de données à trois dimensions ou plus, comme les piles d'images et les cubes spectraux, sont importés sous la forme d'une image ou d'un signal par trame. La sélection d'un tel élément affiche un sélecteur de vue sous l'aperçu :

- Vue : images de deux axes, ou signaux selon un axe
- Trames selon : l'axe selon lequel les trames sont prises, avec la Première et la Dernière trame et un Pas
- Indices fixés : l'indice des autres axes, le cas échéant
- Importer ces trames : rend l'élément cochable et le coche ; décocher l'élément annule l'import

La première trame de la vue est affichée en aperçu, avec le nombre de trames. Par défaut, ce sont des images des deux derniers axes qui sont proposées, avec toutes les trames selon l'axe précédent, sauf si l'attribut NeXus `interpretation` du jeu de données désigne des spectres (signaux selon le dernier axe) ou des images RVB (une image par canal). DataLab demande une confirmation avant d'importer plus de 100 trames. Les trames des jeux de données NeXus conservent les axes de leurs dimensions affichées ; leur titre donne les indices de la trame et, si les autres axes ont des coordonnées, la position de la trame (par exemple `frames[3, :, :] (theta=0.3)`). Les développeurs peuvent aussi lire des vues sans créer d'objets, avec l'API de sélection décrite dans le guide de SigimaX.

Valeurs numériques et origine expérimentale

Les comptages de détecteur entiers non signés sur 32 bits au-delà de 2 147 483 647 sont importés comme images `float64` sans écrêtage ni arrondi. Les autres tableaux entiers sont aussi convertis vers un type pris en charge exact lorsque c'est possible. Si la conversion en flottant ne peut pas préserver chaque entier (notamment au-delà de `+/-2**53` pour `float64`), Sigima émet un avertissement au lieu de promettre silencieusement des valeurs exactes. Les signaux complexes conservent leur partie imaginaire, y compris en simple précision.

Les objets NeXus et décrits par un profil portent une entrée de métadonnées `experimental_source` décrivant le fichier source, le groupe, le jeu de données mesuré, les axes (`"."` pour une dimension sans axe) et les conversions enregistrées, avec le titre et les dates de début et de fin de l'entrée NeXus et, pour les trames, les indices et la position de la trame. Les objets décrits par un profil identifient également leur profil. Cette entrée décrit l'origine de la mesure ; ce n'est pas la provenance des traitements affichée dans le panneau Propriétés, et elle ne rend pas l'expérience reproductible.

Enregistrez les objets importés dans un espace de travail DataLab pour conserver leurs données et ces métadonnées. La réouverture de cet espace de travail ne nécessite ni le fichier NeXus original ni le plugin du profil. Les imports génériques de jeux de données bruts ne reçoivent pas automatiquement cette description structurée de leur origine expérimentale.

3.1.4 Panneau d'historique

Vue d'ensemble

Le « panneau d'historique » enregistre la séquence des actions effectuées par l'utilisateur sur les signaux et les images, organisée en sessions. Chaque session est une liste chronologique constituée soit :

- d"actions de l'interface (création d'un nouveau signal, suppression des objets sélectionnés, enregistrement de l'espace de travail au format HDF5, …), soit
- de calculs (FFT, moyenne, ajustement gaussien, …) dont l'exécution est confiée à Sigima par les processeurs de DataLab, soit
- de mutations (modifications en place d'objets existants, telles que l'édition de régions d'intérêt). Voir Mutations d'objets.

Une session enregistrée peut être :

- rejouée silencieusement ou pas à pas. Le rejeu recalcule en place chaque action de calcul avec ses paramètres enregistrés : les objets de sortie existants sont mis à jour, et les sorties supprimées dont l'action fait toujours partie de l'historique sont recréées sous leurs identifiants d'origine, ce qui préserve la validité de la chaîne de traitement en aval. En mode pas à pas, les paramètres peuvent être examinés et modifiés avant le recalcul de chaque étape. Les actions de l'interface enregistrées sont invoquées par leurs propres méthodes et peuvent reproduire leurs effets, notamment créer, importer ou dupliquer des objets ;
- dupliquée sous forme de chaînes de traitement indépendantes dans de nouvelles sessions d'historique, les objets signal/image nécessaires étant clonés au cours de l'opération ;

— sauvegardée dans un fichier d'historique autonome (`.dlhist`) ou intégrée à l'espace de travail lors de l'enregistrement au format HDF5, de sorte que toute la chaîne de traitement accompagne les données.

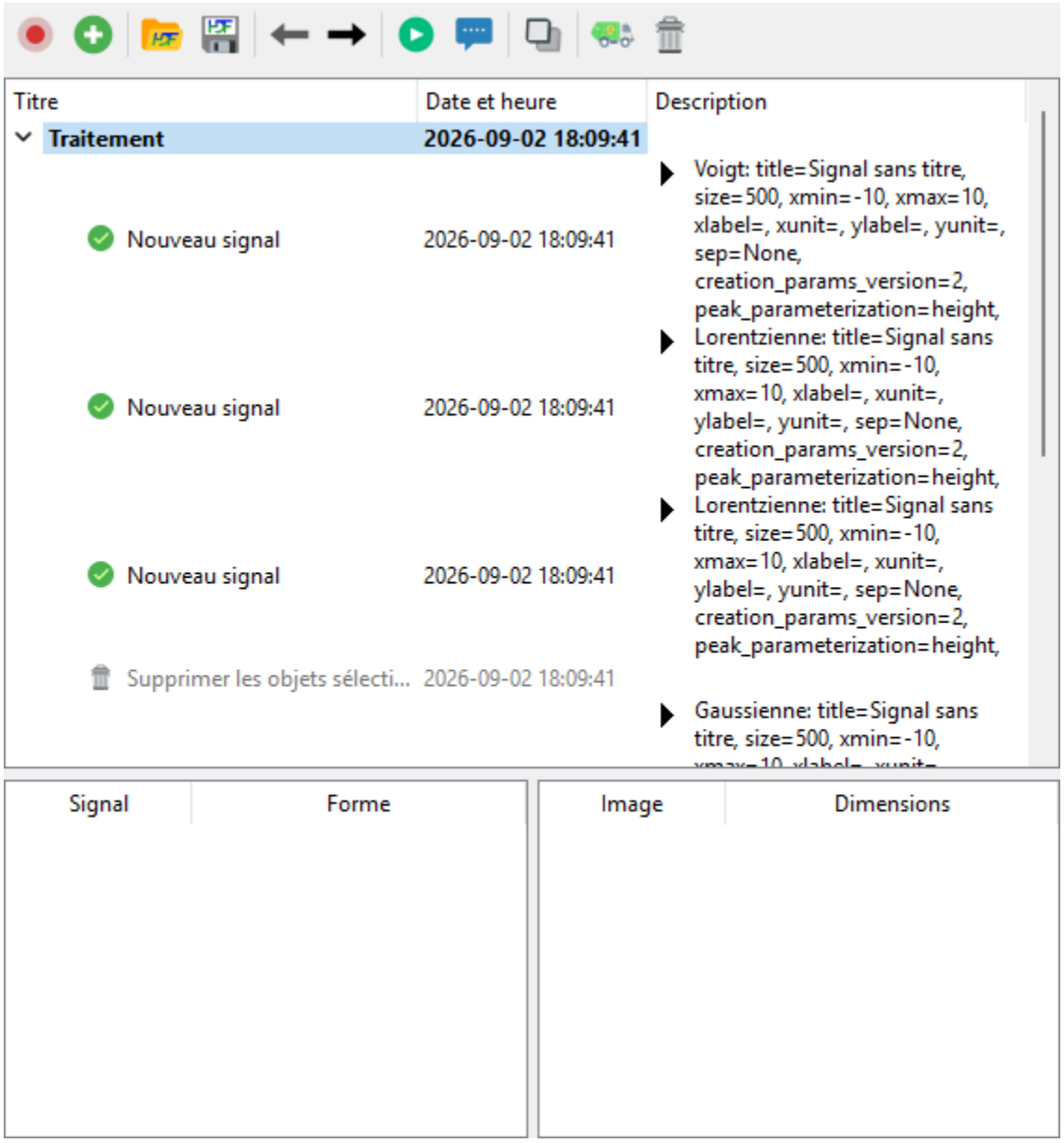


Fig. 11 – Le panneau d'historique après l'enregistrement d'une session représentative : création de trois signaux (Voigt, lorentzien, lorentzien), suppression de l'un d'eux, création d'un signal gaussien, calcul de la moyenne, ajout d'un bruit gaussien au résultat et ajustement par une gaussienne.

Mutations d'objets

Outre les actions de l'interface et les calculs, le panneau enregistre des mutations : des modifications en place d'objets existants qui ne créent pas de nouveaux objets. Les mutations couvrent actuellement les régions d'intérêt (ROI) : la définition ou l'édition de ROI de manière graphique ou numérique, la suppression d'une ou de toutes les ROI et le collage de ROI enregistrent chacun une entrée de mutation générique unique contenant les objets concernés et l'état de ROI résultant.

Le rejeu d'une mutation réapplique l'état de ROI enregistré à ses objets cibles (un état vide supprime les ROI). En mode pas à pas, les paramètres des ROI peuvent être examinés et modifiés dans une boîte de dialogue avant d'être réappliqués ;

leur modification déclenche un recalcul des calculs dépendants en aval.

Lorsqu'une action recalculée recrée ou met à jour un objet, les mutations enregistrées sur cet objet sont réappliquées dans l'ordre, et les analyses dépendant de l'objet muté sont recalculées. Les ROI définies par l'utilisateur sur un objet de sortie sont préservées par les recalculs en place, sauf si le recalcul produit lui-même une ROI.

Les entrées de mutation sont sauvegardées avec les sessions (fichiers `.dlhist` autonomes et espaces de travail HDF5) comme toute autre action ; les fichiers d'historique créés avec des versions antérieures de DataLab se chargent sans modification.

Enregistrement et cycle de vie des sessions

Les actions ne sont enregistrées que lorsque le Mode d'enregistrement est activé. Sa désactivation conserve les sessions existantes, mais n'ajoute aucune nouvelle entrée.

Il existe une unique session d'enregistrement active, partagée par les panneaux Signaux et Images. Les nouvelles actions des deux panneaux sont enchaînées dans cette session (une session est créée à la première utilisation), de sorte que les chaînes mixtes signaux/images sont enregistrées ensemble et que l'enregistrement reprend dans la session sélectionnée par l'utilisateur. Chaque session reste associée au panneau Signaux ou Images pour l'affichage, au travers des actions qu'elle contient.

Lorsqu'un nouvel objet est créé ou qu'un fichier est chargé dans une session active non vide, une politique configurable détermine si DataLab demande quoi faire, démarre une nouvelle session ou continue dans la session courante. Des politiques distinctes s'appliquent aux objets créés par des plugins. Une portée explicite de chargement multiple par un plugin fournit une politique de session unique et durable pour l'ensemble du lot. Avec le comportement Demander ordinaire, un mécanisme anti-rebond évite les demandes répétées pour les ajouts synchrones au même panneau pendant le tour courant de la boucle d'événements Qt.

Ces options sont disponibles sous `Fichier > Préférences > Traitement > Sessions d'historique`. Voir Sessions d'historique pour la liste complète des libellés et des valeurs par défaut.

Barre d'outils

La barre d'outils en haut du panneau expose les actions suivantes :

- Mode d'enregistrement : active ou désactive l'enregistrement des nouvelles actions. Lorsqu'il est désactivé, aucune nouvelle entrée n'est ajoutée à l'historique (les sessions existantes sont conservées).
- Nouvelle session : démarre une nouvelle session d'historique et en fait la session d'enregistrement active.
- Ouvrir un fichier d'historique : charge des sessions enregistrées depuis un fichier `.dlhist` autonome.
- Enregistrer un fichier d'historique : enregistre les sessions actuellement enregistrées dans un fichier `.dlhist` autonome.
- Étape précédente : sélectionne l'action précédente dans la session courante (raccourci clavier : `Ctrl+Gauche`).
- Étape suivante : sélectionne l'action suivante dans la session courante (raccourci clavier : `Ctrl+Droite`).
- Rejouer : recalcule la sélection en place, silencieusement (sans boîtes de dialogue de paramètres). Sélectionner une action rejoue cette action ; sélectionner une session rejoue toutes ses actions. Une sélection couvrant plusieurs actions ou sessions est fusionnée, dédoublonnée et exécutée dans l'ordre des sessions. Chaque action de calcul est réexécutée avec ses paramètres enregistrés et met à jour son ou ses objets de sortie existants, en conservant les mêmes identifiants afin que les étapes en aval restent valides. Les sorties supprimées du panneau de données sont recréées sous leurs identifiants d'origine (flux de travail typique : supprimer un résultat erroné, modifier ses paramètres, puis rejouer pour le régénérer). Les actions dont les objets source n'existent plus sont ignorées avec un avertissement, et une action en échec bloque sa branche en aval. Les actions dont les paramètres ont été modifiés (en mode pas à pas ou depuis l'onglet Traitement) sont marquées comme obsolètes ; le rejeu les

recalcule et, lorsque les paramètres ont été modifiés, recalcule également leurs actions dépendantes en aval. Les actions d'analyse sont rejouées en recalculant leurs résultats sur les objets source : chaque analyse enregistre les résultats qu'elle a stockés sur l'objet (ses effets), de sorte que le rejeu met à jour exactement ces résultats — les valeurs précédentes sont remplacées et, si le recalcul échoue, les résultats précédents sont restaurés. Les analyses enregistrées avec des versions antérieures de DataLab sont rejouées à partir de leur état sauvegardé. Les actions de l'interface sont rejouées en invoquant leur méthode enregistrée et peuvent reproduire des effets, notamment créer, importer ou dupliquer des objets ; les actions de chargement rechargent leurs objets même si ceux-ci ont été supprimés après leur enregistrement. Les actions destructives enregistrées (par exemple la suppression d'objets) sont ignorées avec un avertissement chaque fois que leur rejeu serait risqué : lorsque leur état capturé ne correspond plus à l'espace de travail courant, lorsque leurs cibles ont été recréées plus tôt au cours du même rejeu, ou lorsque leurs cibles appartiennent à une autre session. En conséquence, une session rejouée peut intentionnellement ne pas reproduire un état final enregistré qui comportait de telles suppressions.

— Pas à pas : rejoue la même sélection étape par étape, en ouvrant la boîte de dialogue des paramètres pour chaque action prise en charge (création d'objet, calcul, extraction de ROI) avant de la recalculer. Les modifications acceptées se propagent aux actions dépendantes en aval, qui sont également recalculées. L'annulation d'une boîte de dialogue interrompt le rejeu, restaure les modifications de paramètres effectuées durant cette exécution et recalcule silencieusement les actions restées obsolètes afin que la chaîne reste à jour.

— Afficher la provenance de l'exécution : afficher la provenance enregistrée de la dernière exécution réussie de l'action de calcul sélectionnée (un onglet par résultat), voir Provenance des résultats. Si les paramètres de l'action ont été modifiés sans qu'elle soit exécutée à nouveau, la boîte de dialogue l'indique.

— Dupliquer : duplique la chaîne de traitement contenant chaque action sélectionnée, ou les chaînes de traitement de chaque session sélectionnée. DataLab clone les objets nécessaires et crée des sessions d'historique indépendantes. La copie est une copie profonde autonome de la chaîne — comme si les opérations enregistrées avaient été refaites à la main sur les objets clonés — de sorte que le rejeu de la session d'origine ou de la session dupliquée n'affecte que ses propres objets.

— Supprimer les actions incompatibles : supprime toutes les actions dont l'état de l'espace de travail n'est plus compatible avec l'espace de travail actuel. Une boîte de dialogue de confirmation indique combien d'actions seront supprimées.

— Supprimer : retire de l'historique les actions ou sessions sélectionnées (raccourci clavier : `Suppr`, actif uniquement lorsque l'arborescence de l'historique a le focus, afin de ne jamais supprimer à la place des objets signal ou image). La suppression d'une action intermédiaire la retire de la chaîne et conserve ses étapes en aval sous forme de chaîne indépendante.

> **Note**
>
> Un double-clic sur un élément de l'arborescence déclenche Rejouer pour la sélection courante, avec la même sémantique de recalcul en place que celle documentée ci-dessus.

> **Note**
>
> Lorsqu'un rejeu ou un recalcul est en cours, les commandes du panneau Historique sont indisponibles : les actions de la barre d'outils sont désactivées et les interactions avec l'arborescence (menu contextuel, double-clic, touche `Suppr`) sont ignorées jusqu'à la fin de l'exécution.

Arborescence

L'arborescence organise les actions enregistrées dans des sessions dépliables :

- Chaque ligne de premier niveau est une session ; le ou les panneaux auxquels elle se rapporte sont déterminés par les actions qu'elle contient. Les sessions peuvent être démarrées lorsque l'enregistrement est activé, avec Nouvelle session, ou selon la politique de session configurée.
- Chaque ligne enfant est une action, accompagnée de son titre, de sa date et de son heure, ainsi que d'une description résumant ses paramètres ou l'appel résolu lorsqu'ils sont disponibles. Une action de l'interface dont l'appel ne peut pas être résolu peut avoir une description vide.

La sélection d'une ou de plusieurs lignes détermine les entrées ciblées par les commandes de la barre d'outils et du menu contextuel. Le menu contextuel propose les mêmes commandes que la barre d'outils.

Lorsqu'une ligne d'action est sélectionnée, son objet résultat est sélectionné dans le panneau de données correspondant s'il est disponible ; sinon, ses objets d'entrée existants sont sélectionnés. DataLab bascule ensuite vers ce panneau de données.

Lorsque le mode d'enregistrement est activé, sélectionner une ligne de session (ou l'une de ses actions) fait de cette session la session d'enregistrement active.

Les actions qui ne sont pas compatibles avec l'état courant de l'espace de travail (par exemple parce qu'un identifiant d'objet référencé n'existe plus, ou parce que la forme de son tableau de données a changé) sont affichées avec un texte désactivé et une infobulle explicative. Tenter de les rejouer ouvre la boîte de dialogue de résolution de chaîne rompue décrite ci-dessous.

Les actions laissées obsolètes par un recalcul échoué ou interrompu, ou par des modifications de paramètres dont la propagation aux étapes en aval est encore en attente, sont surlignées sur fond ambre et munies d'une infobulle explicative. Les rejouer actualise leur résultat et efface ce marqueur.

Résolution des chaînes rompues

Lancer Rejouer ou Pas à pas (ou double-cliquer sur un élément de l'arborescence) sur une sélection contenant des actions incompatibles avec l'espace de travail courant — typiquement parce que les objets dont elles dépendent ont été supprimés — ouvre une boîte de dialogue de résolution. Deux choix sont proposés :

- Réparer et continuer : les actions rompues ainsi que toutes leurs descendantes en aval sont retirées de l'historique, puis les actions valides restantes de la sélection sont rejouées.
- Annuler : rien n'est modifié, ni l'historique ni l'espace de travail. La boîte de dialogue est de nouveau affichée à la tentative de rejeu suivante tant que la chaîne n'est pas réparée.

Affichage de l'état de l'espace de travail

Sous l'arborescence des actions, un widget en vue divisée affiche l"état de l'espace de travail tel qu'il était au moment de l'action sélectionnée :

- Tableau de gauche : liste les signaux qui étaient sélectionnés, avec la forme de leur tableau.
- Tableau de droite : liste les images qui étaient sélectionnées, avec leurs dimensions.

Ces informations aident l'utilisateur à comprendre le contexte dans lequel chaque action a été exécutée à l'origine et à diagnostiquer les problèmes de compatibilité lors du rejeu de la sélection actuelle.

Persistance

L'historique peut être enregistré de deux manières complémentaires :

- Intégré à l'espace de travail : lorsque l'espace de travail est enregistré au format HDF5 (`Fichier > Enregistrer dans un fichier HDF5`), le contenu du panneau d'historique est automatiquement sauvegardé aux côtés des signaux et des images. Le rechargement de l'espace de travail restaure les sessions enregistrées.
- Fichier d'historique autonome (`.dlhist`) : le fichier embarque à la fois les sessions enregistrées et tous les objets actuellement présents dans les panneaux Signaux et Images, qu'une action y fasse référence ou non. Le fichier est ainsi entièrement autonome :
  - L'ouverture d'un fichier `.dlhist` dans un espace de travail vierge (sans objet de données ni session d'historique existante) restaure directement les objets et les sessions enregistrés.

— Si l'espace de travail est déjà utilisé (il contient un objet de données ou une session d'historique), DataLab importe les objets dans de nouveaux groupes signal/image, réaffecte leurs identifiants afin d'éviter les collisions et ajoute les sessions d'historique importées qui référencent ces nouveaux identifiants.

> **Note**
>
> Une entrée d'historique dont les paramètres enregistrés ne peuvent plus être décodés (par exemple des données de ROI écrites par une version incompatible de DataLab) n'empêche pas le chargement du reste du fichier : l'action concernée est chargée comme incompatible — affichée sous forme de ligne désactivée avec une infobulle explicative, et jamais rejouée — tandis que toutes les autres sessions et actions sont chargées normalement. De telles entrées peuvent être nettoyées avec Supprimer les actions incompatibles.

> **Note**
>
> Chaque action conserve, dans le champ optionnel `provenance_activity_ids` (version 3 du schéma des actions de l'historique), les identifiants des activités de provenance enregistrées par sa dernière exécution réussie. Ce champ n'est jamais copié avec une action (les chaînes dupliquées et les sessions importées n'en ont pas), et les actions enregistrées par des versions antérieures n'en ont simplement pas. La suppression d'une session ne modifie pas l'enregistrement de provenance de l'espace de travail.

> **Avertissement**
>
> Le rejeu d'une session qui dépend de fichiers externes (par exemple l'ouverture d'un jeu de données depuis le disque) ne réussira que si ces fichiers sont toujours disponibles aux mêmes emplacements qu'au moment de l'enregistrement de la session.

Reconnexion de la chaîne lors d'une suppression

Lorsqu'un objet résultat est supprimé depuis le panneau signal ou image (et non depuis l'arborescence du panneau d'historique), et que cet objet a été produit par une étape de traitement enregistrée, le panneau d'historique reconnecte automatiquement la chaîne de traitement :

— Toutes les étapes aval qui consommaient l'objet supprimé sont recâblées pour utiliser la source de l'étape supprimée comme nouvelle entrée.
— Pour les opérations `2_to_1` (par exemple différence), la première source est utilisée pour la reconnexion.
— Si la source de l'étape supprimée n'existe plus (par exemple si elle a déjà été supprimée, ou si elle est supprimée en même temps), la suppression a lieu et les résultats dépendants restent inchangés : un unique avertissement liste les étapes de traitement concernées, chacune une seule fois.
— La suppression d'un objet qui n'a pas été produit par une étape de traitement enregistrée (par exemple un objet créé ou chargé, ou tout objet lorsque l'enregistrement était désactivé) laisse inchangés les résultats qui en dépendent, sans aucun avertissement.

Ce comportement reproduit la suppression d'un maillon d'une chaîne : les maillons adjacents se reconnectent pour préserver le flux de traitement.

> **Note**
>
> La reconnexion n'est déclenchée que par les suppressions effectuées depuis les panneaux Signaux ou Images. La suppression d'une action directement depuis l'arborescence du panneau d'historique suit un autre comportement : l'action sélectionnée est retirée de la chaîne au lieu de tronquer la session. Si des étapes en aval en dépendent, DataLab les conserve sous forme de chaîne indépendante en clonant l'objet intermédiaire nécessaire et en reconnectant ces

étapes au clone. La suppression d'une session supprime cette session dans son intégralité.

Recalcul automatique

> **Note**
>
> Lorsqu'un objet résultat est sélectionné dans le panneau signal/image et qu'il possède des paramètres de traitement (c'est-à-dire qu'il a été produit par un calcul 1-à-1), un onglet Traitement apparaît dans le panneau Propriétés. Cocher Recalcul automatique lors de l'édition dans cet onglet relancera automatiquement le calcul 300 ms après toute modification d'un paramètre.

3.1.5 Provenance des résultats

Pour les parcours de traitement pris en charge, DataLab enregistre comment un résultat a été obtenu : le traitement ou l'analyse appliqué, ses paramètres, les objets utilisés avec leur contenu exact et les versions des logiciels. Cet enregistrement est sauvegardé avec l'espace de travail HDF5 et relu à sa réouverture, par DataLab ou par DataLab-Web.

Cette page décrit comment consulter cet enregistrement pour un objet, comment le contrôler et le vérifier, comment appliquer à nouveau un traitement enregistré, et quel est son lien avec le panneau Historique. Pour échanger un espace de travail avec la provenance de ses traitements, voir Capsules d'espace de travail.

Consulter la provenance d'un objet

Sélectionnez un objet et ouvrez l'onglet Provenance du panneau des propriétés. Lorsque l'espace de travail contient un enregistrement de provenance pour cet objet, l'onglet affiche sa provenance :

- un résumé : l'exécution enregistrée qui a produit le contenu actuel, si les données restent à contrôler, et toute limite (par exemple des données altérées dans le fichier, ou un état observé sans production enregistrée) ;
- la Production de l'objet, ses Analyses enregistrées, ses États enregistrés antérieurs et, pour les espaces de travail enregistrés par d'anciennes versions, les Enregistrements d'un ancien espace de travail et l"Historique des métadonnées (non vérifié).

Chaque étape enregistrée est présentée sous forme de fiche : l'opération, comment, quand et par quel logiciel (DataLab Desktop ou DataLab Web) elle a été enregistrée, ses paramètres, ses entrées et ses résultats avec leur statut, et des liens qui agissent sur ce qu'ils accompagnent. Les liens marqués d'un triangle (▸) affichent une partie masquée de la vue et deviennent ▾ pour la masquer à nouveau ; une étape antérieure est affichée encadrée dans la fiche de l'étape qui l'a utilisée :

- Afficher l'étape enregistrée (sous une entrée) montre comment cette entrée a elle-même été produite, étape par étape, sans rien calculer. L'état réellement utilisé est affiché, même si l'objet d'entrée a été modifié ou retraité depuis : par exemple, si `S2` a été calculé à partir de `S1` filtré avec sigma = 2, puis `S1` filtré à nouveau avec sigma = 5, l'étape enregistrée de l'entrée de `S2` montre sigma = 2. Masquer l'étape enregistrée la masque à nouveau ;
- Aller à l'objet (sous une entrée ou un résultat) sélectionne, dans le panneau des signaux ou des images, l'objet qui la porte aujourd'hui (il a pu changer depuis le calcul) ;
- Vérifier cette étape recalcule l'étape séparément et compare chaque résultat au résultat enregistré : Exact, Dans la tolérance, Différent ou Non vérifié (avec la raison, par exemple une entrée d'origine absente). L'espace de travail n'est pas modifié ;
- Action de l'historique sélectionne l'action de l'historique dont la dernière exécution a enregistré l'étape, lorsqu'elle existe ;
- Afficher les détails affiche le contexte de l'étape (options, alignement en X, métadonnées lues) et les versions des logiciels.

La consultation lit uniquement l'enregistrement de provenance : elle ne calcule jamais, ne modifie jamais l'espace de travail et n'ajoute jamais d'action à l'historique. La vue est bornée et les valeurs longues sont abrégées : une vue tronquée ne signifie pas que des enregistrements ont été supprimés.

Les boutons situés sous les fiches agissent sur l'objet :

— Contrôler les données compare l'objet, les entrées directes et les résultats de sa production et les étapes antérieures affichées à leurs états enregistrés, sans calculer. Les étapes affichées et les détails ouverts le restent. Une entrée ou un résultat supprimé est signalé absent immédiatement, sans contrôle (rien n'a besoin d'être lu pour le savoir) ;
— Vérifier le calcul vérifie la production de l'objet, comme Vérifier cette étape.

États et messages

La provenance n'invente jamais ce qui est inconnu. Les principaux messages sont :

Message	Signification
Données à contrôler	Rien n'a encore été comparé : utilisez Contrôler les données
Cette entrée a changé depuis le calcul	L'objet d'entrée ne contient plus l'état utilisé par le calcul ; l'état enregistré reste décrit
Entrée d'origine absente	L'objet d'entrée a été supprimé (ou son identifiant est désormais porté par un autre objet, voir Ajouter un espace de travail)
Le résultat a été modifié depuis le calcul	Le résultat a été modifié après son calcul : il est signalé comme altéré plutôt que considéré comme fiable
Les données de l'objet ne sont pas disponibles	Les données ne peuvent pas être lues pour être comparées (elles ne sont pas contrôlées)
L'état ne peut pas être contrôlé (pas d'empreinte)	L'enregistrement ne permet pas de comparaison
État observé, sans production enregistrée	L'objet figure dans l'enregistrement, mais la manière dont il a été produit est inconnue (par exemple des données chargées depuis un fichier)

Appliquer à nouveau un traitement

Lorsque le dernier état enregistré d'un objet a été produit par un traitement de DataLab, l'onglet Traitement du panneau des propriétés restaure cet appel : son opération, ses paramètres et chacune de ses entrées, dans l'ordre et avec son rôle (source, opérande, ou chacun des objets agrégés). Cela s'applique aux traitements combinant plusieurs objets (par exemple la différence de deux signaux, la moyenne de plusieurs images) comme aux traitements 1 vers 1, et aux espaces de travail rouverts depuis un fichier, enregistrés par DataLab ou par DataLab-Web.

— Avec des paramètres, modifiez-les puis cliquez sur Appliquer ;
— Sans paramètres, cliquez sur Réappliquer.

Le résultat est mis à jour sur place avec les entrées enregistrées, jamais avec la sélection courante, et la nouvelle exécution est enregistrée. Lorsque le traitement ne peut pas être appliqué à nouveau, l'onglet en indique la raison (par exemple un objet d'entrée n'existe plus, les paramètres enregistrés ne peuvent pas être utilisés, ou l'opération n'est pas disponible dans cette version de DataLab).

Lorsqu'aucune provenance n'est enregistrée pour un objet, les métadonnées de traitement stockées avec lui sont utilisées comme auparavant.

Lien avec le panneau Historique

Lorsque le panneau Historique enregistre une session, chaque action conserve les identifiants des activités de provenance enregistrées par sa dernière exécution réussie. Sélectionnez une action et utilisez Afficher la provenance de l'exécution (menu ou barre d'outils du panneau Historique) pour les afficher, avec un onglet par résultat (ou par objet analysé). Si les paramètres de l'action ont été modifiés sans qu'elle soit exécutée à nouveau, la boîte de dialogue indique que la provenance décrit la dernière exécution réussie.

L'action de l'historique est la recette, la provenance enregistre chaque exécution :

— appliquer à nouveau un traitement depuis l'onglet Traitement, ou une analyse depuis l'onglet Analyse, est enregistré dans la provenance du résultat ;
— en mode d'édition du panneau Historique, les paramètres modifiés deviennent aussi ceux de l'action de l'historique, qui est liée à cette nouvelle exécution. Pour une commande produisant plusieurs résultats (par exemple plusieurs

filtres appliqués en une fois), seuls les paramètres du résultat modifié changent dans l'action, et rejouer l'action les applique à ce résultat pour chaque objet traité ;
— hors du mode d'édition, l'action de l'historique reste inchangée : la rejouer applique à nouveau ses propres paramètres.

La capture automatique de provenance ne dépend pas du mode Enregistrer du panneau Historique. Enregistrer une recette et enregistrer une exécution scientifique sont deux opérations différentes.

Ces identifiants sont stockés dans le champ optionnel `provenance_activity_ids` des actions de l'historique (version 3 du schéma des actions de l'historique). Ils ne sont jamais copiés avec une action, et les actions enregistrées par des versions antérieures n'en ont simplement pas. La suppression d'une session de l'historique ne modifie pas l'enregistrement de provenance.

Ajouter un espace de travail

Lorsqu'un fichier d'espace de travail est ouvert sans vider l'espace de travail actuel, ses données sont ajoutées mais sa provenance antérieure n'est pas importée : l'enregistrement de provenance de l'espace de travail actuel reste inchangé, et les traitements ultérieurs des objets ajoutés sont enregistrés normalement. Un objet ajouté n'hérite jamais de la provenance d'un autre objet, même lorsqu'il porte l'identifiant d'un objet supprimé de l'espace de travail actuel : un tel objet n'est ni affiché ni utilisé comme entrée ou résultat d'origine d'une étape enregistrée, y compris lors de sa vérification.

DataLab signale cette limite avant toute modification :
— lorsque DataLab demande s'il faut vider l'espace de travail actuel, les fichiers sont choisis d'abord et, si l'un d'eux contient une provenance, la question le précise ;
— sinon, un avertissement unique propose d"Ajouter sans la provenance ou d"Annuler.

Pour conserver la provenance d'un fichier, ouvrez-le en remplaçant l'espace de travail actuel. Lorsque des espaces de travail sont chargés depuis un script avec *`load_h5_workspace()`*, aucune question n'est posée et l'avis est journalisé.

Ce que la provenance ne fait pas

— Elle ne conserve pas tous les tableaux antérieurs. Les états antérieurs sont décrits par leurs empreintes ; leurs données ne restent disponibles que tant qu'un objet les contient. Supprimer ou remplacer les données numériques d'une entrée peut empêcher le rejeu, même après sauvegarde et réouverture. Par exemple, sauvegarder un signal filtré après avoir supprimé sa source conserve la description de la source, pas ses tableaux. Conservez les entrées nécessaires et les résultats enregistrés avant de sauvegarder.
— Les modifications des seules ROI constituent une exception limitée. Si les données numériques d'une entrée sont inchangées, ses ROI enregistrées peuvent être rétablies sur une copie pour la vérification, y compris lorsqu'une détection a ajouté des ROI. Cela ne reconstruit pas des valeurs disparues. Une sortie absente ou altérée reste une référence de comparaison invalide : rétablir les ROI d'une entrée ne répare pas la référence.
— Les valeurs d'analyse ne sont pas toutes archivées. Une analyse ultérieure peut remplacer un résultat sous la même clé de métadonnées. L'artefact antérieur reste décrit comme `replaced`, sans nécessairement conserver son ancienne valeur.
— Elle ne nettoie pas l'espace de travail. Les essais et les objets intermédiaires restent jusqu'à leur suppression. Leurs étapes enregistrées restent dans le registre de provenance : supprimer un objet ne purge jamais ses activités ni les informations sensibles.
— Elle ne rend pas tout vérifiable. Les modifications faites en dehors des traitements enregistrés peuvent être observées ultérieurement sans production connue. Une opération sans contrat peut être enregistrée comme opaque, mais n'est jamais rejouée. Les anciens enregistrements de métadonnées ont des états d'entrée inconnus et parfois des paramètres ou environnements incomplets ; ils ne sont ni qualifiés rétroactivement ni exécutés à partir de leurs noms Python.
— Elle ne prouve pas la validité scientifique. Exact ou Dans la tolérance indique que le calcul enregistré est reproduit selon la règle déclarée pour cette opération, pas que le traitement était le bon.
— Elle n'identifie pas l'auteur. Les empreintes montrent que le contenu n'a pas été altéré ; elles n'authentifient pas qui l'a produit, et les capsules ne sont pas signées.

Périmètre de capture et conditions de vérification

Les traitements ordinaires et groupés, les analyses, les conversions signal/image et l'extraction de ROI utilisent les parcours de capture pris en charge. Traitement, Recalculer et le rejeu d'un calcul pris en charge depuis le panneau Historique enregistrent une nouvelle exécution ; la vérification elle-même travaille sur des copies et n'enregistre aucune nouvelle exécution. Un aperçu accepté peut devenir un résultat validé, mais un aperçu abandonné, un calcul en erreur ou une annulation ne sont pas des exécutions réussies. Une détection réussie qui ne trouve rien est un résultat normal enregistré.

Importer ou créer un objet n'enregistre pas comment son acquisition physique ou ses conversions antérieures ont été réalisées. Éditer directement les données, les ROI ou les métadonnées n'enregistre pas en soi un appel scientifique. Les ajustements interactifs Desktop et la création d'un signal à partir d'une table de marqueurs ne sont actuellement pas des appels scientifiques enregistrés, même lorsqu'une action d'historique éditable existe.

Les plugins, macros, scripts et clients de l'API distante sont enregistrés lorsqu'ils utilisent les processeurs pris en charge. Du code qui crée ou modifie directement des objets peut les contourner. La capture ne bloque pas le traitement : si elle échoue, un résultat peut exister sans activité ; l'échec est journalisé et compté, pas réparé silencieusement.

Un rejeu nécessite un contrat d'opération local qualifié pour la version enregistrée, des paramètres décodables, un contexte applicable, des données d'entrée disponibles et des références de sortie valides. Les spectres de phase signal et image sont identifiés mais non qualifiés. Une opération, une version, un codec ou une politique inconnus produisent un refus explicite : le lecteur ne télécharge jamais de code pour en deviner le sens.

Sigima augmente la version d'un contrat d'opération dès qu'une modification du calcul change son résultat, quelle que soit la version de DataLab. Les activités enregistrées avec une version antérieure du contrat sont alors refusées explicitement (contrat non pris en charge) au lieu d'être rejouées avec un résultat différent, et leur traitement ne peut pas être réappliqué tel qu'enregistré : recalculez le résultat pour l'enregistrer avec la version actuelle. Il en va de même pour les règles versionnées du contexte d'exécution, comme la sélection des pixels d'une image par les régions d'intérêt. Voir Contrats d'opération et versions dans la documentation de Sigima.

Les empreintes scientifiques couvrent les données numériques, les types, les coordonnées et les unités ; les ROI sont contrôlées séparément. Les titres, les paramètres d'affichage et les autres métadonnées ne sont pas automatiquement couverts. Seules les dépendances déclarées, telles que les options globales, l'alignement en X ou les métadonnées lues par le calcul, sont reproduites. Par exemple, modifier les paramètres d'ajustement nécessaires à `evaluate_fit` peut entraîner un refus de rejeu même si les valeurs du signal sont inchangées.

Consultez les règles communes et le périmètre de capture et d'interopérabilité pour les conditions et les parcours couverts. Les limites d'échange, de confidentialité et de révision sont décrites dans Capsules d'espace de travail.

3.1.6 Capsules d'espace de travail

Une capsule (fichier `.dlcapsule`) contient un espace de travail et une description de la provenance de ses traitements (voir Provenance des résultats), afin qu'une autre personne puisse les vérifier et les rejouer, avec DataLab ou avec DataLab-Web. Les capsules sont créées, lues et vérifiées par DataLab-Capsule : ouvrir ou inspecter une capsule n'exécute jamais rien de ce qu'elle contient.

Les capsules sont gérées depuis Fichier > Capsule d'espace de travail…, une boîte de dialogue comportant les mêmes onglets que dans DataLab-Web. Trois résultats sont toujours présentés séparément :

- l"intégrité d'une capsule, calculée à partir de ses fichiers (rien n'est recalculé) ;
- les documents fournis (rapports de vérification et revues) : ils sont vérifiés par rapport à la capsule, mais leurs verdicts sont ceux qu'affirment leurs auteurs ;
- le rejeu local d'une activité de l'espace de travail ouvert, recalculée par DataLab et comparée à son résultat enregistré.

Exporter

Exporte l'espace de travail actuel sous forme de capsule :

- Nom et Description de la capsule ;
- Révision de … : affiché lorsque l'espace de travail a été ouvert depuis une capsule, y compris après avoir été enregistré dans un fichier d'espace de travail puis rouvert. Lorsque cette case est cochée, la nouvelle capsule désigne cette capsule comme sa révision parente. Les pièces jointes de la révision parente ne sont pas copiées : la boîte de dialogue les énumère, afin de pouvoir les sélectionner à nouveau si elles s'appliquent toujours ;
- Référencer l'espace de travail au lieu de l'inclure : la capsule contient alors l'adresse (IRI absolue) où l'espace de travail sera publié, avec sa taille et son SHA-256, et l'espace de travail est enregistré dans un fichier HDF5 distinct, à publier à cette adresse ;
- Pièces jointes : fichiers légers (protocole, notes, petits tableaux, 16 Mio au plus chacun) déclarés dans la capsule avec leur taille et leur empreinte. Ils ne sont jamais ouverts ni exécutés.

Exporter la capsule… demande où enregistrer la capsule (et, pour un espace de travail référencé, le fichier de l'espace de travail).

> **Avertissement**
>
> Examinez les données, métadonnées, titres, essais, dates et pièces jointes avant de partager. Rien n'est anonymisé ni expurgé automatiquement. Supprimer un objet ne garantit pas que ses traces disparaissent du registre de provenance.

L'export transporte l'espace de travail actuel, son registre et les pièces jointes choisies, pas toutes les ressources externes utilisées en amont. Il ne sélectionne pas automatiquement l'analyse minimale nécessaire à un résultat : des essais et objets intermédiaires peuvent voyager avec lui. Copier uniquement un résultat dans un autre espace de travail ne fait pas de celui-ci une analyse complète vérifiable. Préparez un échange réduit en conservant ou reconstruisant volontairement les entrées utiles et en vérifiant ce qui reste vérifiable, pas en modifiant manuellement le registre.

La licence du logiciel ne détermine pas celle des données. Le manifeste contient une déclaration de licence de l'espace de travail ; lorsqu'aucune n'est fournie, il utilise la formulation neutre « No licence was specified for this workspace. » (aucune licence n'a été spécifiée pour cet espace de travail).

Inspecter

Choisir une capsule… vérifie une capsule sans l'ouvrir et affiche :

- son résumé : nom, description, empreinte, révision parente, espace de travail inclus ou référencé, licence, et ses activités (celles qui peuvent être rejouées) ;
- ses Pièces jointes, vérifiées par rapport au manifeste et enregistrées seulement lorsque vous cliquez sur Enregistrer… ;
- son Intégrité : archive, manifeste, espace de travail et états enregistrés. Pour un espace de travail référencé, Sélectionner l'espace de travail référencé… fournit le fichier, qui doit correspondre à la taille et au SHA-256 déclarés dans la capsule ;
- les Documents fournis ajoutés avec Ajouter un rapport de vérification… et Ajouter une revue…, chacun vérifié par rapport à la capsule.

Rédiger une revue crée une revue de cette révision : un fichier JSON distinct (relecteur, déclaration, verdict facultatif approuvé, contesté ou commentaire, rapports de vérification cités). La capsule elle-même n'est pas modifiée.

Ouvrir comme espace de travail remplace l'espace de travail actuel par celui de la capsule (après confirmation si l'espace de travail n'est pas vide). Cette action n'est disponible que pour une capsule intacte. Une capsule exportée ensuite depuis cet espace de travail est proposée comme une révision de celle-ci. L'ouverture d'une capsule est enregistrée dans le panneau Historique.

Pour une capsule référencée, obtenez le fichier HDF5 séparément : son adresse n'est pas téléchargée automatiquement, et sa taille et son empreinte doivent correspondre. Un registre ne fournit pas cet espace de travail à votre place.

Rejeu local

Énumère les activités enregistrées de l'espace de travail ouvert. Rejouer recalcule une activité séparément et compare chaque résultat au résultat enregistré : Exact, Dans la tolérance, Différent ou Non vérifié (avec la raison). L'espace de travail n'est pas modifié. Enregistrer le rapport… enregistre le rapport de vérification dans un fichier JSON, à fournir avec la capsule ou à citer dans une revue.

Révisions

Énumère les révisions exportées, inspectées ou récupérées depuis un registre sur cet ordinateur (avec leur révision parente), et compare deux d'entre elles : changements scientifiques, de présentation, de stockage et d'intégrité, d'environnement et de pièces jointes, avec une conclusion scientifique qui n'affirme jamais une égalité au-delà de ce qu'enregistrent les capsules. La liste est conservée dans le dossier de configuration de DataLab (100 révisions au plus) ; c'est une commodité, jamais une source de vérité : ce sont les fichiers des capsules qui le sont.

Une capsule est immuable et désigne au plus un parent direct. Si vous ouvrez A puis exportez B, l'export suivant propose toujours A comme parent, pas B ; la sauvegarde et la réouverture de l'espace de travail conservent cette origine. Inspecter un autre fichier ne la change pas. Ajouter un espace de travail conserve le registre et le parent actuels sans fusionner les provenances (voir Ajouter un espace de travail).

Deux exports peuvent avoir des empreintes différentes sans nouveau calcul : le stockage HDF5, les dates ou la présentation peuvent changer. `none_detected` signifie qu'aucun changement scientifique n'a été détecté dans les activités et états enregistrés, pas que tout ce qui est hors registre est égal. La modification d'un protocole joint est listée parmi les pièces jointes ; le lecteur, pas la comparaison, juge son importance scientifique. Il n'existe pas de fusion automatique de travaux concurrents, d'édition collaborative ou de résolution de conflits.

Registre

Un registre (facultatif, servi par DataLab-Capsule) publie les manifestes des capsules, leurs pièces jointes et leurs revues, jamais les espaces de travail. Tout ce qu'il renvoie est vérifié par DataLab.

- Adresse du registre et Jeton de publication : l'adresse est enregistrée dans la configuration ; le jeton n'est conservé que pour la session en cours ;
- Lister les révisions énumère les révisions publiées ; Récupérer vérifie une révision et ses revues, l'ajoute à l'onglet Révisions et propose ses pièces jointes stockées, vérifiées par rapport au manifeste ;
- une fois un registre défini, l'onglet Inspecter propose Publier dans le registre (manifeste, pièces jointes et revues fournies valides) et Publier la revue.

Le registre de référence n'offre pas de comptes, d'identité forte des auteurs, de quotas, de limitation de débit, d'API de suppression ni d'autorisations fines. Son jeton de publication ne prouve pas l'identité de l'auteur scientifique. Consultez les recommandations de déploiement et de confiance existantes. Les revues et rapports ciblent une révision ; une assertion d'auteur contrôlée ne remplace pas un rejeu local et ne constitue pas un circuit complet d'approbation.

Depuis des scripts

Les capsules peuvent être exportées et ouvertes depuis des scripts et des macros (voir Contrôle à distance (XML-RPC)), avec *`export_capsule()`* et *`open_capsule()`* :

```
fingerprint = proxy.export_capsule(
    "chain.dlcapsule", name="chain", attachments=["protocol.txt"]
)
proxy.open_capsule("chain.dlcapsule")
```

La ligne de commande `workspace-capsule` de DataLab-Capsule permet aussi de valider, inspecter, comparer, vérifier et publier des capsules.

Sa commande `verify` n'effectue aucun calcul Sigima. Sans h5py, le contrôle de l'espace de travail se limite à la taille et au SHA-256 (`integrity_only`), pas aux états HDF5 individuels. Un espace de travail référencé non fourni réduit encore les contrôles possibles.

Limites

DataLab vérifie la taille d'une capsule, et de l'espace de travail qu'elle référence, avant de la lire, et n'écrit jamais une capsule qu'il refuserait d'ouvrir. Ces limites sont celles de DataLab-Capsule (un espace de travail de 4 Gio au plus) : ce sont des tailles acceptées, pas une capacité testée.

Ce qui a été mesuré, sur un ordinateur portable de 2026 (32 Gio de mémoire vive) :

- inspecter ou ouvrir une capsule de 32 Mio depuis la boîte de dialogue prend de 0,2 à 0,5 s, pendant lesquelles DataLab ne répond pas (curseur d'attente) ; l'exporter prend 0,25 s. Les capsules plus grandes prennent plus de temps, à peu près proportionnellement ; la bibliothèque vérifie une capsule de 234 Mio en moins d'une seconde ;
- ouvrir une capsule demande environ sa taille en mémoire, en plus de l'espace de travail qu'elle remplace ;
- enregistrer la provenance d'un traitement coûte les empreintes de ses entrées et de ses résultats : rien de mesurable pour un petit signal, +29 ms pour une image de 2048×2048 filtrée en 450 ms ;
- consulter la provenance d'un résultat prend moins d'une milliseconde, même avec 1000 étapes enregistrées ; lister 1000 étapes dans Rejeu local prend environ une demi-seconde.

Chaque traitement est un peu plus lent à mesure qu'un panneau contient plus d'objets, avec ou sans provenance (environ 0,1 s dans un panneau vide, 0,3 s avec 1000 objets). DataLab-Web limite les capsules à 256 Mio.

Tableau 1 – Politiques de fichiers acceptés, pas capacités garanties

Lecteur	Espace de travail / manifeste
Desktop et bibliothèque	4 Gio / 64 Mio ; l'archive a une limite de taille totale distincte
Web	256 Mio / 16 Mio ; archive également limitée à 256 Mio
Registre de référence	Pas d'espace de travail ; manifeste 8 Mio, revue 1 Mio

Chaque capsule accepte au plus 8 pièces jointes de 16 Mio chacune. La mémoire peut contenir plusieurs représentations de l'archive et du HDF5 pendant l'export ; un fichier valide peut dépasser la politique d'une autre édition. Le registre a aussi des bornes de volume JSON et de structure, même sans limite fixe du nombre d'activités. La troncature de la vue ne supprime pas les enregistrements. Consultez les limites techniques et capacités mesurées.

Le rejeu local vérifie une activité sur des copies ; il ne reconstruit pas une chaîne entière, ne restaure pas tous les états manquants et ne fournit pas de workflow pour de nouveaux jeux de données. Les versions de l'environnement sont décrites, pas réinstallées : une capsule n'est pas un système d'exploitation avec toutes ses dépendances, et une comparaison réussie ne garantit pas une reproductibilité universelle entre machines.

Les empreintes détectent les changements mais n'authentifient pas les auteurs ; les capsules et revues ne sont pas signées et leurs empreintes peuvent être recalculées après modification. L'import n'exécute jamais les macros, notebooks, pièces jointes ou noms Python reçus ; le rejeu explicite utilise des opérations locales enregistrées et qualifiées. Le profil prescrit accepte les espaces de travail natifs pris en charge, pas des fichiers ZIP/HDF5 arbitraires : les entrées ZIP ne sont pas compressées (indépendamment de la compression interne HDF5), et les liens et archives chiffrées sont refusés. Les contrôles de format ne remplacent pas un bac à sable : le décodage HDF5 et les dépendances restent du code de confiance sur le Desktop.

Le profil 0.1 est expérimental et les versions inconnues sont refusées. Les métadonnées RO-Crate/PROV ne garantissent pas que tout lecteur tiers puisse restaurer ou rejouer DataLab. Consultez les règles communes et l"analyse de sécurité. Le Desktop et la CLI peuvent travailler hors réseau une fois installés ; le Web télécharge son runtime au démarrage et certaines dépendances optionnelles au premier usage. Les calculs restent locaux ; ni un registre ni les pièces jointes ne certifient l'acquisition physique.

3.2 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.

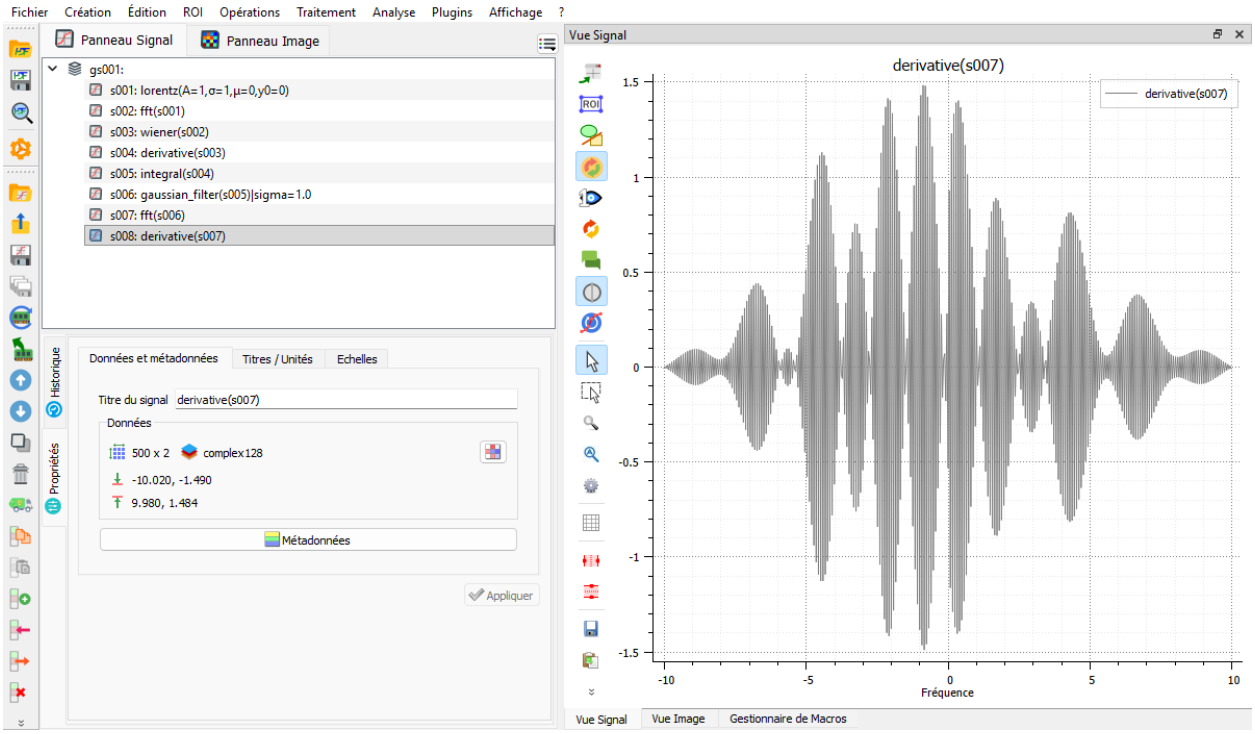


Fig. 12 – Fenêtre principale de DataLab : vue du traitement du signal

3.2.1 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).
- Exporter, inspecter, rejouer et comparer des capsules d'espace de travail, c'est-à-dire un espace de travail avec la provenance de ses traitements (voir Capsules d'espace de travail).
- Modifier les préférences de DataLab (voir Préférences).

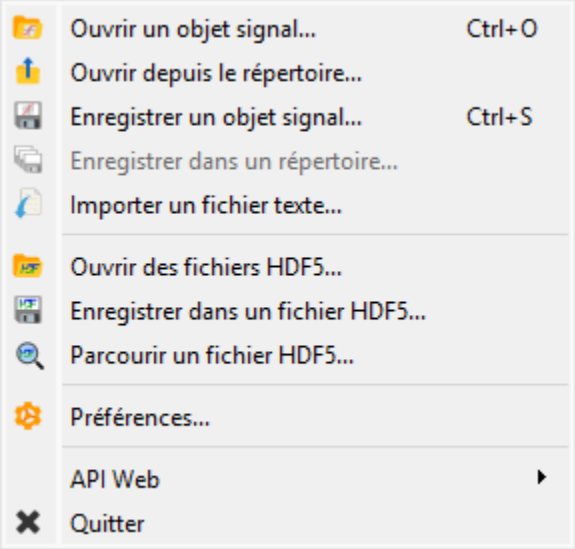


Fig. 13 – Capture d'écran du menu « Fichier ».

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.

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.).

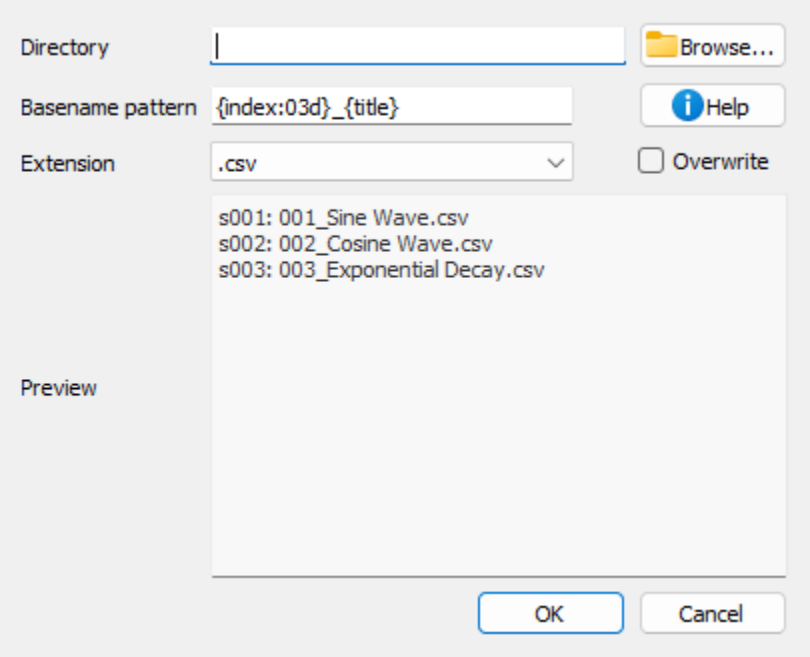


Fig. 14 – Boîte de dialogue Enregistrer les signaux dans un répertoire.

— 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 du signal
— `{index}` : Index du signal dans la sélection (avec remplissage de zéros)
— `{count}` : Nombre total de signaux sélectionnés
— `{xlabel}`, `{xunit}`, `{ylabel}`, `{yunit}` : 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 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.

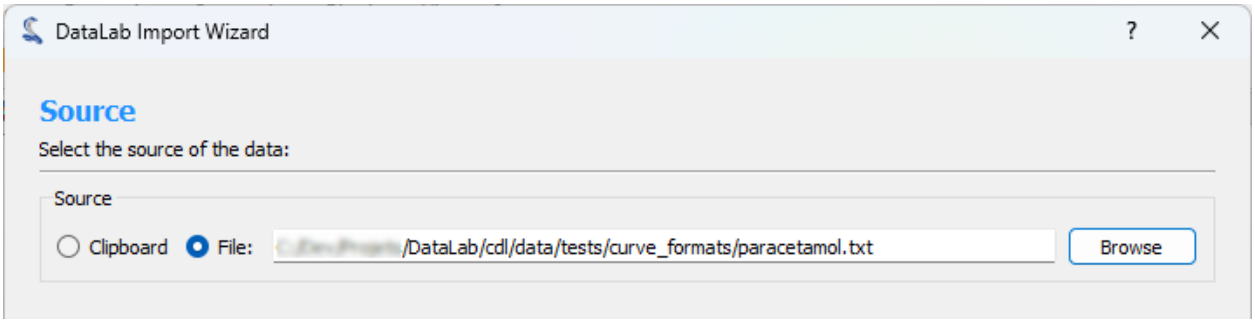


Fig. 15 – Etape 1 : Sélectionner la source

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.

3.2.2 Créer des signaux

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

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

suite sur la page suivante

Tableau 2 – suite de la page précédente

Icône	Modèle	Équation
	Distribution uniforme	$y[i]$ suit une distribution uniforme entre deux bornes configurables
	Gaussienne	$y = y_0 + A \cdot \exp\left(-\frac{1}{2} \cdot \left(\frac{x-\mu}{\sigma}\right)^2\right)$
	Lorentzienne	$y = y_0 + \frac{A}{1 + \left(\frac{x-\mu}{\sigma}\right)^2}$
	Voigt	$y = y_0 + A \cdot \frac{\Re\left(w(z)\right)}{\Re\left(w(j/\sqrt{2})\right)}$ avec $w(z) = \exp\left(-z^2\right) \cdot \mathrm{erfc}(-j \cdot z)$ et $z = \frac{x - \mu + j \cdot \sigma}{\sqrt{2} \cdot \sigma}$
	Corps noir (loi de Planck)	$y = \frac{2hc^2}{\lambda^5 \left(\exp\left(\frac{hc}{\lambda k T}\right) - 1\right)}$
	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	$y = \left(\begin{cases} y_0 & \text{si } x < t_0 \\ y_0 + A \cdot \frac{x - t_0}{t_r} & \text{si } t_0 \le x < t_0 + t_r \\ y_0 + A & \text{si } x \ge t_0 + t_r \end{cases} \right) + \mathcal{N}\left(0, \sigma_n\right)$ 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 2 – suite de la page précédente

Icône	Modèle	Équation
		$y(x) = \left\{ \begin{array}{ll} y_0 & \text{si } x < t_0 \\ y_0 + A \cdot \dfrac{x - t_0}{t_r} & \text{si } t_0 \leq x < t_0 + t_r \\ y_0 + A & \text{si } t_0 + t_r \leq x < t_1 \\ y_0 + A - A \cdot \dfrac{x - t_1}{t_f} & \text{si } t_1 \leq x < t_1 + t_f \\ y_0 & \text{si } x \geq t_1 + t_f \end{array} \right. + \mathcal{N}(0, \sigma_n)$ 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 $d = t_{\text{FWHM}} - \dfrac{t_r + t_f}{2}$ à partir de la largeur à mi-hauteur t_{FWHM} **Avertissement** La durée du plateau d ne doit pas être négative.
	Impulsion carrée	
	Polynôme	$y = a_0 + a_1 \cdot x + a_2 \cdot x^2 + \ldots + a_5 \cdot x^5$
	Personnalisé	Saisie manuelle des valeurs X et Y

Pour les modèles gaussien, lorentzien et de Voigt, A est la hauteur signée du pic au-dessus de la ligne de base y_0, exprimée dans l'unité Y du signal.

Le signal de corps noir est la luminance énergétique spectrale (par unité de longueur d'onde) d'un corps noir à la température T : les longueurs d'onde λ sont exprimées en mètres et la luminance énergétique spectrale en W·sr^{-1}·m^{-3}. Ces titres et unités d'axes sont définis par défaut et peuvent être modifiés dans la boîte de dialogue de création.

3.2.3 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.

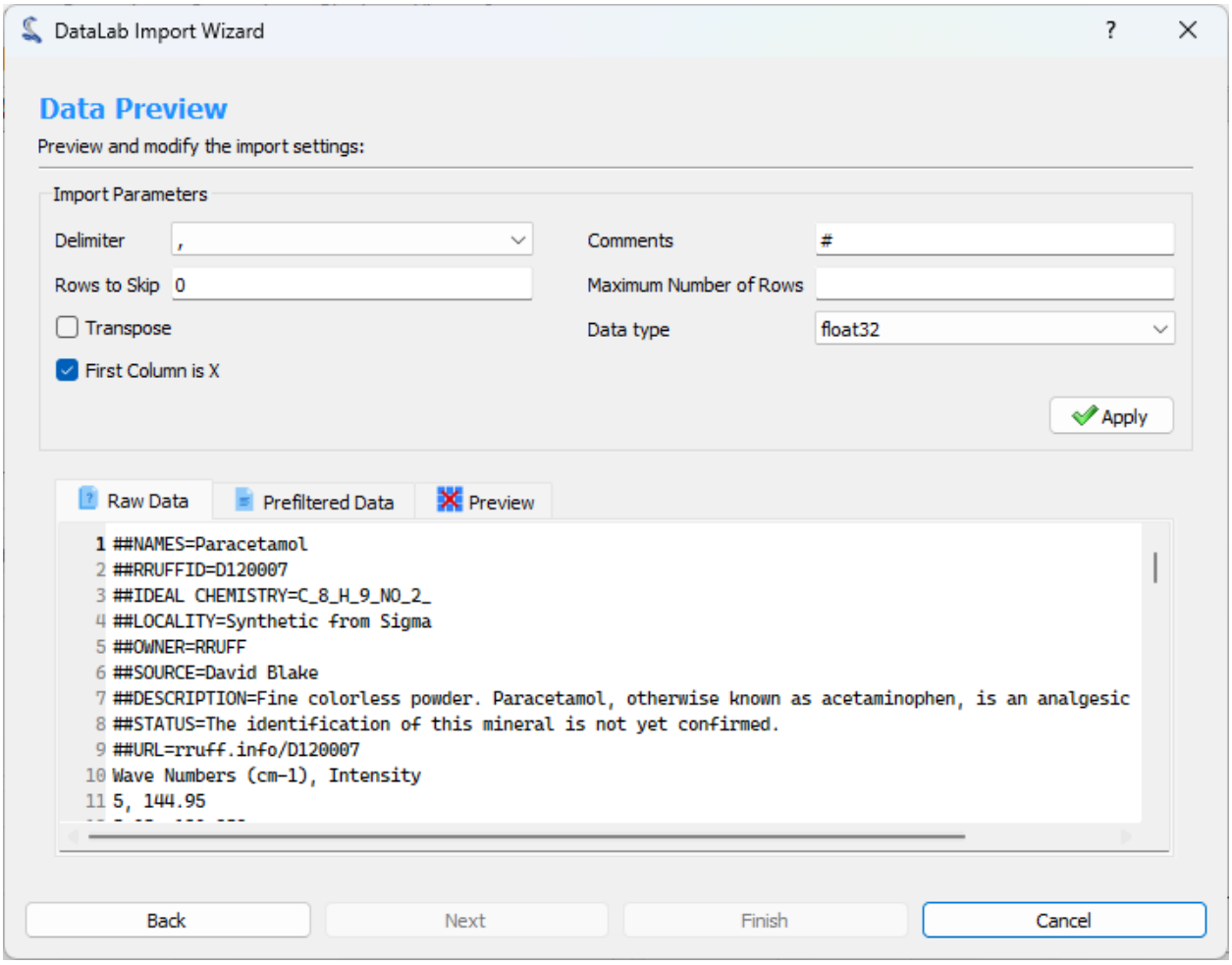


Fig. 16 – Etape 2 : Configurer l'importation

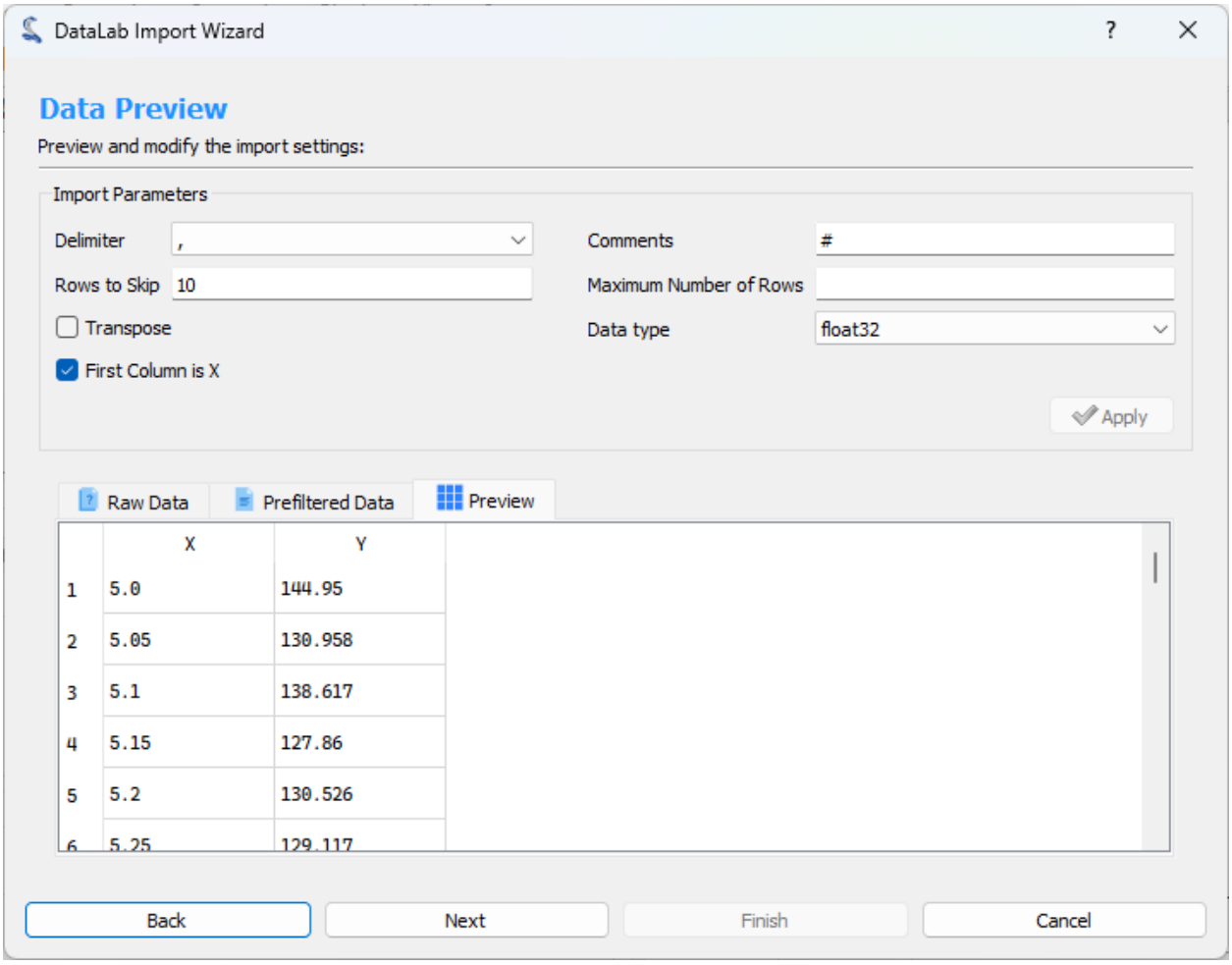


Fig. 17 – Etape 2 : Prévisualiser le résultat

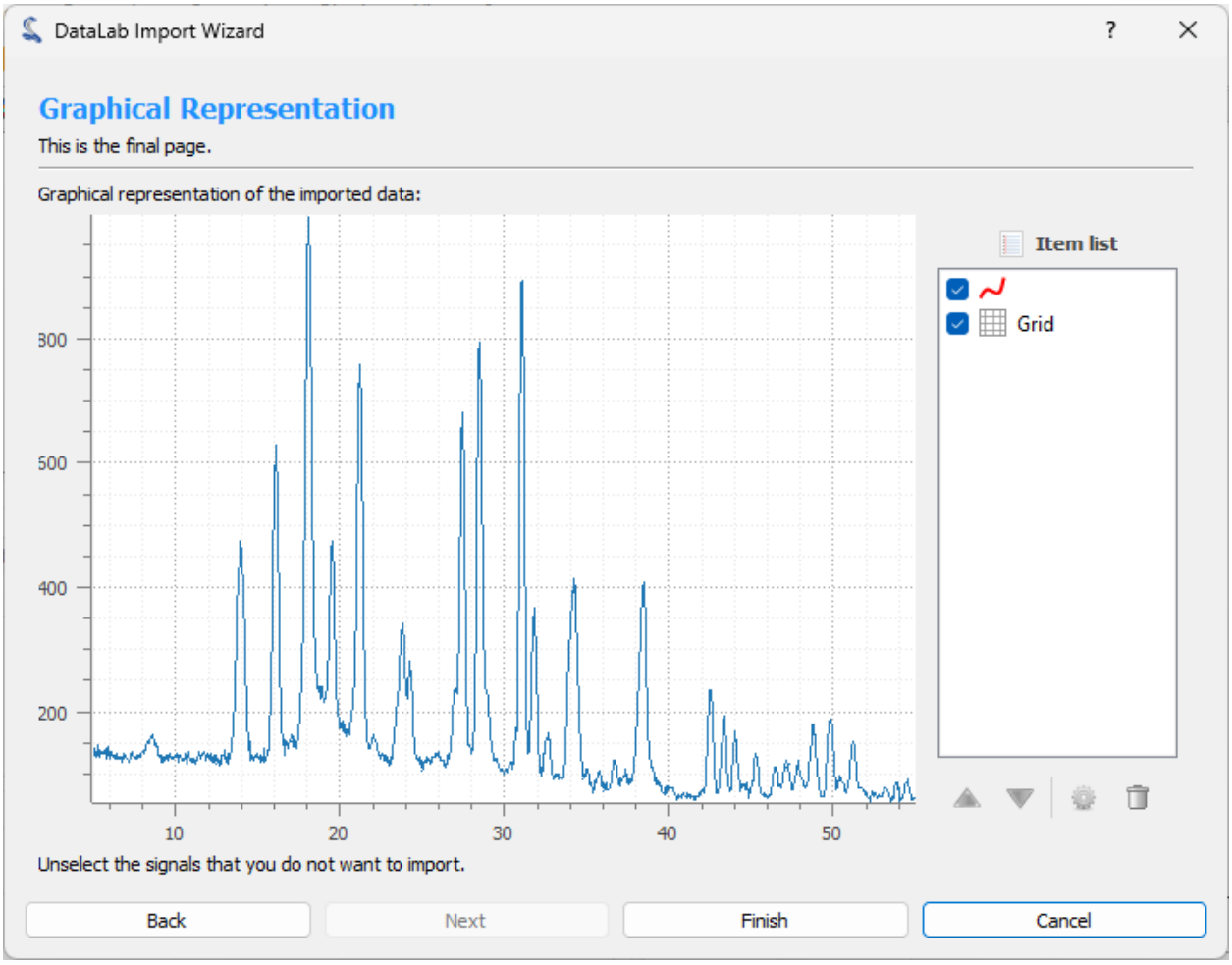


Fig. 18 – Etape 3 : Afficher la représentation graphique



Fig. 19 – Capture d'écran du menu « Création ».

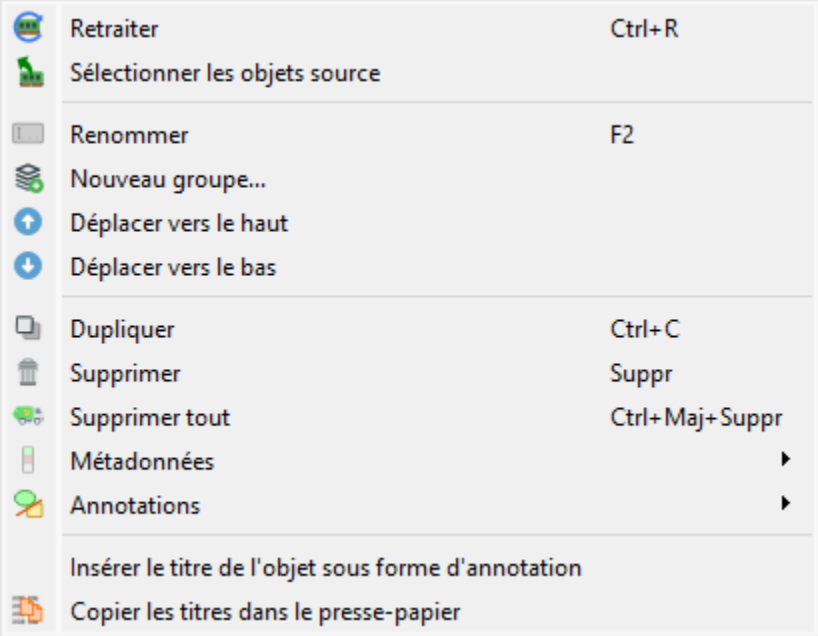


Fig. 20 – Capture d'écran du menu « Édition ».

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.

Cette action actualise également les résultats d'analyse (statistiques, FWHM, centroïde, détection de pics ou de contours, etc.) à partir de leurs paramètres enregistrés. Les résultats d'analyse ne sont pas recalculés automatiquement lorsque vous modifiez une région d'intérêt, les données ou les propriétés de l'objet : les résultats existants restent inchangés jusqu'à ce que vous déclenchiez explicitement cette action, vous laissant ainsi un contrôle total sur le moment où les analyses sont actualisées.

> **Note**
>
> Cette action n'est disponible que pour les signaux disposant de paramètres de traitement enregistrés (signaux créés par des opérations de traitement) ou de paramètres d'analyse enregistrés (opérations d'analyse exécutées précédemment sur ceux-ci).

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

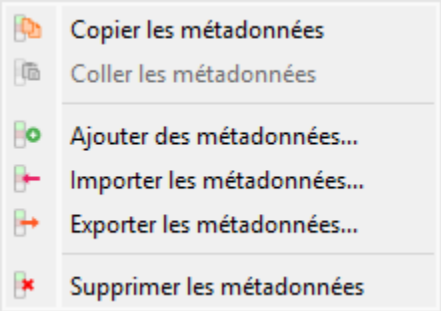


Fig. 21 – Capture d'écran du sous-menu « Métadonnées ».

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.

Lorsque vous collez des métadonnées, une boîte de dialogue vous permet de sélectionner les catégories de métadonnées à conserver à partir du presse-papiers :

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.

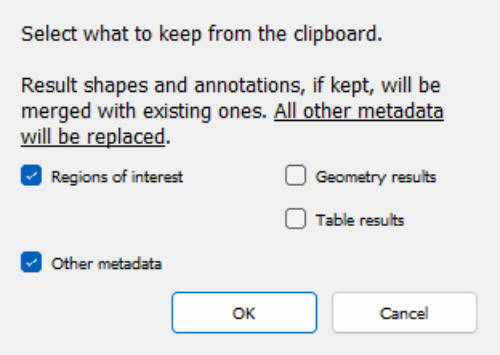


Fig. 22 – Capture d'écran de la boîte de dialogue « Coller les métadonnées ».

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.

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

- Clé de métadonnée : Saisir le nom du champ de métadonnée à ajouter (lettres, chiffres, `_`, `.` et `-`, par exemple une clé de plugin telle que `plugin.org.example.my-plugin.gain`)
- Clés connues : Choisir une clé déjà présente sur les signaux sélectionnés, ou une clé attendue par les méthodes des plugins d'application installés, pour la copier dans la clé de métadonnée
- Modèle de valeur : Définissez un modèle pour la valeur des métadonnées en utilisant des chaînes de format Python
- Motif d'extraction : Facultativement, extraire la valeur du texte formaté avec une expression régulière Python : son premier groupe, ou toute la correspondance si elle n'a pas de groupe, devient la valeur
- Sans correspondance : Laisser inchangés les signaux sans correspondance (par défaut), ou signaler une erreur
- Conversion : Choisissez comment stocker la valeur (chaîne, flottant, entier ou booléen)
- Facteur d'échelle : Multiplier les valeurs numériques, par exemple `0.001` pour convertir des millisecondes en secondes
- Aperçu : Voir comment les métadonnées seront ajoutées à chaque signal sélectionné, et quels signaux restent inchangés

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

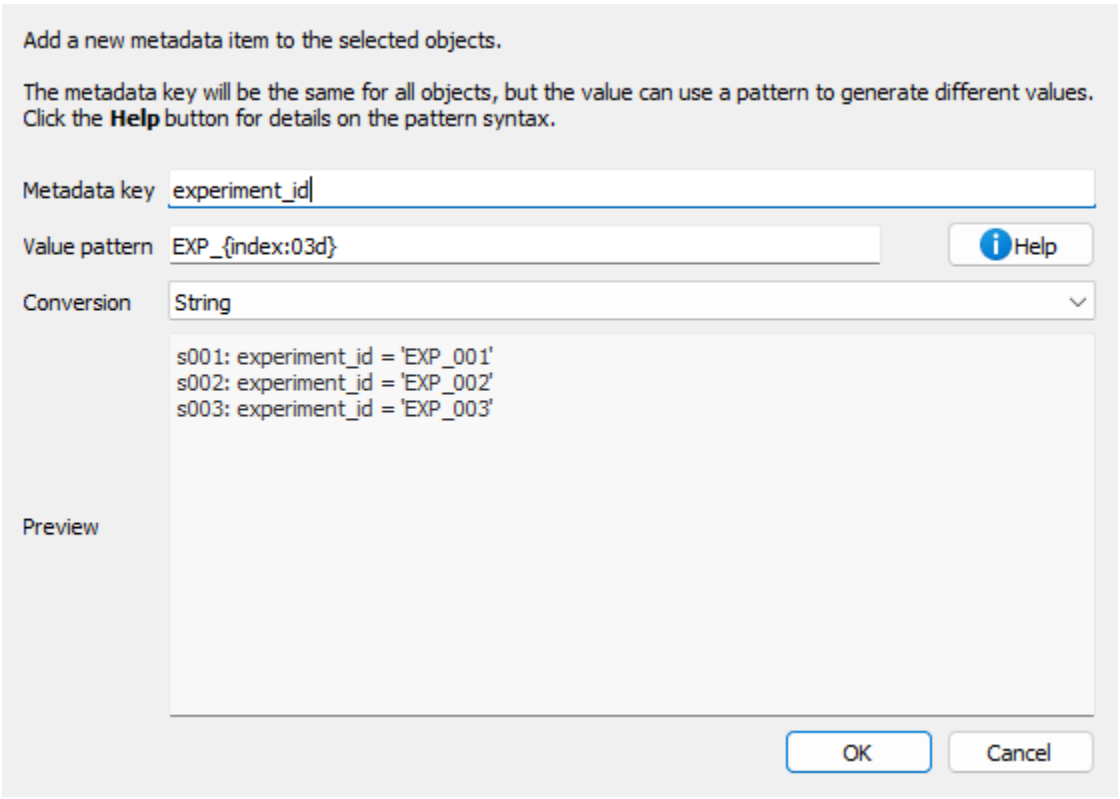


Fig. 23 – Boîte de dialogue Ajouter des métadonnées.

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
- Numéroter des tirs à partir de leurs titres : clé=``shot``, modèle=``{title}``, extraction=``shots*(d+)``, conversion=entier → Définit `shot=42` sur un signal intitulé « CH1 shot 042 »

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.

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 »).

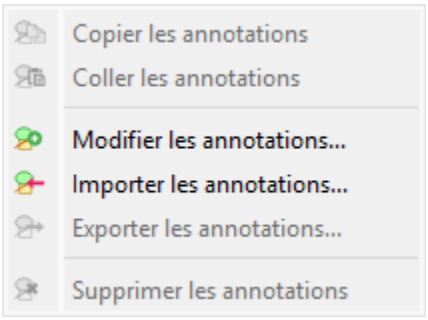


Fig. 24 – Capture d'écran du sous-menu « Annotations ».

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,σ=1,μ=0,y0=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)
```

3.2.4 Régions d'intérêt (ROI)

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

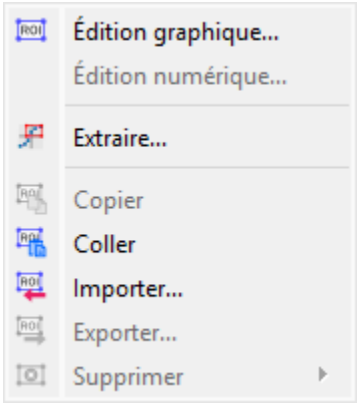


Fig. 25 – Capture d'écran du menu « ROI ».

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.

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

> **Note**
>
> Sur le graphique, chaque ROI est dessinée comme une zone translucide découpée le long de la courbe du signal (avec une ligne de base à `y = 0`, ou au bas de la zone de tracé sur un axe Y logarithmique), au lieu d'un rectangle occupant toute la hauteur. Les ROI successives utilisent des couleurs différentes, prises dans une palette cyclique de dix couleurs : la couleur est déduite de l'indice du titre `ROI<n>`, si bien qu'une ROI donnée conserve la même couleur à travers les suppressions, les réorganisations et les réouvertures de l'éditeur. Seules les coordonnées sont enregistrées : ces propriétés visuelles sont recalculées à chaque tracé des ROI.

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.

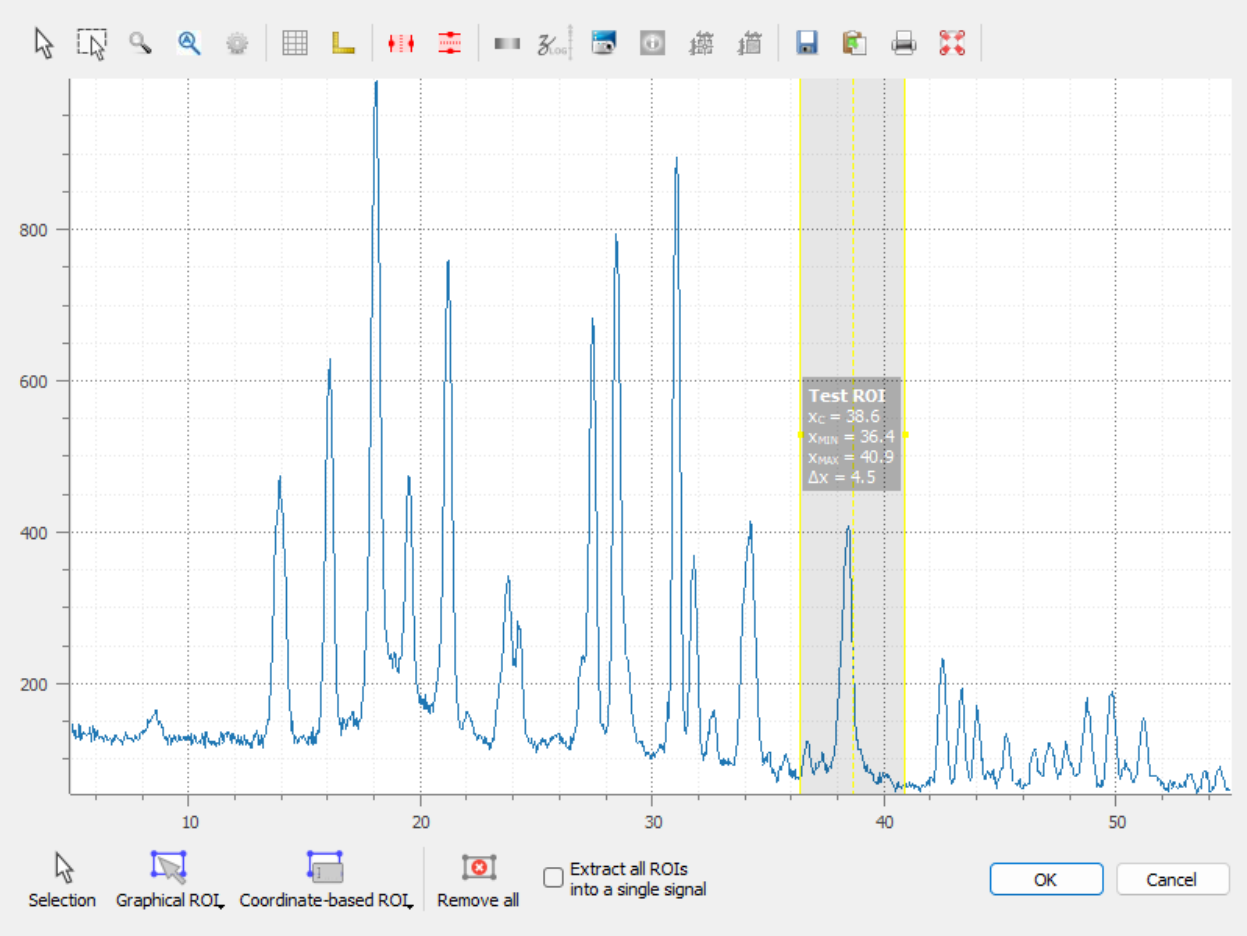


Fig. 26 – Un signal avec une ROI.

— « 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.

— « 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.

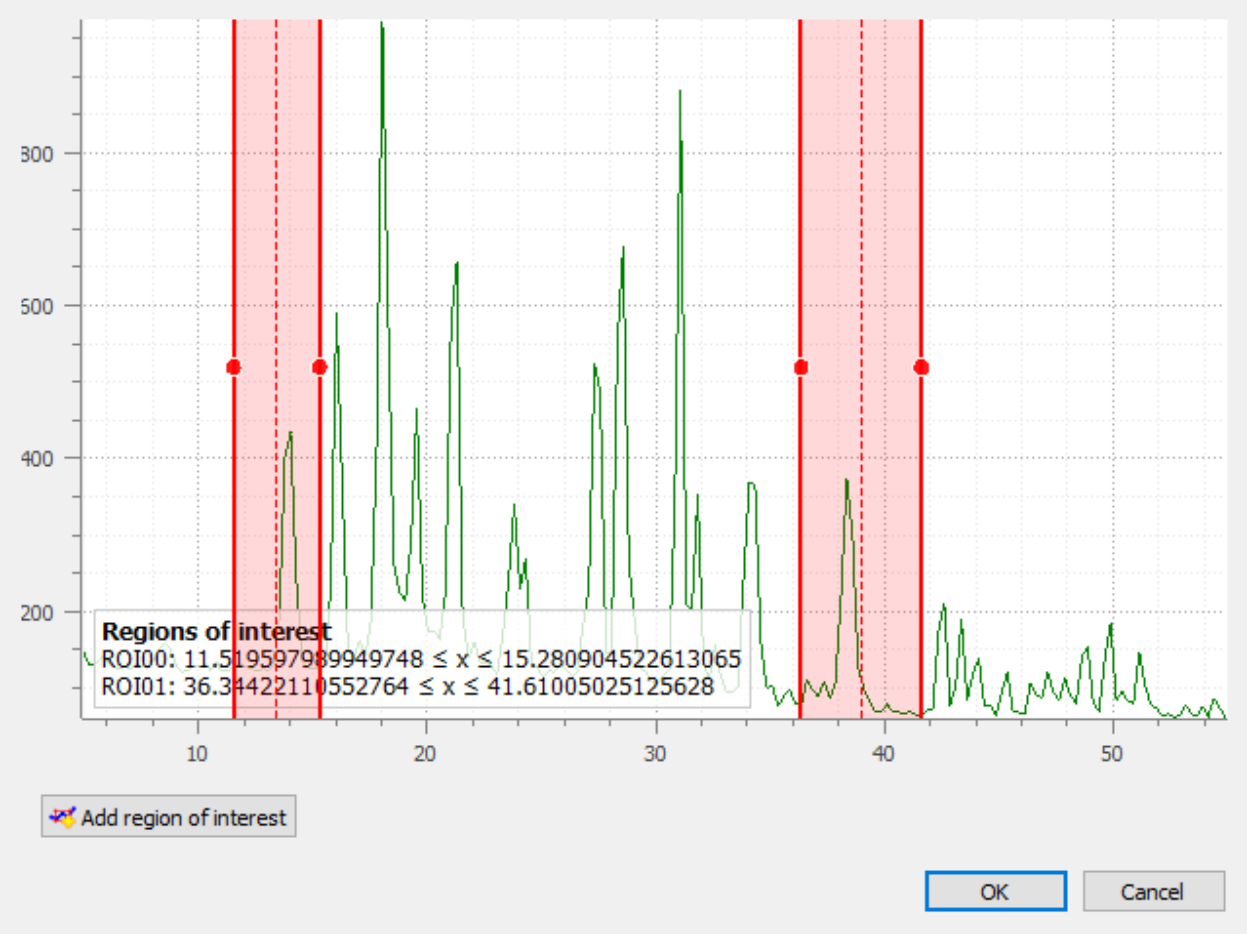


Fig. 27 – Boîte de dialogue d'extraction des ROI : la ROI est définie en déplaçant la position et en ajustant la largeur d'une plage horizontale.

3.2.5 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.

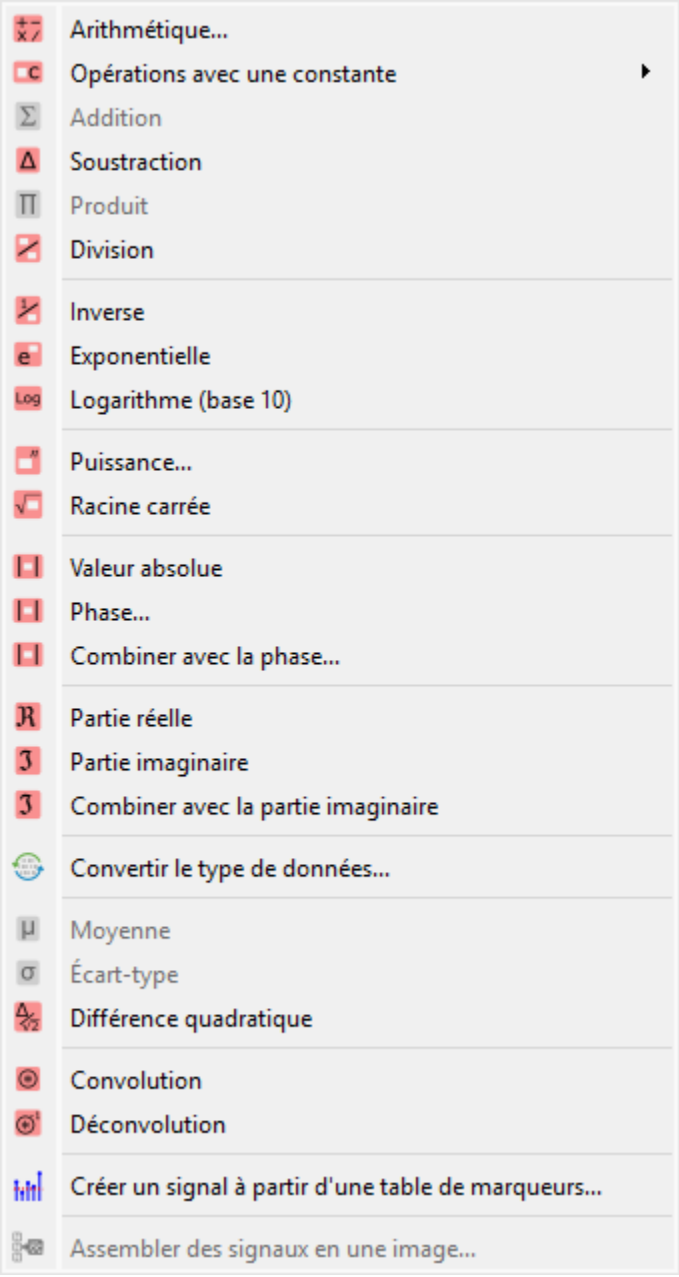


Fig. 28 – Capture d'écran du menu « Opérations ».

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

Arithmétique

Évalue une expression arithmétique définie par l'utilisateur sur les signaux sélectionnés. L'expression peut référencer les tableaux X et Y du signal comme variables.

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	$y_M = \sqrt{\frac{1}{M} \sum_{k=0}^{M-1} (y_k - \bar{y})^2}$
Différence quadratique	$y_2 = \sqrt{y_1^2 - y_0^2}$

Autres fonctions mathématiques

Fonction	Description
Puissance	$y_k = y_k^n$
Racine carrée	$y_k = \sqrt{y_k}$
Assembler les signaux en une image	Créer une image 2D en assemblant les signaux 1D sélectionnés en lignes ou colonnes, avec une normalisation optionnelle.
Créer un signal à partir d'une table de marqueurs…	Construit un signal de type bâtons à partir d'un résultat de type table de marqueurs XY attaché au signal sélectionné — typiquement produit par Analyse Extraire les positions des pics. Lorsque plusieurs tables de marqueurs XY sont stockées sur le signal, une boîte de dialogue permet de choisir la table source.

3.2.6 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.

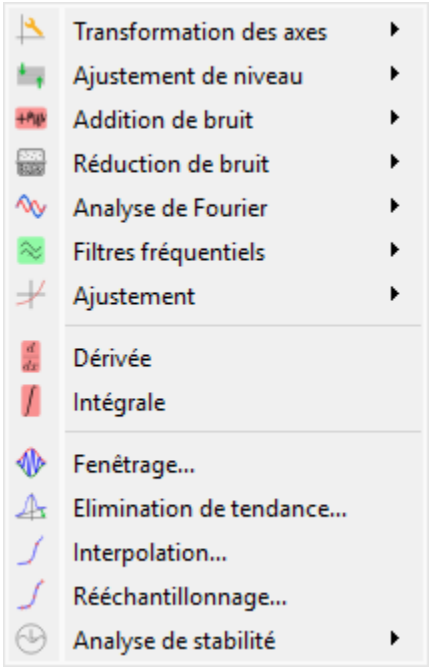


Fig. 29 – Capture d'écran du menu « Traitement ».

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.

Remplacer X par le Y d'un autre signal

Crée un nouveau signal en combinant deux signaux, en utilisant les valeurs Y du second signal comme coordonnées X pour les valeurs Y du premier signal.

Cette fonctionnalité est particulièrement utile pour les scénarios d'étalonnage, tels que :
- Appliquer un étalonnage en longueur d'onde aux données de spectroscopie
- Utiliser un signal de référence étalonné pour recalibrer les données de mesure
- Combiner des signaux où l'un contient des points d'étalonnage

Les deux signaux doivent avoir le même nombre de points. L'opération utilise directement les tableaux Y sans interpolation.

Entrée	Description
Signal 1	Fournit les données Y pour le signal de sortie
Signal 2	Fournit les données Y qui deviennent les coordonnées X du signal de sortie

Mode X-Y

Crée un nouveau signal en simulant le mode X-Y d'un oscilloscope.

Cette opération trace un signal en fonction d'un autre en :

1. Trouvant la plage X qui se chevauche entre les deux signaux
2. Rééchantillonnant les deux signaux sur une grille X commune dans ce chevauchement
3. Utilisant les valeurs Y rééchantillonnées comme coordonnées (Y du premier signal comme X, Y du second signal comme Y)

> **Note**
>
> Ceci est différent de « Remplacer X par le Y d'un autre signal » car il effectue une interpolation pour gérer les signaux avec des tableaux X différents.

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	$y_1 = \frac{y'_0}{\max(y'_0)}$ avec $y'_0 = y_0 - \min(y_0)$
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.

Remplacer les valeurs spéciales

Crée un nouveau signal où les échantillons `NaN`, `+Inf` et `-Inf` sont remplacés à l'aide d'une stratégie configurable pour chaque valeur cible.

Chacune des trois cibles (`NaN`, `+Inf`, `-Inf`) est traitée indépendamment. Pour chaque cible, l'utilisateur sélectionne l'une des stratégies suivantes (ou Ne rien faire pour laisser la cible inchangée) :

Stratégie	Description
Ne rien faire	Laisser les valeurs cibles inchangées.
Remplacer par zéro	Remplacer par `0`.
Remplacer par une constante	Remplacer par une valeur constante définie par l'utilisateur.
Remplacer par le minimum / maximum	Remplacer par le minimum / maximum des échantillons finis.
Remplacer par la moyenne / médiane	Remplacer par la moyenne / médiane des échantillons finis.
Supprimer les points	Supprimer les échantillons affectés du signal (le tableau X est également réduit). Un avertissement est émis si le signal était échantillonné uniformément.
Remplissage avant / remplissage arrière	Propager la dernière valeur valide (respectivement la suivante).
Interpolation : linéaire / spline / quadratique / cubique / PCHIP	Reconstruire les échantillons manquants en interpolant les échantillons valides environnants.
Minimum / maximum / moyenne / médiane sur N voisins	Remplacer par la statistique correspondante calculée sur les `2N+1` échantillons voisins autour de chaque position affectée.

Avant d'ouvrir le formulaire de paramètres standard, la boîte de dialogue affiche le nombre de chaque cible trouvée dans le signal (par exemple NaN : 17 (1,7 %)). Lorsqu'une stratégie fondée sur les voisins est sélectionnée, un petit aperçu montre le noyau qui sera utilisé.

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)	$y_1 = \lvert\text{FFT}\,(y_0)\rvert$ ou $y_1 = 20\log_{10}(\lvert\text{FFT}\,(y_0)\rvert)$ (dB)
Spectre de phase	Phase de la FFT exprimée en degrés, utilisant la fonction numpy.angle	$y_1 = \angle\,\text{FFT}\,(y_0)$
Densité spectrale de puissance (PSD)	Optionnel : sortie en décibels (dB). La PSD est estimée en utilisant la méthode de Welch (voir scipy.signal.welch)	$y_1 = \text{PSD}\,(y_0)$ ou $y_1 = 10\log_{10}(\text{PSD}\,(y_0))$ (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
 Passe-bas	Filtre les hautes fréquences, au-dessus d'une fréquence de coupure
 Passe-haut	Filtre les basses fréquences, en-dessous d'une fréquence de coupure
 Passe-bande	Filtre les fréquences en dehors d'une plage
 Coupe-bande	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
Tchebychev I	Filtre de Chebyshev de type I, utilisant la fonction scipy.signal.cheby1
Tchebychev II	Filtre de Chebyshev de type II, utilisant la fonction scipy.signal.cheby2
Elliptique	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, qui affiche la formule du modèle sous les paramètres d'ajustement.

Le signal ajusté enregistre les paramètres du modèle dans ses métadonnées (`metadata["fit_params"]`), sous les noms indiqués dans la dernière colonne du tableau ci-dessous :

Modèle	Equation	Paramètres enregistrés
Linéaire	$y = a \cdot x + b$	`a`, `b`
Polynomial	$y = a \cdot x^n + b \cdot x^{n-1} + c \cdot x^{n-2} + \ldots$	`a`, `b`, `c`, ...
Gaussien	$y = y_0 + A \exp\left(-\frac{1}{2}\left(\frac{x - x_0}{\sigma}\right)^2\right)$	A : `amplitude`, σ : `sigma`, x_0 : `x0`, y_0 : `y0`
Lorentzien	$y = y_0 + \frac{A}{1 + \left(\frac{x - x_0}{\sigma}\right)^2}$	A : `amplitude`, σ : `sigma`, x_0 : `x0`, y_0 : `y0`
Voigt	$y = y_0 + A \cdot \frac{\Re\left(w(z)\right)}{\Re\left(w(j/\sqrt{2})\right)}$ avec $w(z) = \exp(-z^2)\,\mathrm{erfc}(-jz)$ et $z = \frac{x - x_0 + j\sigma}{\sqrt{2}\sigma}$	A : `amplitude`, σ : `sigma`, x_0 : `x0`, y_0 : `y0`
Multi-gaussien	$y = y_0 + \sum_{i=1}^{N} A_i \exp\left(-\frac{1}{2}\left(\frac{x - x_{0,i}}{\sigma_i}\right)^2\right)$	A_i : `amplitude_i`, σ_i : `sigma_i`, $x_{0,i}$: `x0_i`, y_0 : `y0`
Multi-lorentzien	$y = y_0 + \sum_{i=1}^{N} \frac{A_i}{1 + \left(\frac{x - x_{0,i}}{\sigma_i}\right)^2}$	A_i : `amplitude_i`, σ_i : `sigma_i`, $x_{0,i}$: `x0_i`, y_0 : `y0`
Planck	$y = y_0 + A \cdot \left(\frac{x}{x_0}\right)^{-5} \cdot \left(\exp\left(\frac{5x_0}{\sigma \cdot x}\right) - 1\right)^{-1}$ où x_0 et σ sont des facteurs d'échelle et de température (voir la note ci-dessous)	A : `amp`, x_0 : `x0`, σ : `sigma`, y_0 : `y0`
Deux demi-gaussiennes	$y = y_{0,L} + A_L \cdot \exp\left(-\frac{1}{2}\left(\frac{x - x_0}{\sigma_L}\right)^2\right)$ si $x < x_0$ $y = y_{0,R} + A_R \cdot \exp\left(-\frac{1}{2}\left(\frac{x - x_0}{\sigma_R}\right)^2\right)$ sinon	A_L : `amp_left`, A_R : `amp_right`, σ_L : `sigma_left`, σ_R : `sigma_right`, x_0 : `x0`, $y_{0,L}$: `y0_left`, $y_{0,R}$: `y0_right`
Deux exponentielles dos à dos	$y = y_0 + a_L \cdot \exp\left(b_L \cdot x\right)$ si $x < x_c$ $y = y_0 + a_R \cdot \exp\left(b_R \cdot x\right)$ sinon	x_c : `x_center`, a_L : `a_left`, b_L : `b_left`, a_R : `a_right`, b_R : `b_right`, y_0 : `y0`
Exponentielle	$y = y_0 + a \exp\left(b \cdot x\right)$	`a`, `b`, y_0 : `y0`
Sinusoïdal	$y = y_0 + A \sin\left(2\pi f \cdot x + \phi\right)$	A : `amplitude`, f : `frequency`, ϕ : `phase` (en radians), y_0 : `offset`
Fonction de distribution cumulative (CDF)	$y = y_0 + A\,\mathrm{erf}\left(\frac{x - \mu}{\sqrt{2}\sigma}\right)$	A : `amplitude`, μ : `mu`, σ : `sigma`, y_0 : `baseline`
Sigmoïde	$y = y_0 + \frac{A}{1 + \exp\left(-k\left(x - x_0\right)\right)}$	A : `amplitude`, `k`, x_0 : `x0`, y_0 : `offset`

Pour les modèles gaussien, lorentzien et de Voigt, A est la hauteur signée du pic au-dessus de la ligne de base y_0, exprimée dans l'unité Y du signal.

Note

Le modèle de Planck est sur-paramétré : il est invariant par $x_0 \to kx_0$, $\sigma \to k\sigma$, $A \to A/k^5$. Seuls le rapport x_0/σ et le produit $A \cdot x_0^5$ sont déterminés par les données : les valeurs ajustées ne doivent donc pas être interprétées individuellement. En particulier, le pic d'émission se situe approximativement en $1.007\,x_0/\sigma$, et non en x_0.

Les grandeurs physiques se déduisent de ces combinaisons. Avec des longueurs d'onde en mètres, le modèle correspond à la loi de Planck (luminance énergétique spectrale par unité de longueur d'onde) pour :

— une température $T = \dfrac{hc\,\sigma}{5k_B\,x_0} \approx 2.878 \times 10^{-3}\,\dfrac{\sigma}{x_0}$ K,

— une émissivité (ou tout autre facteur d'échelle de la mesure) $\varepsilon = \dfrac{A\,x_0^5}{2hc^2}$, lorsque le signal est une luminance énergétique spectrale en W·sr^{-1}·m^{-3}.

Par exemple, l'ajustement du signal de corps noir créé à 293 K redonne $T = 293$ K et $\varepsilon = 1$.

Paramètres d'ajustement multi-pic

Pour les ajustements multi-gaussien et multi-lorentzien, la boîte de dialogue de détection de pics sélectionne d'abord le centre $x_{0,i}$ de chaque composante. Ces positions restent fixes : la boîte de dialogue d'ajustement interactif expose uniquement les paramètres `amplitude_i` et `sigma_i` de chaque composante ainsi que la ligne de base commune `y0`.

Lorsque l'ajustement est accepté, le signal généré stocke le modèle complet sous `metadata["fit_params"]`, y compris chaque centre fixe `x0_i`. Les métadonnées peuvent être consultées avec le bouton Métadonnées ou exportées vers un fichier JSON `.dlabmeta` à l'aide des actions d'importation et d'exportation des métadonnées.

Les paramètres exportés suffisent pour évaluer le modèle théorique sur n'importe quel axe X en dehors de DataLab :

```
import numpy as np
from sigima.tools.signal import fitting

fit_params = fitted_signal.metadata["fit_params"]
x = np.linspace(xmin, xmax, size)
y_theoretical = fitting.evaluate_fit(x, **fit_params)
```

Évaluer l'ajustement

Applique un ajustement précédemment calculé à un signal. Cette fonctionnalité utilise les paramètres d'ajustement stockés dans les métadonnées du signal (issus d'une opération d'ajustement antérieure) pour générer la courbe ajustée. Cela est utile pour évaluer l'ajustement sur une plage X différente ou sur un autre signal.

Dérivée

Crée un signal à partir de la dérivée de chaque signal sélectionné. Basé sur numpy.gradient.

Intégrale

Crée un signal à partir de l'intégrale de chaque signal sélectionné. Basé sur scipy.integrate.cumulative_trapezoid.

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()`
À sommet plat	`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.
Variance 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.

3.2.7 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.

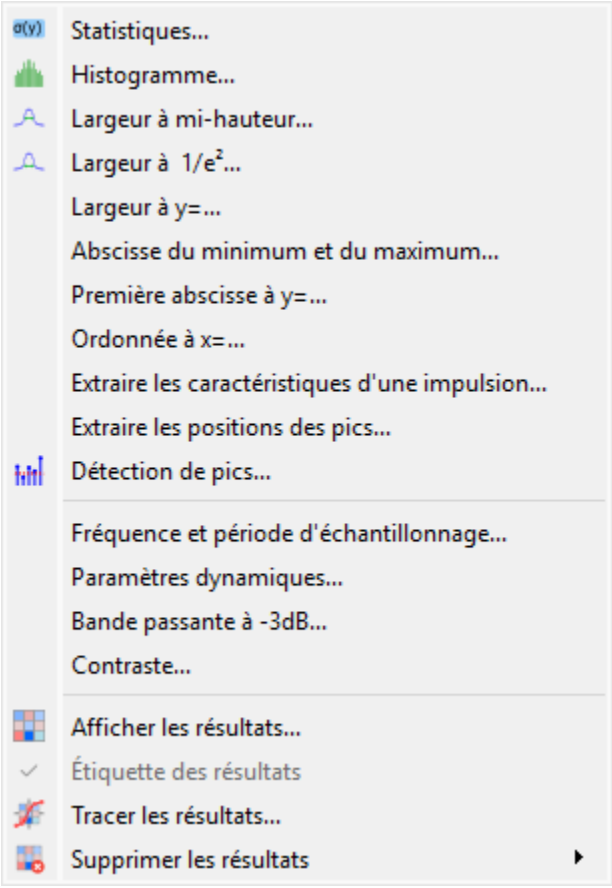


Fig. 30 – Capture d'écran du menu Analyse ».

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.

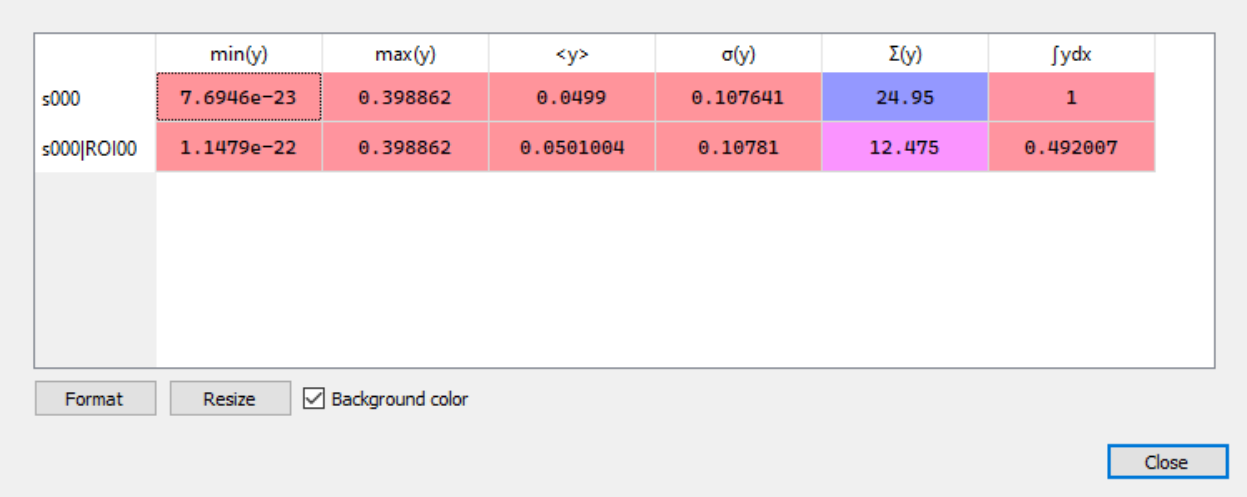


Fig. 31 – Exemple de tableau récapitulatif de statistiques : chaque ligne est associée à une ROI (à l'exception de la première qui correspond aux statistiques calculées sur la totalité des données).

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

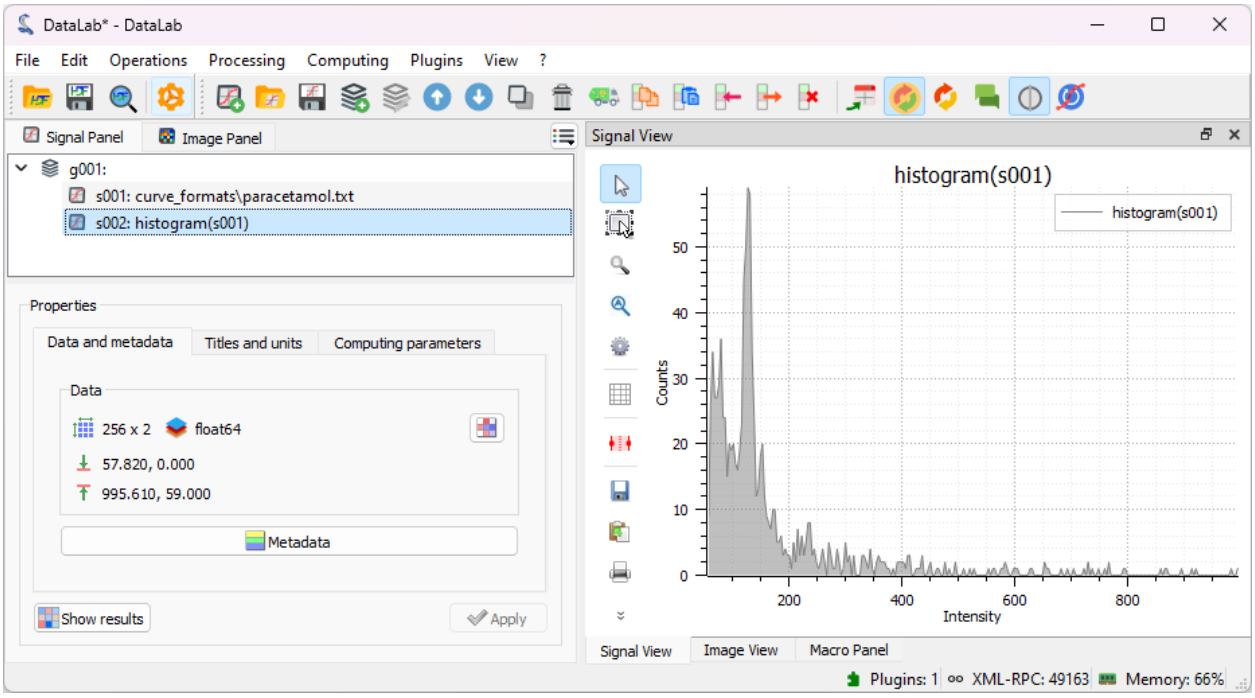


Fig. 32 – Exemple d'histogramme.

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

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.

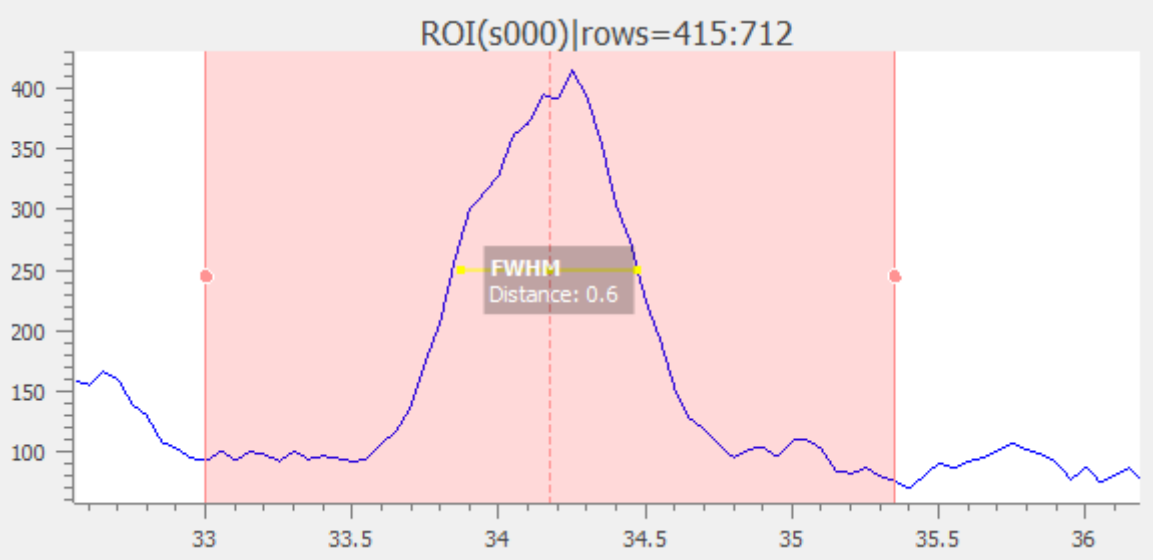


Fig. 33 – Le résultat du calcul est affiché sous la forme d'un segment annoté.

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

DataLab propose deux entrées complémentaires pour détecter les pics. Les deux produisent le même résultat : un `TableResult` de type marqueurs XY attaché au signal (visible depuis Vue Afficher les résultats et superposé à la courbe).

Entrée	Description
Extraire les positions des pics	Entrée paramétrique : remplissez la boîte de dialogue standard (seuil, distance minimale) et exécutez.
Détection de pics…	Entrée interactive : ouvre une boîte de dialogue dédiée dans laquelle le seuil est ajusté en déplaçant un curseur horizontal et la distance minimale via un slider. Les pics détectés sont mis à jour en temps réel sous forme de marqueurs verticaux.

> **Astuce**
>
> Pour reconstruire l'ancien signal de type bâtons à partir des pics détectés, utilisez Opérations Créer un signal à partir d'une table de marqueurs… sur le signal sélectionné.

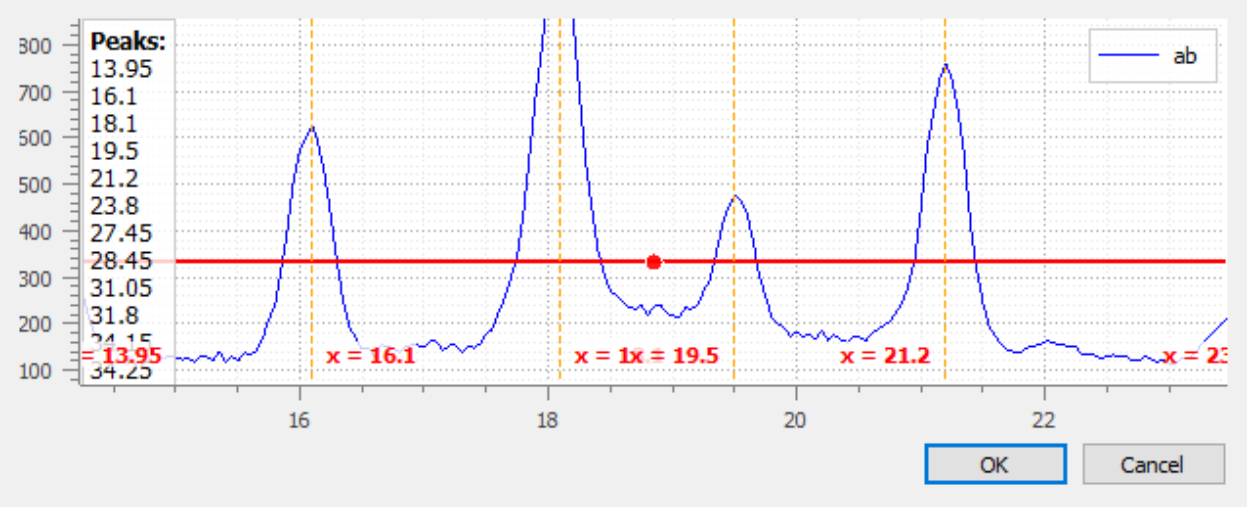


Fig. 34 – Boîte de dialogue Détection de pics : le seuil est ajustable en déplaçant le curseur horizontal, les pics sont détectés automatiquement (des marqueurs verticaux indiquent les pics détectés avec leur position).

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 :

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

> **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.

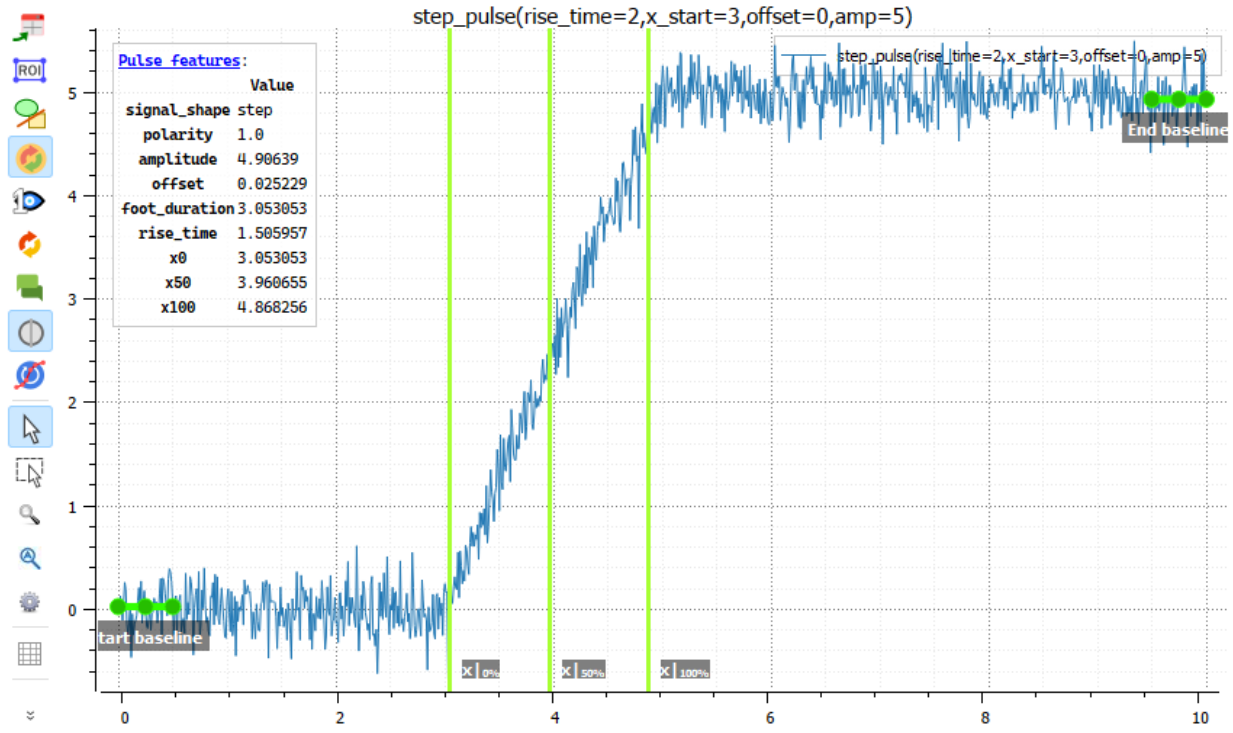


Fig. 35 – Résultats des caractéristiques d'impulsions tels qu'affichés dans la vue Signal.

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).

3.2.8 Options d'affichage pour les signaux

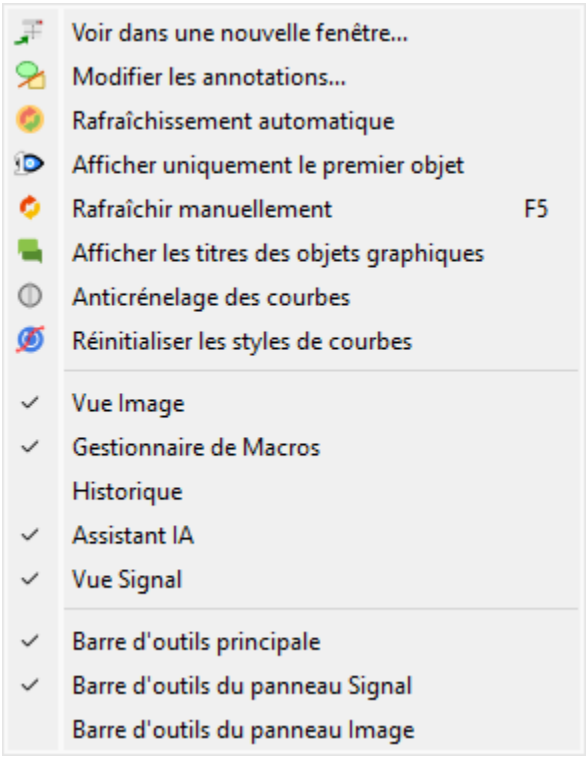


Fig. 36 – Capture d'écran du menu « Affichage ».

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.

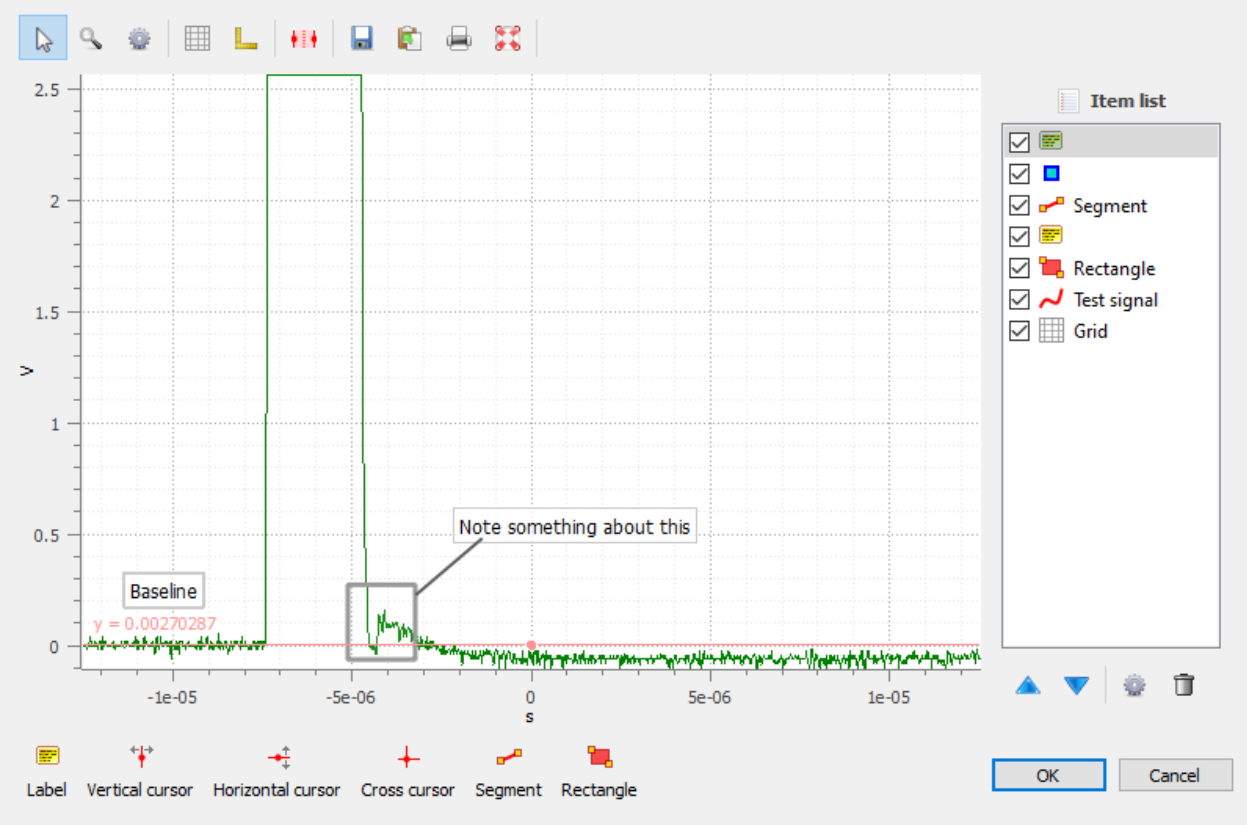


Fig. 37 – Les annotations peuvent être ajoutées dans la vue séparée.

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.

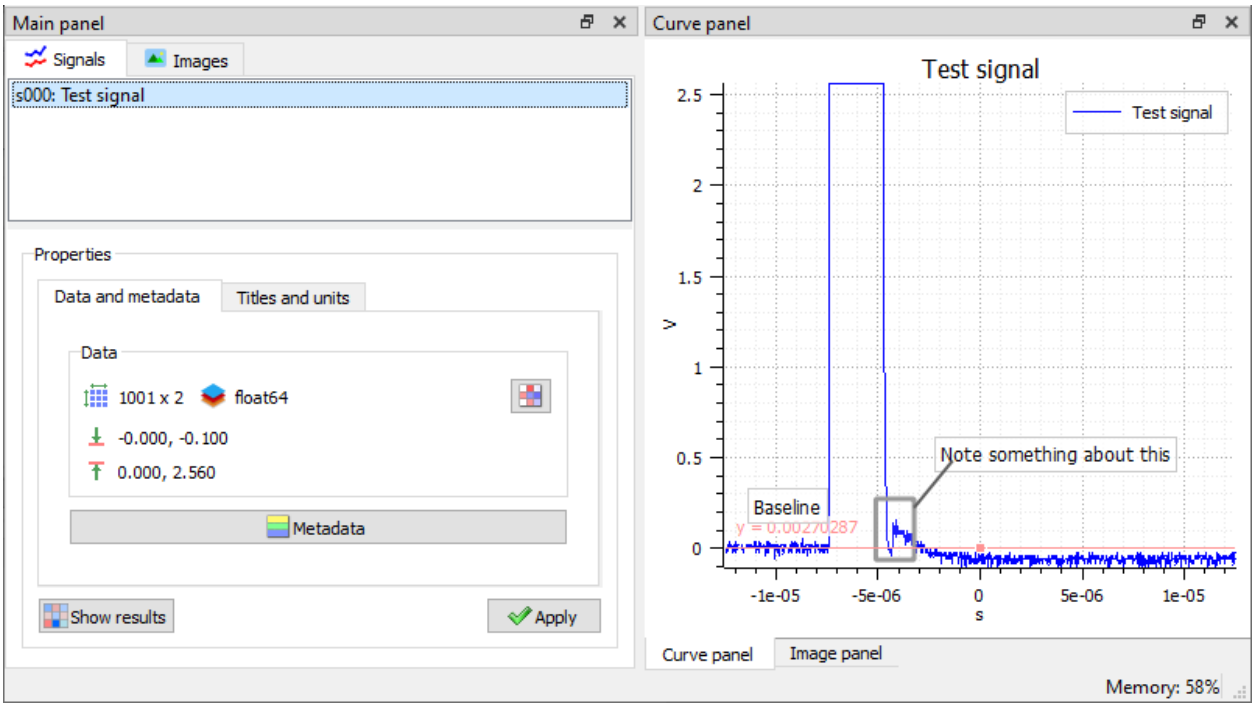


Fig. 38 – Les annotations font désormais partie des métadonnées du signal.

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

Rend aux signaux sélectionnés des styles de courbes automatiques.

Lors de l'affichage de courbes, DataLab attribue automatiquement une couleur et un style de trait à chaque courbe, selon sa position parmi les courbes affichées dans le graphique (dans l'ordre des signaux du panneau) : les premières courbes sont tracées en traits pleins de couleurs bien distinctes, lisibles sur fond clair comme sur fond sombre, et les suivantes en tirets avec les mêmes couleurs. La première courbe affichée reçoit donc toujours la première couleur, quels que soient les signaux affichés.

Les styles automatiques ne sont pas enregistrés avec les signaux. Si vous choisissez une autre couleur ou un autre style de trait pour une courbe (avec l'option « Paramètres » de son menu contextuel), ce style est enregistré dans les métadonnées du signal et il est conservé par la suite.

Cette entrée de menu supprime les couleurs et les styles de trait choisis pour les signaux sélectionnés, afin que leurs courbes reçoivent à nouveau des styles automatiques.

Note

Les espaces de travail et les fichiers d'historique enregistrés avec DataLab 1.3 ou une version antérieure contiennent le style automatique de chaque courbe : à leur ouverture, ces styles sont supprimés des métadonnées des signaux, dont les courbes reçoivent les styles automatiques actuels. Une couleur et un style de trait choisis à l'époque ne peuvent pas être distingués d'un style automatique lorsqu'ils lui sont identiques : ils sont également supprimés. Les fichiers enregistrés avec les versions ultérieures, ainsi que les métadonnées importées depuis un fichier, ne sont pas convertis.

3.3 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.

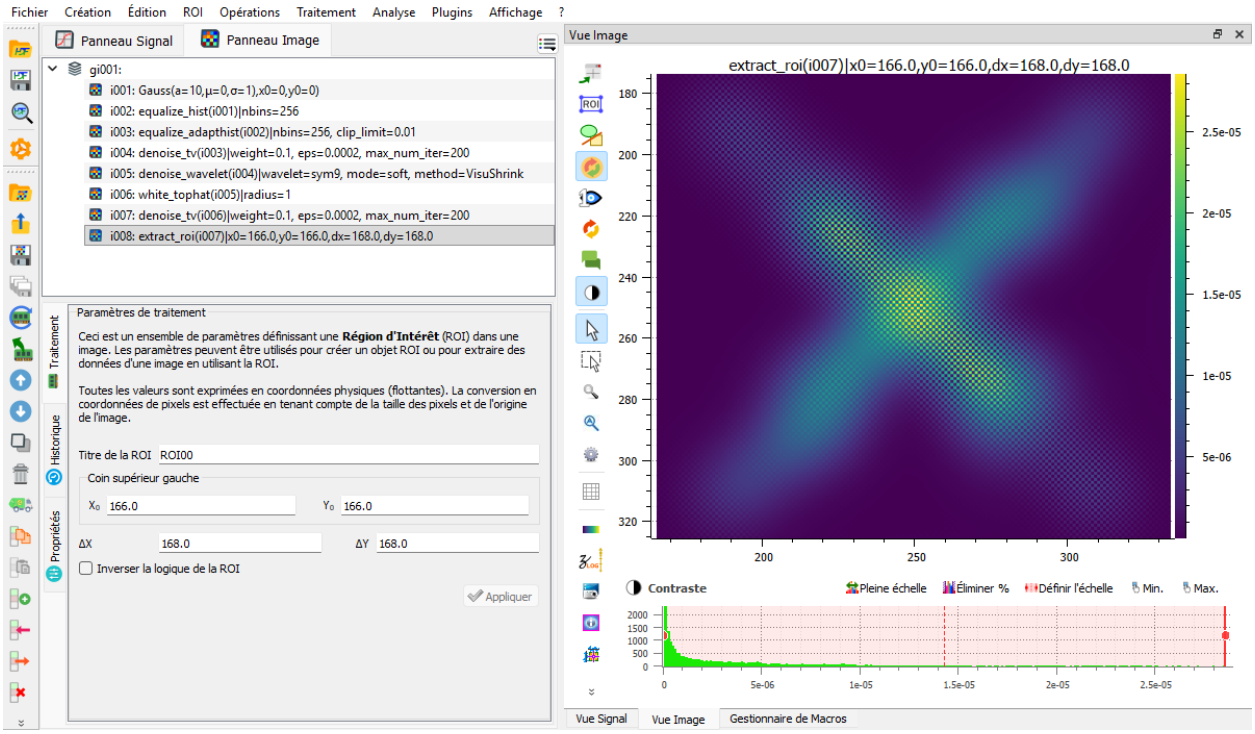


Fig. 39 – Fenêtre principale de DataLab : vue du traitement d'image

3.3.1 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).
- Exporter, inspecter, rejouer et comparer des capsules d'espace de travail, c'est-à-dire un espace de travail avec la provenance de ses traitements (voir Capsules d'espace de travail).
- Modifier les préférences de DataLab (voir Préférences).

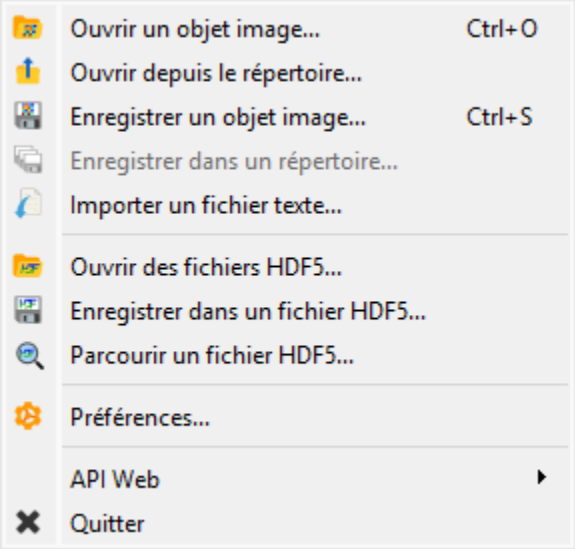


Fig. 40 – Capture d'écran du menu « Fichier ».

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
Fichiers RAW sans en-tête	.raw

À l'ouverture d'un fichier RAW sans en-tête (`.raw`), DataLab affiche une boîte de dialogue Import d'image RAW permettant de décrire sa structure binaire : type de données des pixels, largeur et hauteur de l'image, décalage avant la première image, nombre d'images, espacement entre images consécutives, ordre des octets, ainsi que l'option Blanc à zéro (qui affiche l'image importée avec une palette de niveaux de gris inversée sans modifier les valeurs des pixels). Les fichiers multi-images créent une image par trame. Lorsque plusieurs fichiers RAW sont ouverts simultanément, DataLab demande s'il faut appliquer les mêmes paramètres d'import à tous les fichiers ou définir les paramètres de chaque fichier individuellement.

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'))`**

Séquence immuable intégrée.

Si aucun argument n'est fourni, le constructeur renvoie un tuple vide. Si iterable est fourni, le tuple est initialisé à partir des éléments de cet itérable.

Si l'argument est un tuple, la valeur renvoyée est ce même objet.

Ouvrir depuis un répertoire

Ouvrir plusieurs images depuis un répertoire spécifié.

Enregistrer l'image

Enregistre l'image courante (voir les types de fichiers pris en charge dans « Ouvrir une image » ; l'écriture de fichiers `.raw` n'est pas prise en charge).

Après le choix du fichier de destination, DataLab ouvre une boîte de dialogue Exporter l'image adaptée au format de fichier sélectionné, avec un aperçu en direct de l'image exportée (affichée en niveaux de gris telle qu'elle sera relue depuis le fichier) :

- Préparation des données : mode de normalisation (aucune, min-max, percentiles ou bornes manuelles), comportement hors plage (écrêtage ou rééchelonnage), traitement des valeurs non finies (erreur, écrêtage ou remplacement), correction gamma, inversion et type de données cible parmi ceux pris en charge par le format de destination. Lorsqu'un format ne prend en charge que des pixels entiers (par exemple PNG, JPEG ou BMP) et que l'image source utilise un autre type de données, le rééchelonnage min-max est sélectionné par défaut.
- Options propres au format : compression, qualité, prédicteur, tuilage, résolution, interprétation photométrique, séparateur et précision des fichiers texte, ou compression MAT, selon le format choisi.

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.

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

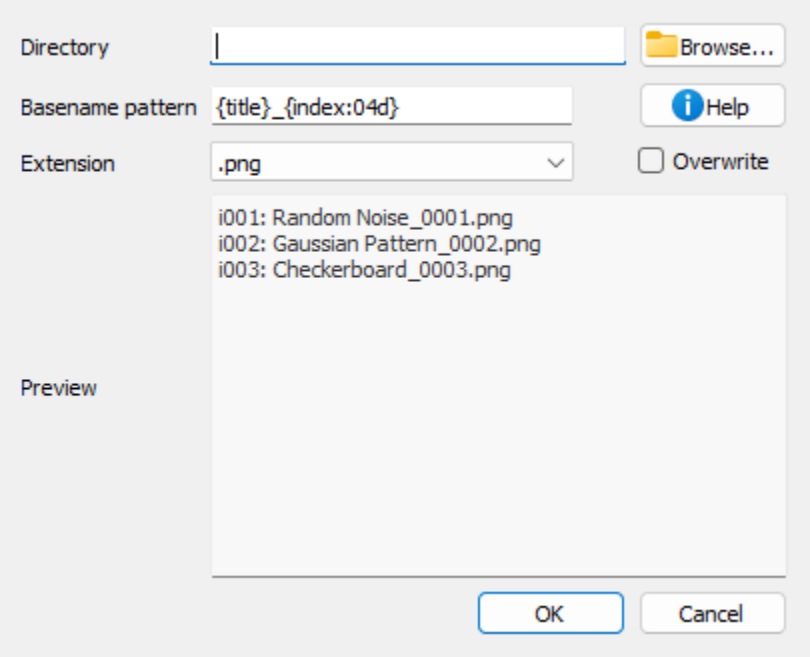


Fig. 41 – Boîte de dialogue Enregistrer les images dans un répertoire.

— `{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.

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.

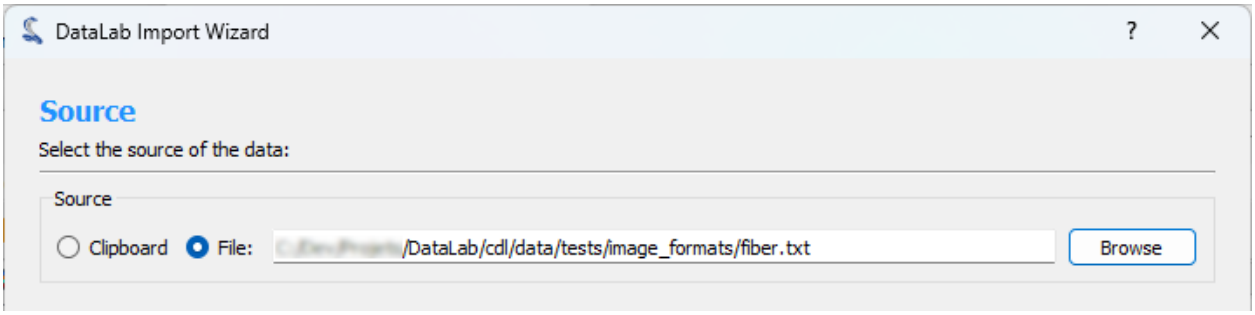


Fig. 42 – Etape 1 : Sélectionner la source

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

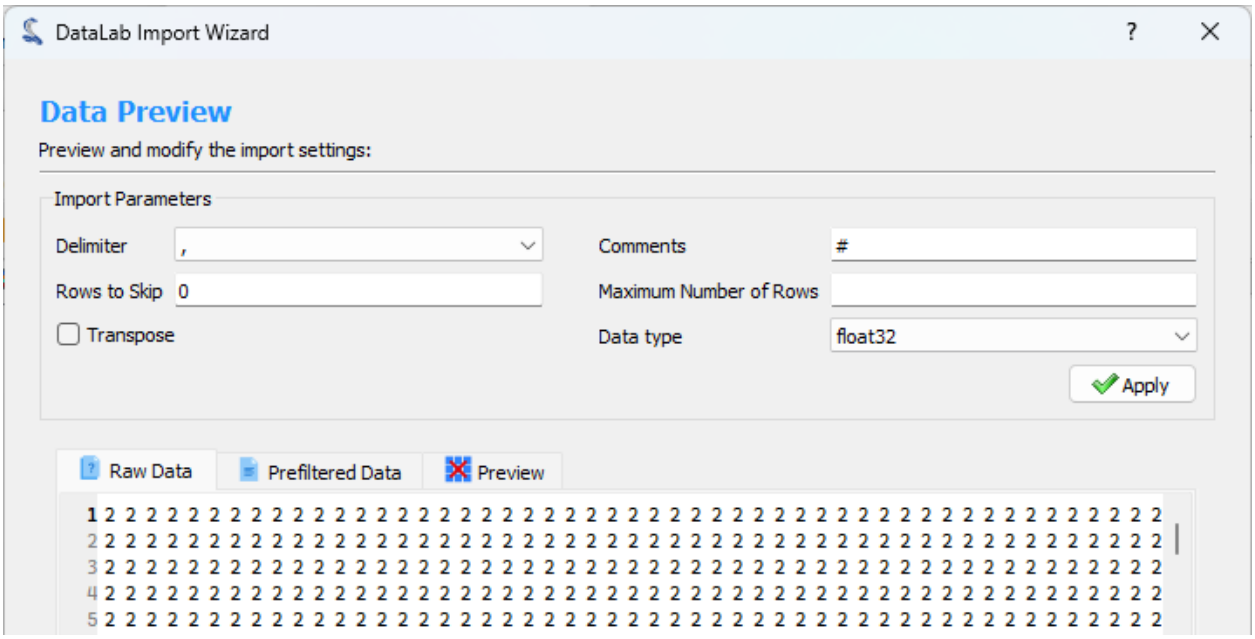


Fig. 43 – Etape 2 : Configurer l'importation

Etape 3 : Afficher la représentation graphique

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

3.3.2 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).

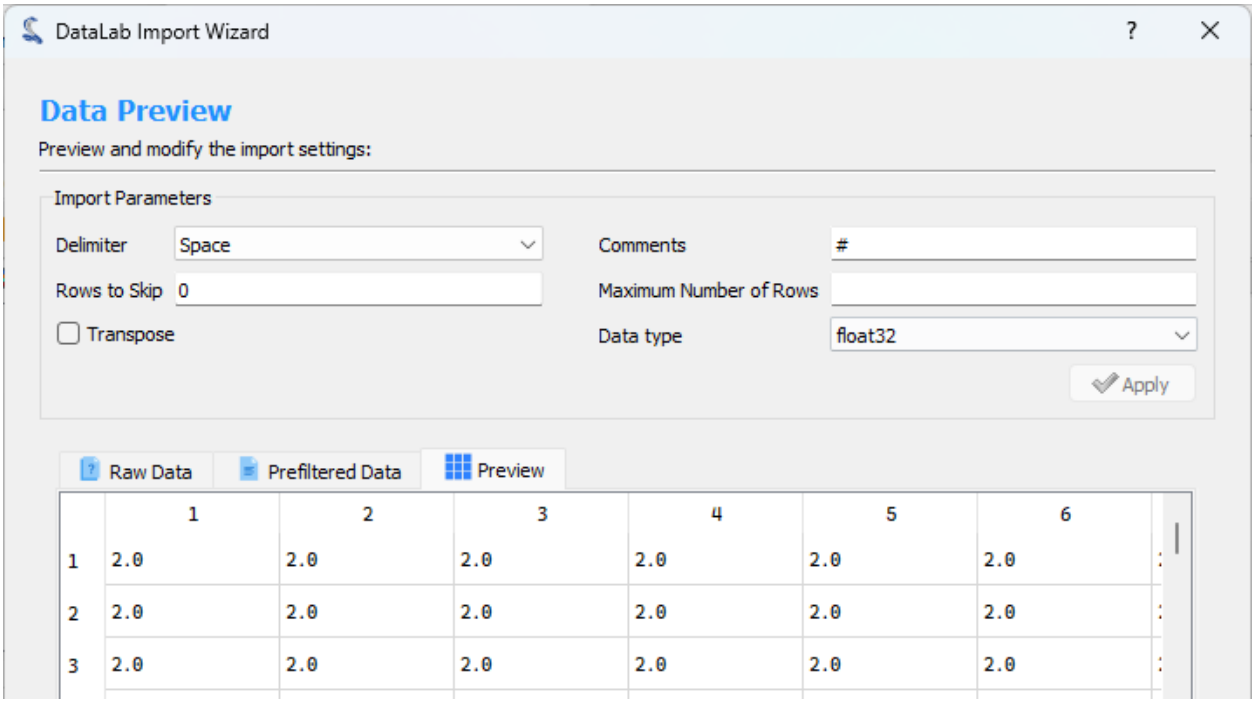


Fig. 44 – Etape 2 : Prévisualiser le résultat

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

suite sur la page suivante

Tableau 3 – suite de la page précédente

Icône	Modèle	Équation
	Étoile de Siemens	Motif de rayons radiaux pour tester la résolution
	Sinc 2D	$z = A \cdot \mathrm{sinc}\left(\frac{\sqrt{(x-x_0)^2+(y-y_0)^2}}{\sigma}\right) + C$

Les modèles gaussienne 2D, rampe 2D, réseau sinusoïdal, anneaux, mire de Siemens et sinus cardinal 2D sont calculés sur un domaine $[x_{min}, x_{max}] \times [y_{min}, y_{max}]$. Lorsque l'option Coordonnées physiques est activée (par défaut), ce domaine définit les coordonnées de l'image : les axes représentent x et y tels qu'utilisés dans l'équation, de sorte que les centres, largeurs et fréquences spatiales se lisent directement sur l'image (par exemple, un réseau sinusoïdal de fréquence $f_x = 0.1$ présente une période toutes les 10 unités le long de l'axe X). Lorsque l'option est désactivée, l'image est affichée en coordonnées pixel.

> **Note**
>
> Les images créées avec DataLab 1.3 ou une version antérieure étaient affichées en coordonnées pixel : elles conservent ces coordonnées lorsque leurs paramètres de création sont modifiés ou recalculés.

3.3.3 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.).

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.

Cette action actualise également les résultats d'analyse (statistiques, FWHM, centroïde, détection de pics, de contours ou de blobs, etc.) à partir de leurs paramètres enregistrés. Les résultats d'analyse ne sont pas recalculés automatiquement lorsque vous modifiez une région d'intérêt, les données ou les propriétés de l'objet : les résultats existants restent inchangés jusqu'à ce que vous déclenchiez explicitement cette action, vous laissant ainsi un contrôle total sur le moment où les analyses sont actualisées.

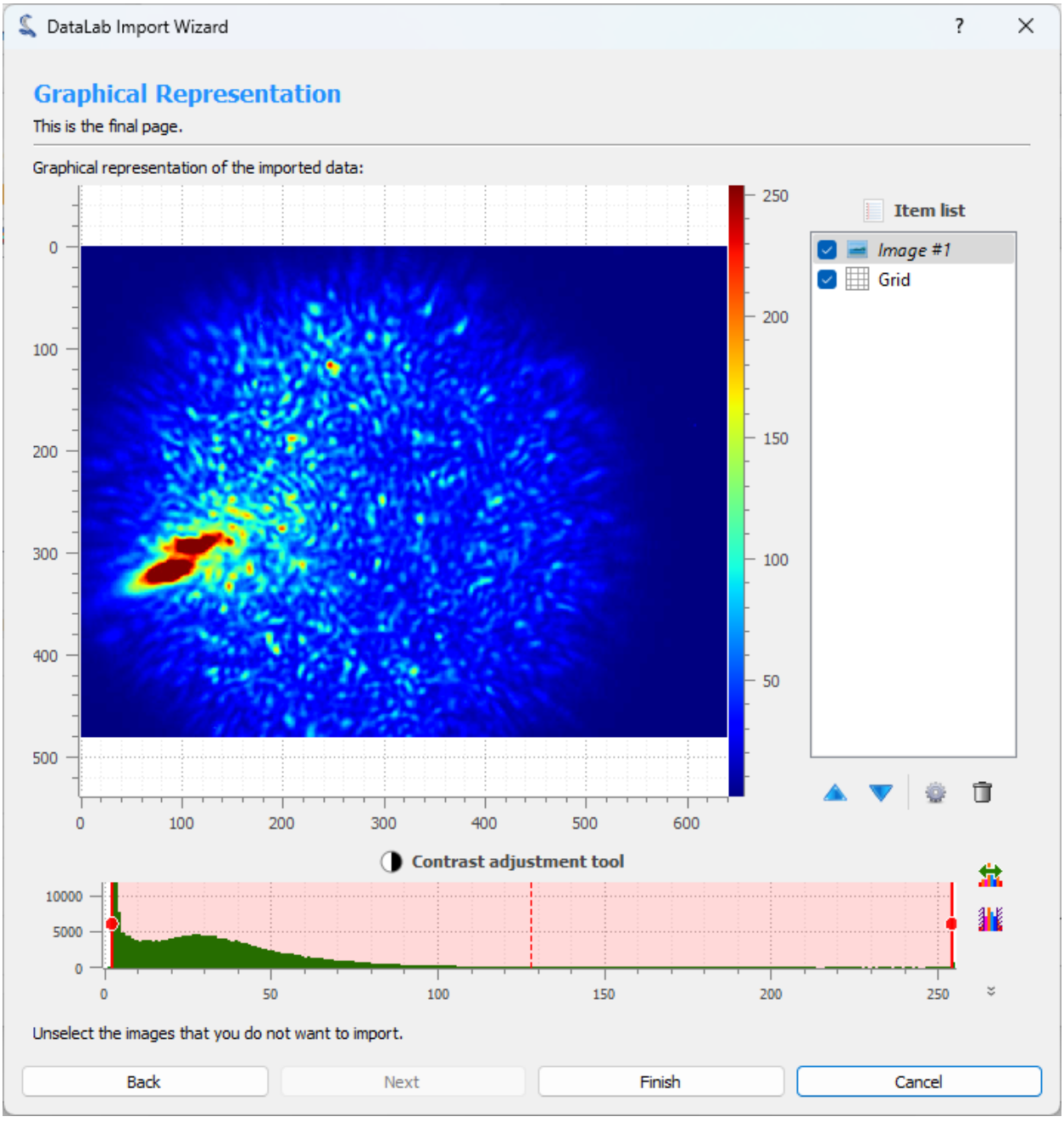


Fig. 45 – Etape 3 : Afficher la représentation graphique

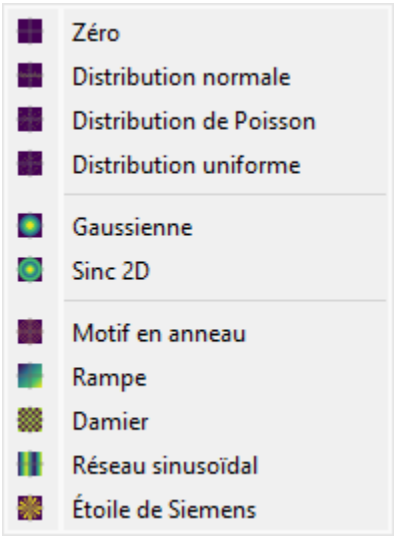


Fig. 46 – Capture d'écran du menu « Création ».

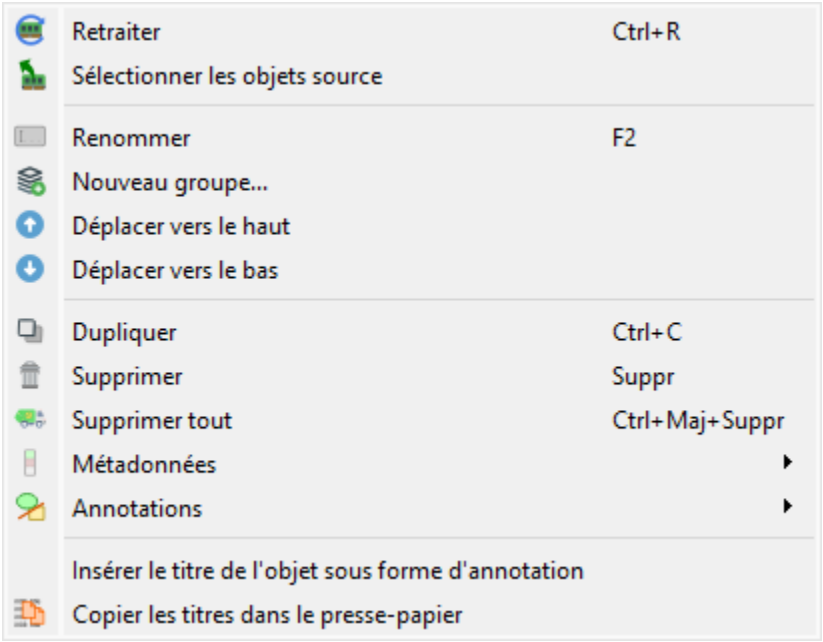


Fig. 47 – Capture d'écran du menu « Édition ».

> **Note**
>
> Cette action n'est disponible que pour les images disposant de paramètres de traitement enregistrés (images créées par des opérations de traitement) ou de paramètres d'analyse enregistrés (opérations d'analyse exécutées précédemment sur celles-ci).

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

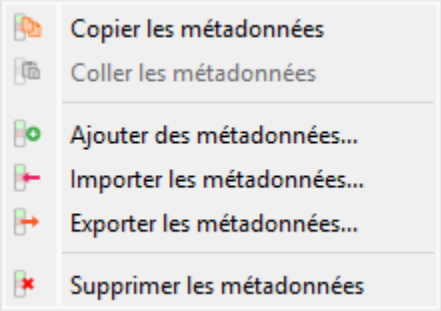


Fig. 48 – Capture d'écran du sous-menu « Métadonnées ».

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.

Lorsque vous collez des métadonnées, une boîte de dialogue vous permet de sélectionner les catégories de métadonnées à conserver à partir du presse-papiers :

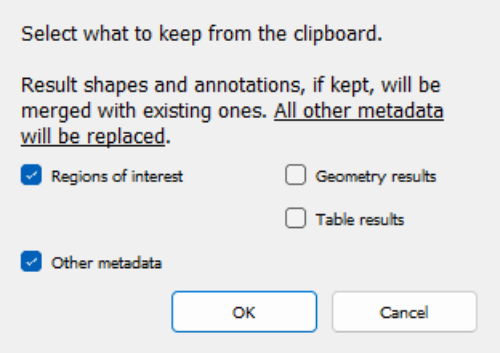


Fig. 49 – Capture d'écran de la boîte de dialogue « Coller les métadonnées ».

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 métadonnée : Saisir le nom du champ de métadonnée à ajouter (lettres, chiffres, _, . et -, par exemple une clé de plugin telle que `plugin.org.example.my-plugin.gain`)
- Clés connues : Choisir une clé déjà présente sur les images sélectionnées, ou une clé attendue par les méthodes des plugins d'application installés, pour la copier dans la clé de métadonnée
- Modèle de valeur : Définir un modèle pour la valeur des métadonnées en utilisant des chaînes de format Python
- Motif d'extraction : Facultativement, extraire la valeur du texte formaté avec une expression régulière Python : son premier groupe, ou toute la correspondance si elle n'a pas de groupe, devient la valeur

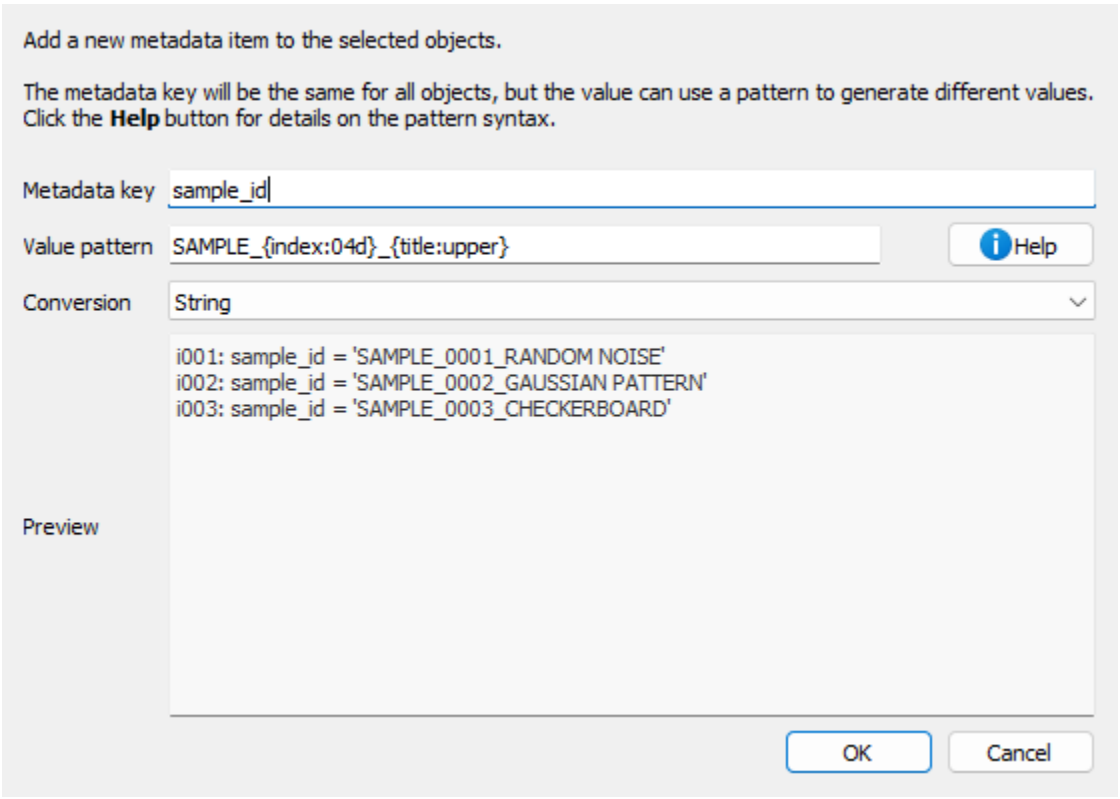


Fig. 50 – Boîte de dialogue Ajouter des métadonnées.

- Sans correspondance : Laisser inchangées les images sans correspondance (par défaut), ou signaler une erreur
- Conversion : Choisissez comment stocker la valeur (chaîne, flottant, entier ou booléen)
- Facteur d'échelle : Multiplier les valeurs numériques, par exemple `0.001` pour convertir des millisecondes en secondes
- Aperçu : Voir comment les métadonnées seront ajoutées à chaque image sélectionnée, et quelles images restent inchangées

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 :

- `{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.
- Lire les temps d'exposition dans les titres : clé=``exposure_time_s``, modèle=``{title}``, extraction=``([d.]+)s*ms``, conversion=flottant, facteur d'échelle=``0.001`` → Définit `exposure_time_s=0.005` sur une image intitulée «

Flat 5 ms 01 » et laisse « Dark 01 » inchangée

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.

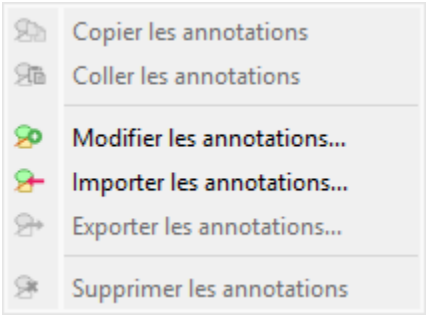


Fig. 51 – Capture d'écran du sous-menu « Annotations ».

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 »).

Modifier les annotations

L'action « Modifier 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,σ=1,μ=0,y0=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)
```

3.3.4 Régions d'intérêt (ROI)

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

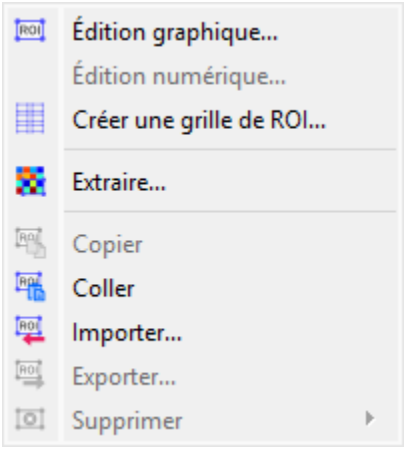


Fig. 52 – Capture d'écran du menu « ROI ».

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).
— « 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.

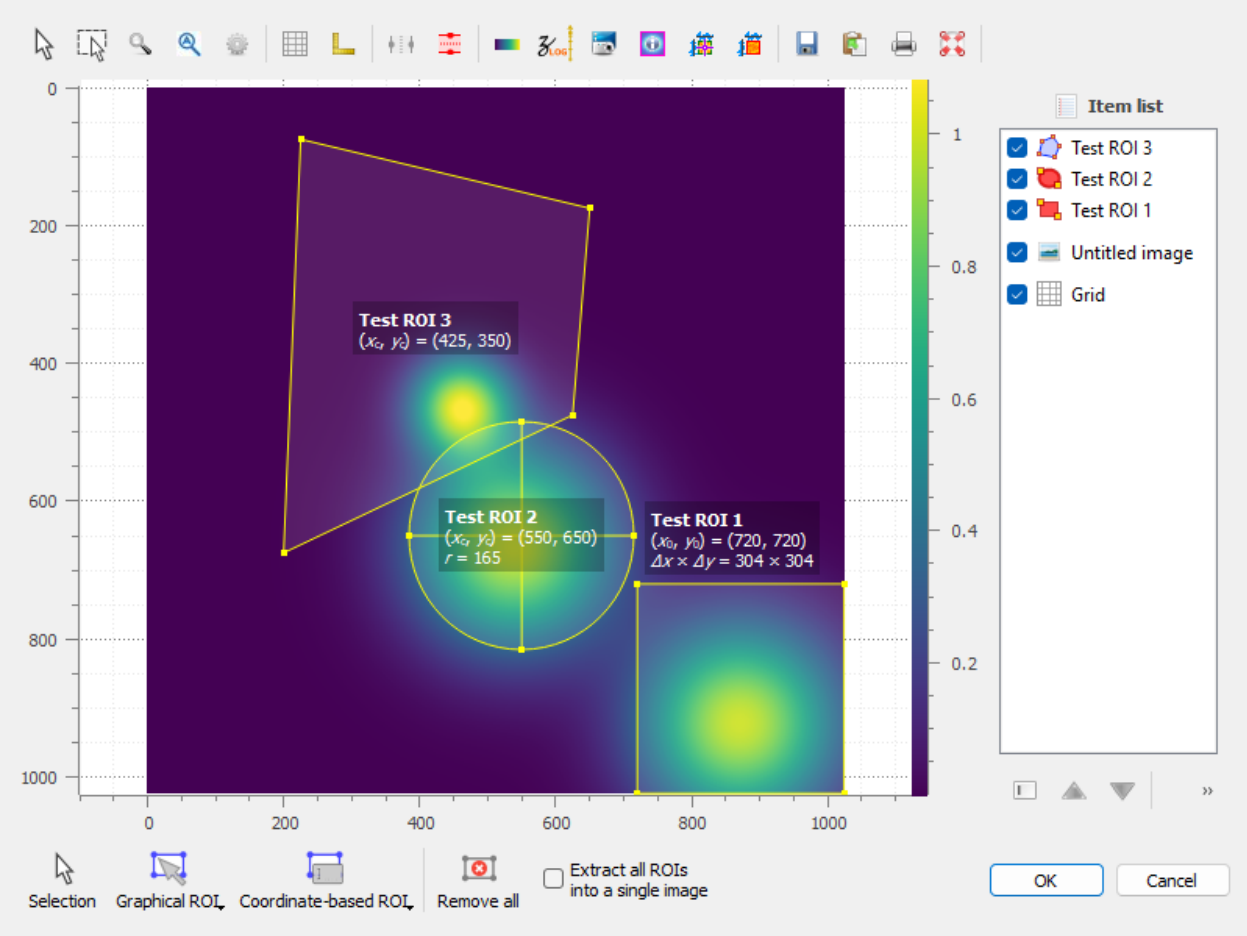


Fig. 53 – Une image avec une ROI.

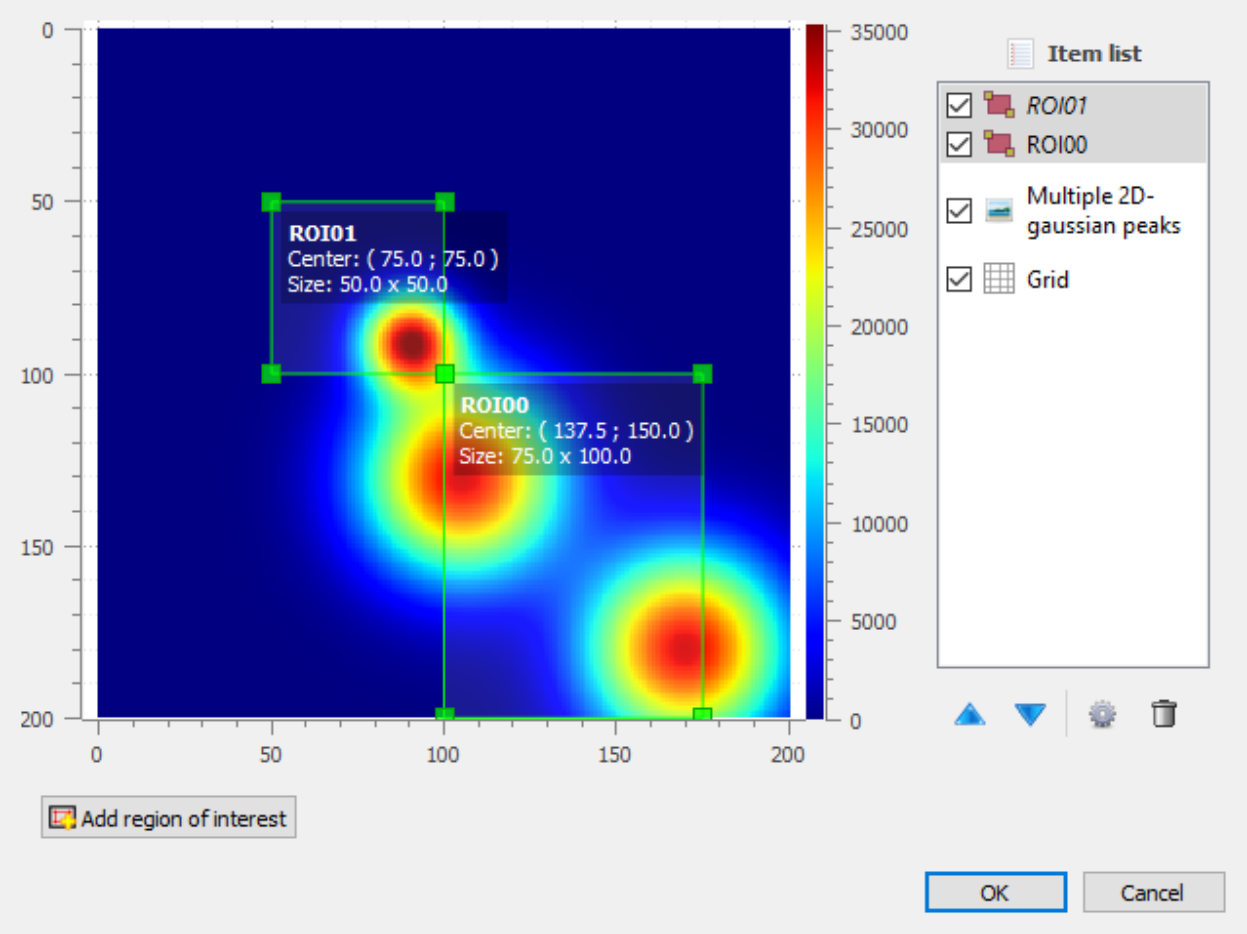


Fig. 54 – Boîte de dialogue d'extraction des ROI : la ROI est définie en déplaçant la position et en ajustant la taille d'une forme rectangulaire.

3.3.5 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.

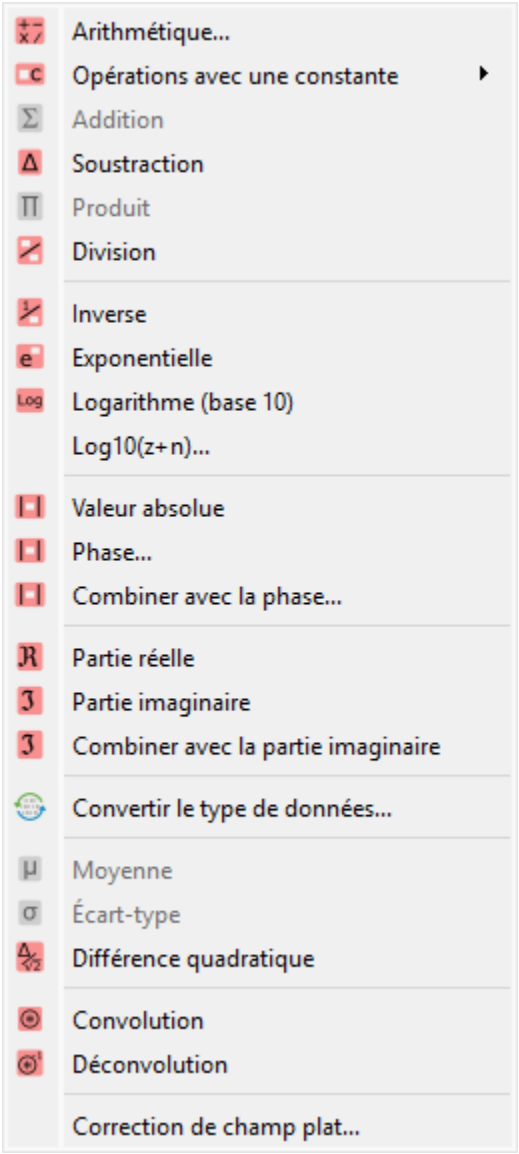


Fig. 55 – Capture d'écran du menu « Opérations ».

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
+c Addition	$z_k = z_{k-1} + conv(c)$
-c Soustraction	$z_k = z_{k-1} - conv(c)$
×c Multiplication	$z_k = conv(z_{k-1} \times c)$
/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
Σ Somme	$z_M = \sum_{k=0}^{M-1} z_k$
Δ Soustraction	$z_2 = z_1 - z_0$
∏ Produit	$z_M = \prod_{k=0}^{M-1} z_k$
⁄ Division	$z_2 = \frac{z_1}{z_0}$
1⁄ Inverse	$z_2 = \frac{1}{z_1}$
e Exponentielle	$z_2 = \exp(z_1)$
Log Logarithme (base 10)	$z_2 = \log_{10}(z_1)$

Fonctions mathématiques de base

Fonction	Description
e Exponentielle	$z_k = \exp(z_k)$
Log 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	$z_k = \|z_{k-1}\|$
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
μ Moyenne	$z_M = \frac{1}{M}\sum_{k=0}^{M-1} z_k$
σ Écart-type	$z_M = \sqrt{\frac{1}{M}\sum_{k=0}^{M-1} (y_k - \bar{y})^2}$
Δ/√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 :

$$z_1 = \begin{cases} \frac{z_0}{z_f}.\overline{z_f} & \text{si } z_0 > z_{threshold} \\ z_0 & \text{sinon} \end{cases}$$

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

3.3.6 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.

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.

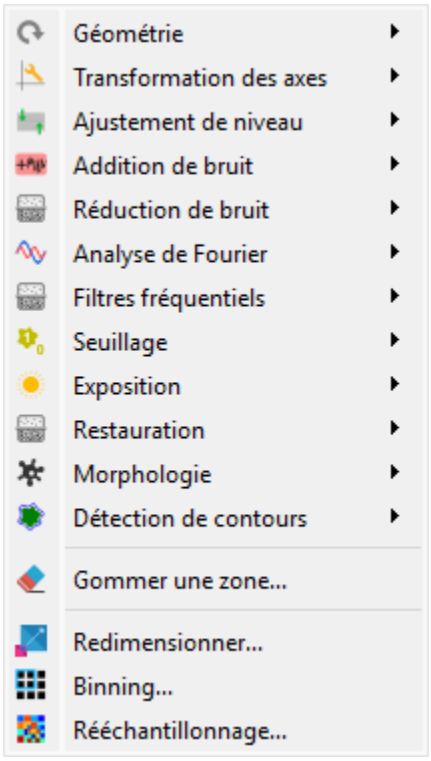


Fig. 56 – Capture d'écran du menu « Traitement ».

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.

Remplacer les valeurs spéciales

Crée une nouvelle image où les pixels `NaN`, `+Inf` et `-Inf` sont remplacés à l'aide d'une stratégie configurable pour chaque valeur cible.

Chacune des trois cibles (`NaN`, `+Inf`, `-Inf`) est traitée indépendamment. Pour chaque cible, l'utilisateur sélectionne l'une des stratégies suivantes (ou Ne rien faire pour laisser la cible inchangée) :

Stratégie	Description
Ne rien faire	Laisser les valeurs cibles inchangées.
Remplacer par zéro	Remplacer par `0`.
Remplacer par une constante	Remplacer par une valeur constante définie par l'utilisateur.
Remplacer par le minimum / maximum	Remplacer par le minimum / maximum des pixels finis.
Remplacer par la moyenne / médiane	Remplacer par la moyenne / médiane des pixels finis.
Minimum / maximum / moyenne / médiane sur N voisins	Remplacer par la statistique correspondante calculée sur le voisinage `(2N+1) × (2N+1)` autour de chaque pixel affecté.

Avant d'ouvrir le formulaire de paramètres standard, la boîte de dialogue affiche le nombre de chaque cible trouvée dans l'image (par exemple NaN : 124 (0,5 %)). Lorsqu'une stratégie fondée sur les voisins est sélectionnée, un petit aperçu montre le noyau 2D qui sera utilisé.

> **Note**
>
> Contrairement aux signaux, aucune stratégie d'interpolation n'est actuellement disponible pour les images.

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)	$z_1 = \lvert\mathrm{FFT}\,(z_0)\rvert$ ou $z_1 = 20\log_{10}\left(\lvert\mathrm{FFT}\,(z_0)\rvert\right)$ (dB)
Spectre de phase	Phase de la FFT exprimée en degrés, en utilisant numpy.angle	$z_1 = \angle\left(\mathrm{FFT}\,(z_0)\right)$
Densité spectrale de puissance	Optionnel : sortie en décibels (dB)	$z_1 = \lvert\mathrm{FFT}\,(z_0)\rvert^2$ ou $z_1 = 10\log_{10}\left(\lvert\mathrm{FFT}\,(z_0)\rvert^2\right)$ (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 de nouvelles images en appliquant des corrections d'exposition. Le réglage de la luminosité et du contraste utilise la première image sélectionnée pour initialiser une plage d'entrée commune, puis applique cette plage indépendamment à chaque image sélectionnée. Chaque résultat conserve la plage de sortie propre à son image source.

Les fonctions suivantes sont disponibles :

Fonction	Implémentation	Commentaires
Luminosité et contraste	sigima.proc.image.adjust_brightness_cor	Remappage linéaire écrêté piloté par histogramme, initialisé à partir de la première image sélectionnée
Correction gamma	skimage.exposure.adjust_gamm	
Correction logarithmique	skimage.exposure.adjust_log	
Correction sigmoïde	skimage.exposure.adjust_sigmc	
Egalisation d'histogramme	skimage.exposure.equalize_hist	
Egalisation d'histogramme adaptative	skimage.exposure.equalize_ada	Algorithme CLAHE (Contrast Limited Adaptive Histogram Equalization)
Ajustement des niveaux	skimage.exposure.rescale_inten	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_bil	
Débruitage par ondelettes	skimage.restoration.denoise_wa	
Débruitage par Top-Hat	skimage.morphology.white_top	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
x_{min}	Coordonnée x du centre du premier pixel de sortie
x_{max}	Coordonnée x du centre du dernier pixel de sortie
y_{min}	Coordonnée y du centre du premier pixel de sortie
y_{max}	Coordonnée y du centre du dernier pixel 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.
ΔX	Taille des pixels selon la direction x (si le mode "Taille des pixels" est sélectionné)
ΔY	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)

3.3.7 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.

Statistiques

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

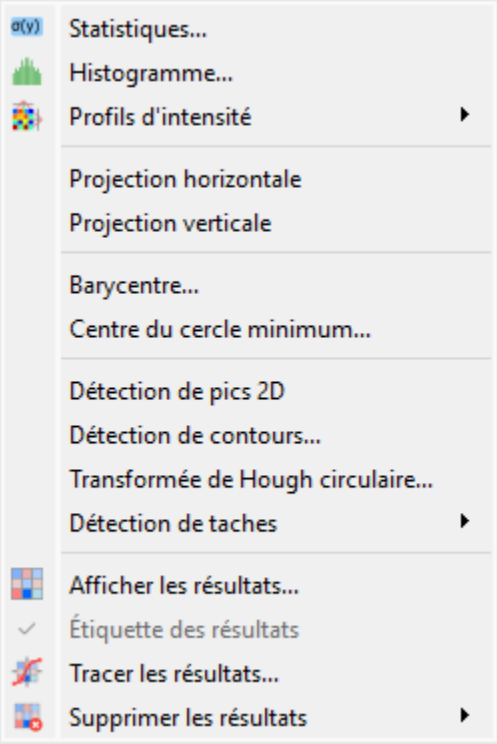


Fig. 57 – Capture d'écran du menu Analyse ».

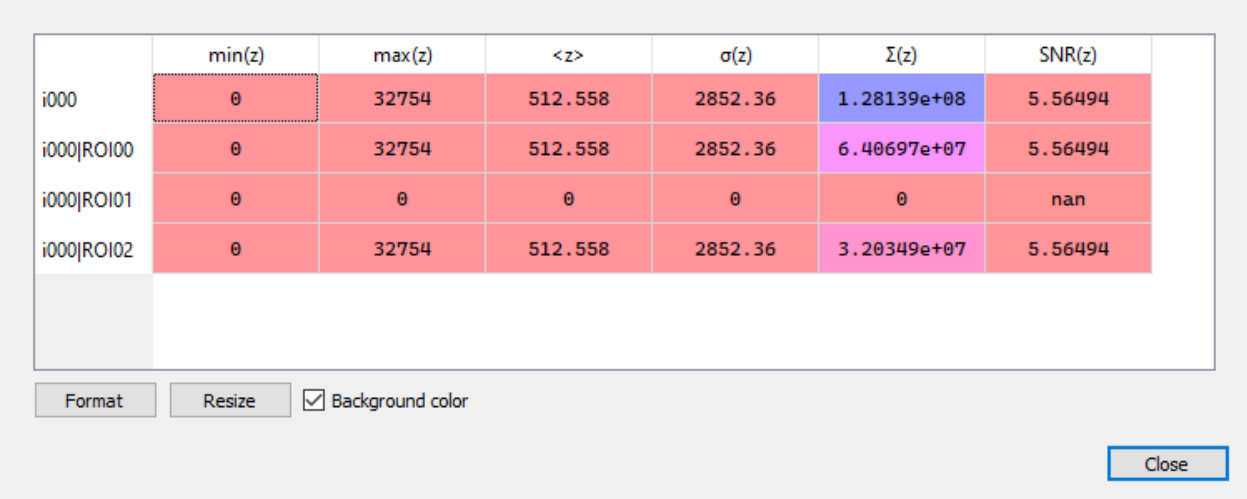


Fig. 58 – Exemple de tableau récapitulatif de statistiques : chaque ligne est associée à une ROI (à l'exception de la première qui correspond aux statistiques calculées sur la totalité des données).

Histogramme

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

Paramètres :

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

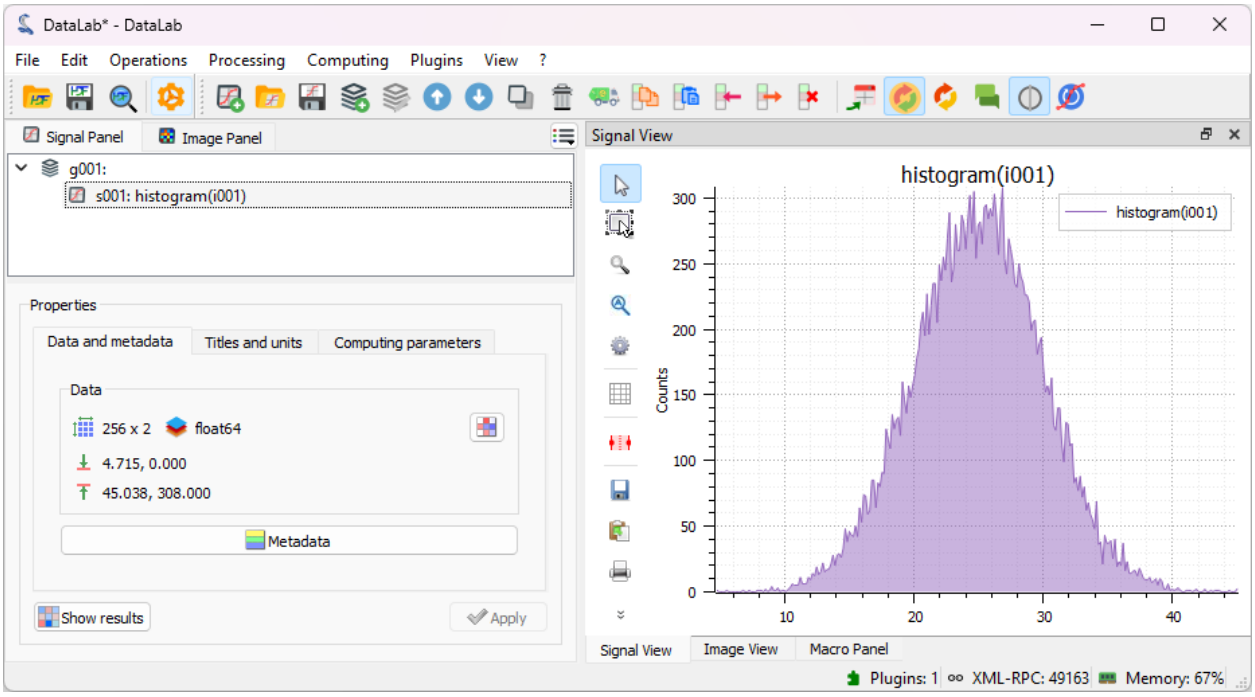


Fig. 59 – Exemple d'histogramme.

Profils d'intensité

Les profils sont définis dans les coordonnées physiques de l'image, comme les régions d'intérêt et les résultats d'analyse. L'axe X des signaux obtenus contient des coordonnées physiques (profils rectilignes et moyens) ou des distances (profils de segment et radiaux), avec l'unité de l'axe correspondant de l'image. Les profils de segment et radiaux exigent la même unité sur les axes X et Y de l'image.

Profil rectiligne

Extraire un profil horizontal ou vertical de chaque image sélectionnée et créer un nouveau signal à partir de chacun de ces profils. Le profil est extrait le long de la ligne (ou de la colonne) de pixels dont le centre est le plus proche de la coordonnée Y (ou X) indiquée.

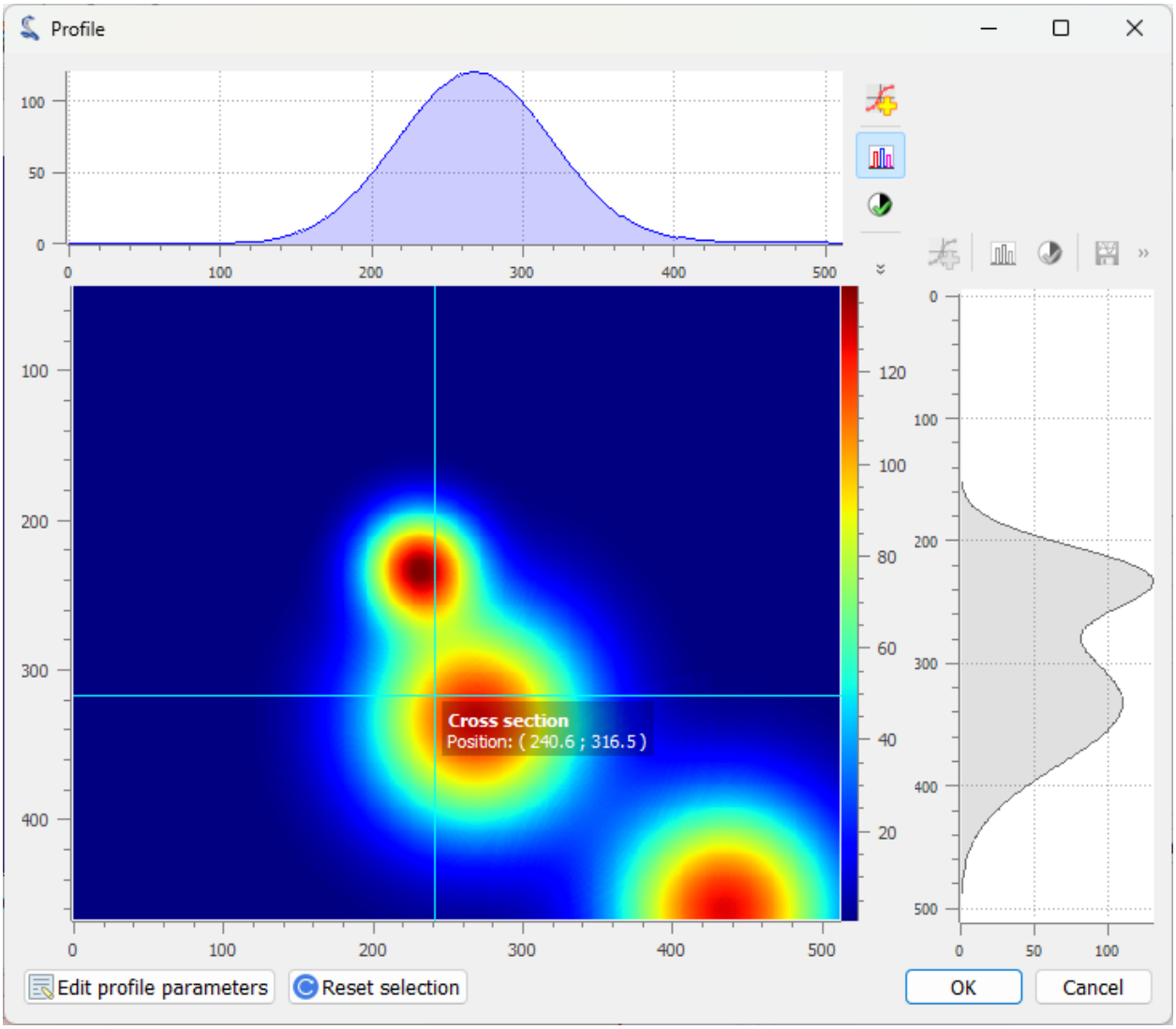


Fig. 60 – Boîte de dialogue d'extraction de profil. Les paramètres peuvent être également définis manuellement (bouton « Editer les paramètres du profil »).

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. Le segment est échantillonné à pas réguliers de son point de départ à son point d'arrivée inclus, avec un échantillon par pixel le long de l'axe sur lequel il couvre le plus de pixels, chaque échantillon prenant la valeur du pixel le plus proche.

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. Les pixels dont le centre se trouve dans le rectangle (bornes comprises) sont moyennés, comme pour les régions d'intérêt rectangulaires.

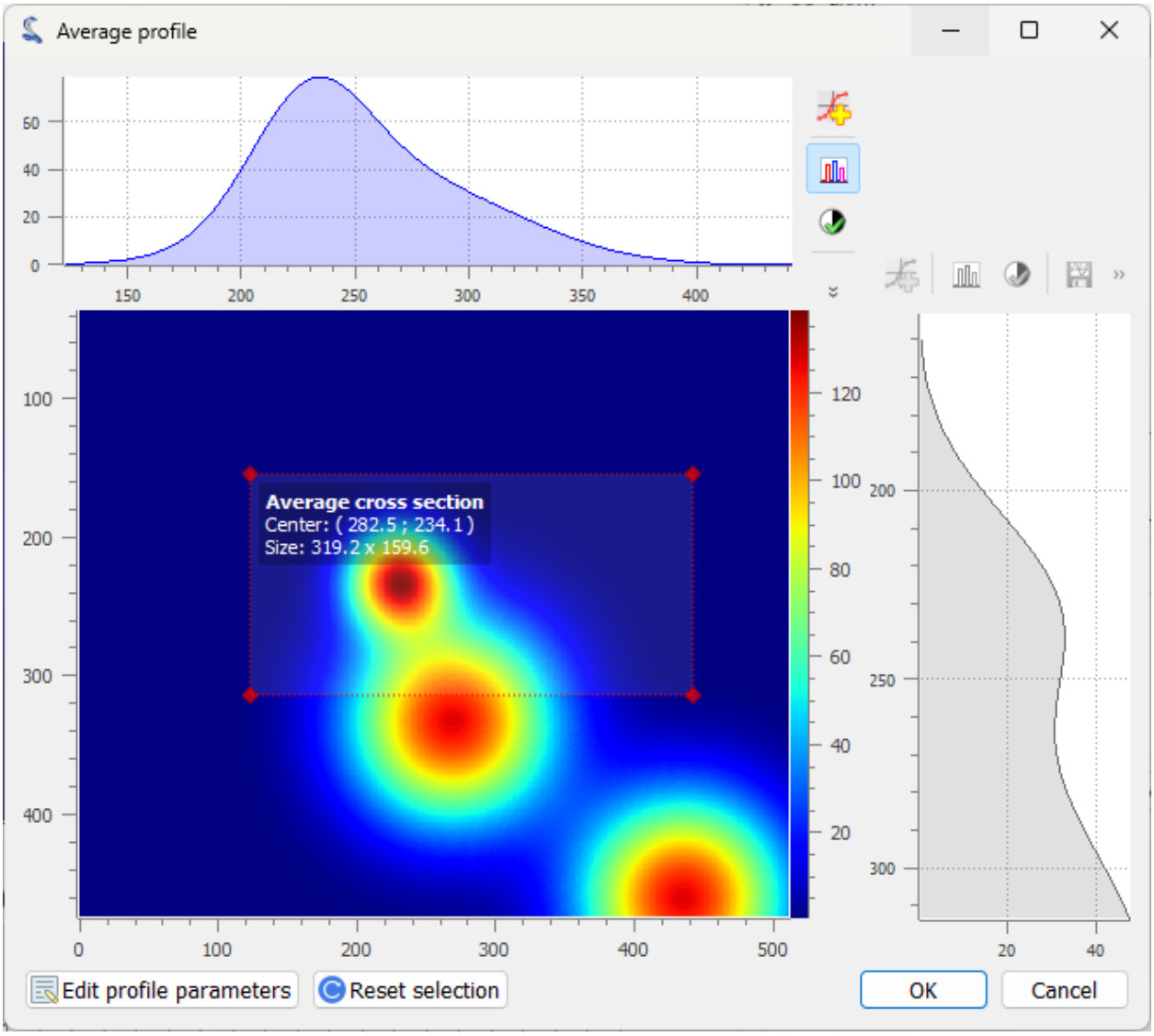


Fig. 61 – Boîte de dialogue d'extraction de profil moyen : la zone est définie par un rectangle. Les paramètres peuvent être également définis manuellement (bouton « Editer les paramètres du profil »).

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 définie par l'utilisateur), en unités physiques
Y	Coordonnée Y du centre (si définie par l'utilisateur), en unités physiques

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.

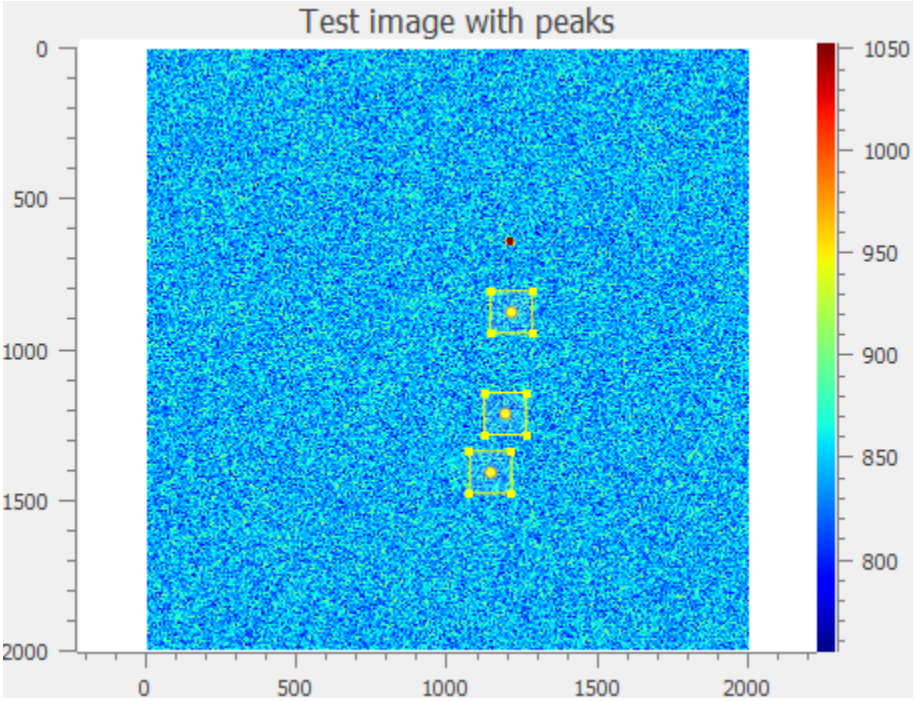


Fig. 62 – Exemple de détection de pics 2D.

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.

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 ski-

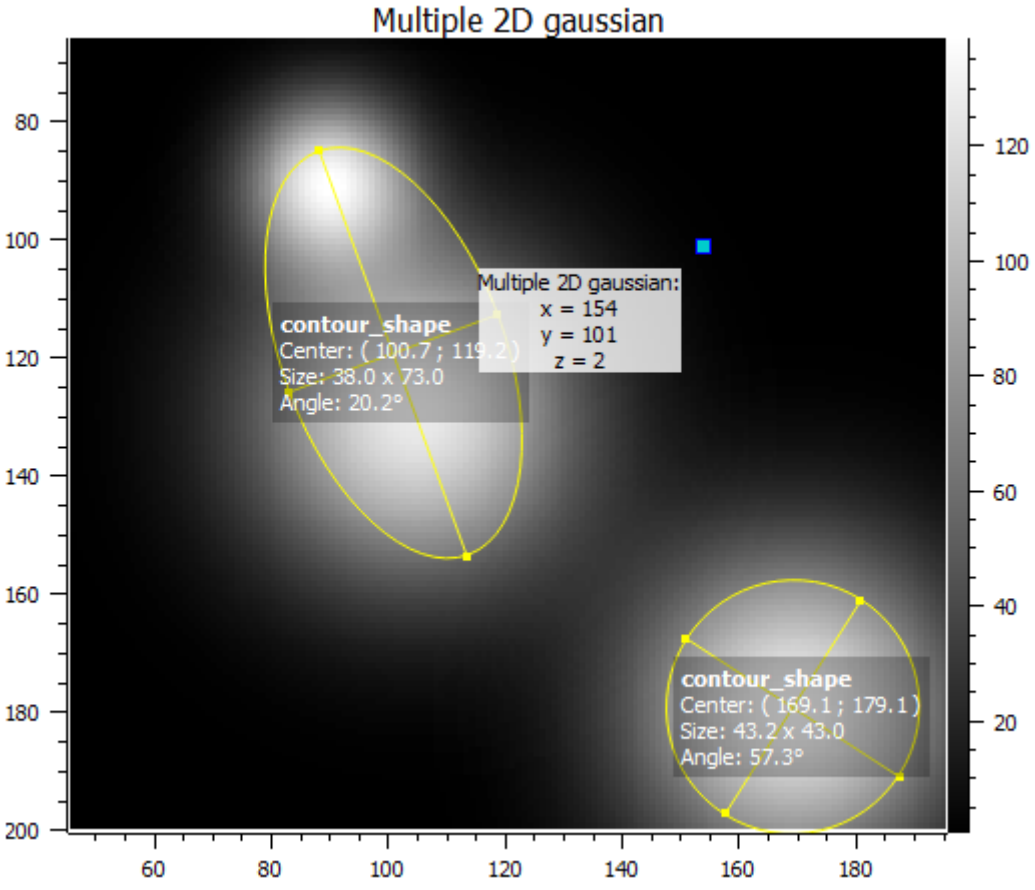


Fig. 63 – Exemple de détection de contours.

mage.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.
>
> Les ROI créées sont ajoutées aux régions d'intérêt déjà définies sur l'image, au lieu de les remplacer : enchaîner plusieurs détections cumule leurs résultats, et aucune boîte de dialogue de confirmation destructive n'est affichée. Il s'agit de ROI ordinaires, qui peuvent ensuite être déplacées, redimensionnées ou supprimées comme toute région définie manuellement.

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).

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

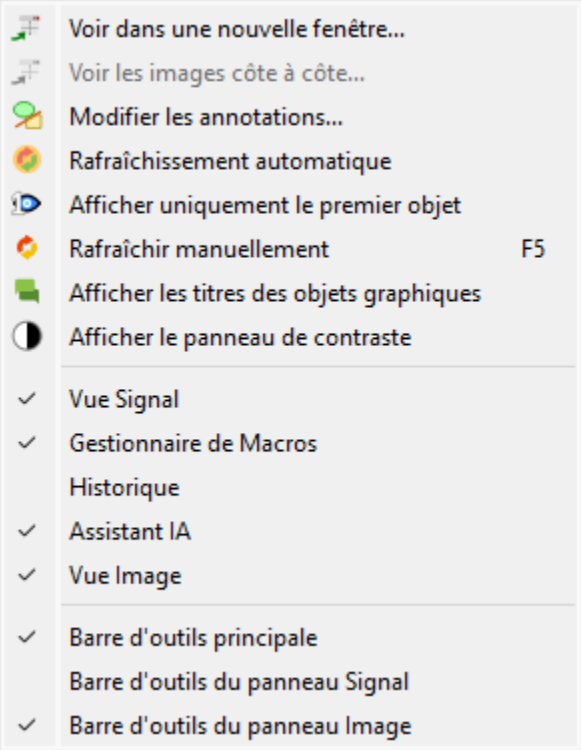


Fig. 64 – Capture d'écran du menu « Affichage ».

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.

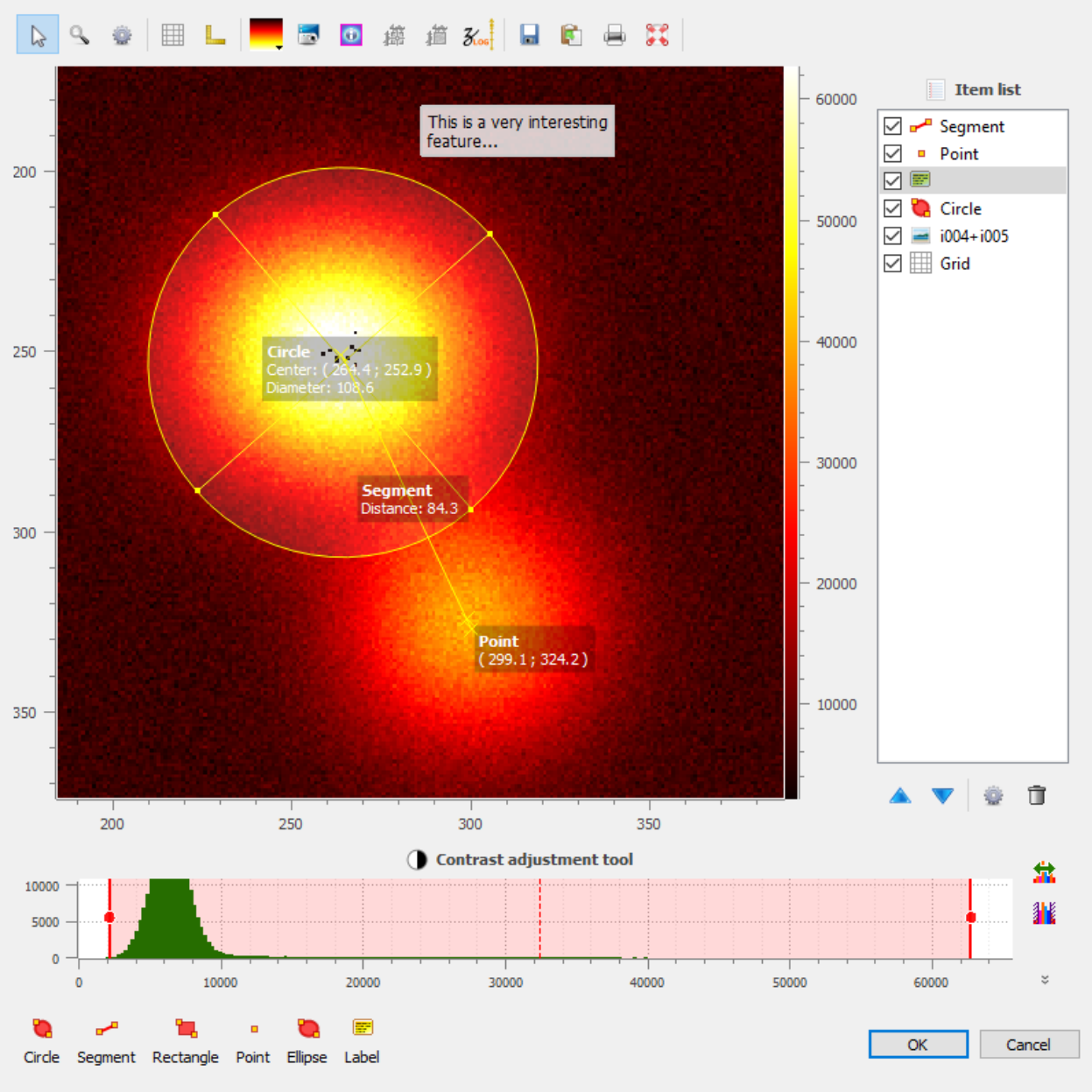


Fig. 65 – Les annotations peuvent être ajoutées dans la vue séparée.

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 de l'image et seront affichées dans la fenêtre principale.

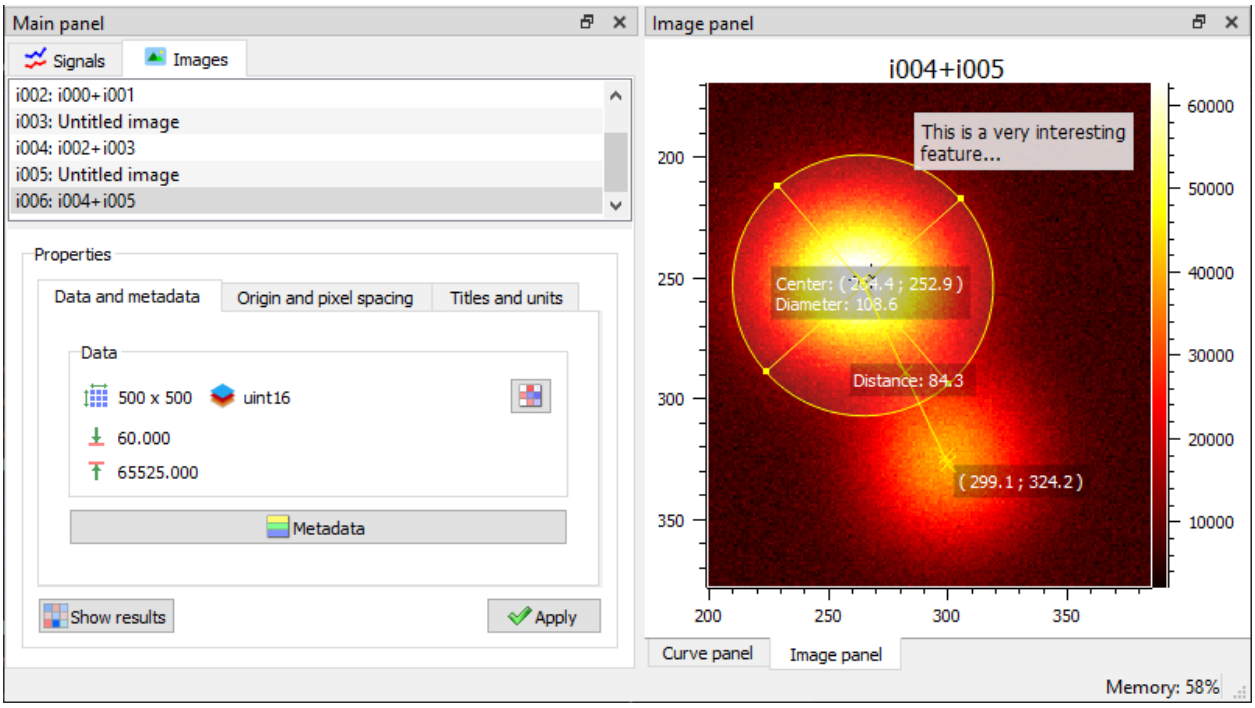


Fig. 66 – Les annotations font désormais partie des métadonnées de l'image.

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

3.3.9 Détection de pics 2D

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

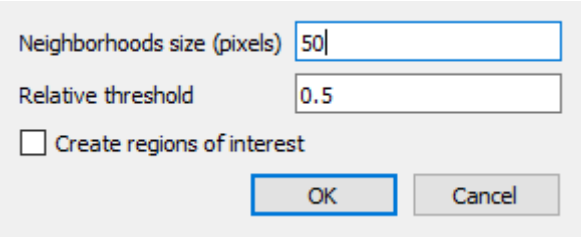


Fig. 67 – Paramètres de détection de pics 2D.

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

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



Fig. 68 – Résultats d'une détection de pics 2D (voir le test « peak2d_app.py »)

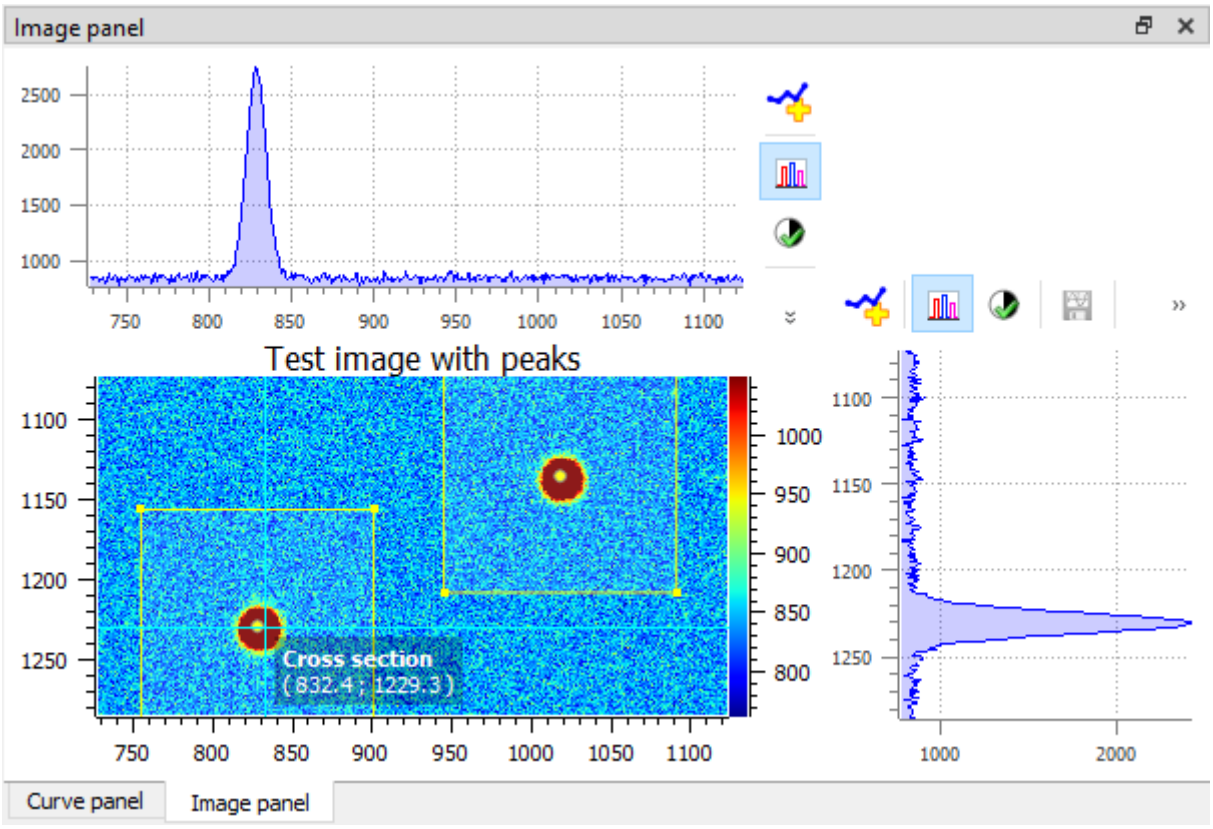


Fig. 69 – Exemple de détection de pics 2D.

— 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` :

```
@warn_if_masked
@check_2d_array(non_constant=True)
def get_2d_peaks_coords(
    data: np.ndarray | ma.MaskedArray, 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))
```

(suite sur la page suivante)

(suite de la page précédente)

```
            coords.append((x_center, y_center))
        if len(coords) > 1:
            # Eventually removing duplicates
            dist = distance_matrix(coords)
            for index in reversed(np.unique(np.where((dist < size) & (dist >␣
↪0))[1])):
                coords.pop(index)
        return np.array(coords)
```

3.3.10 Détection de contours

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

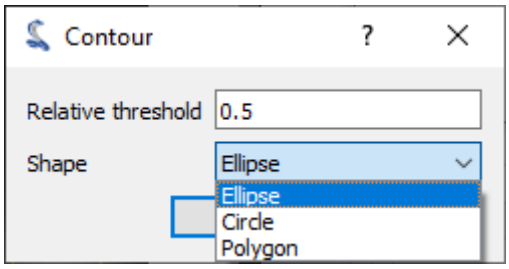


Fig. 70 – Paramètres de détection de contours.

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.tools` :

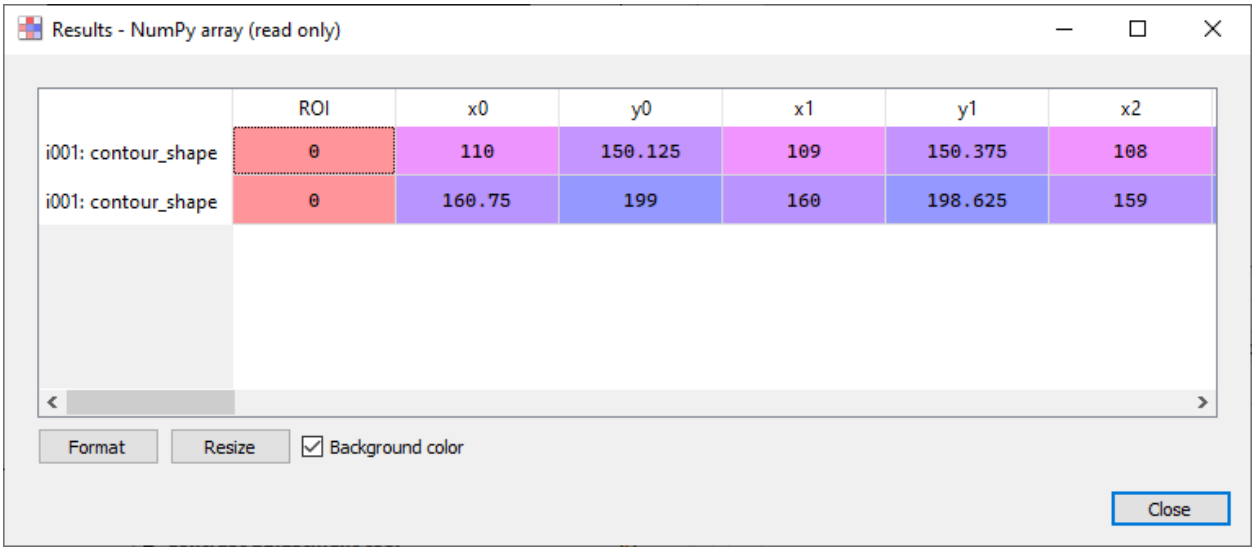


Fig. 71 – Résultats de la détection de contours (cf. test « contour_app.py »)

```
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:
            result = fit_circle_model(contour)
            if result:
                xc, yc, r = result
```

(suite sur la page suivante)

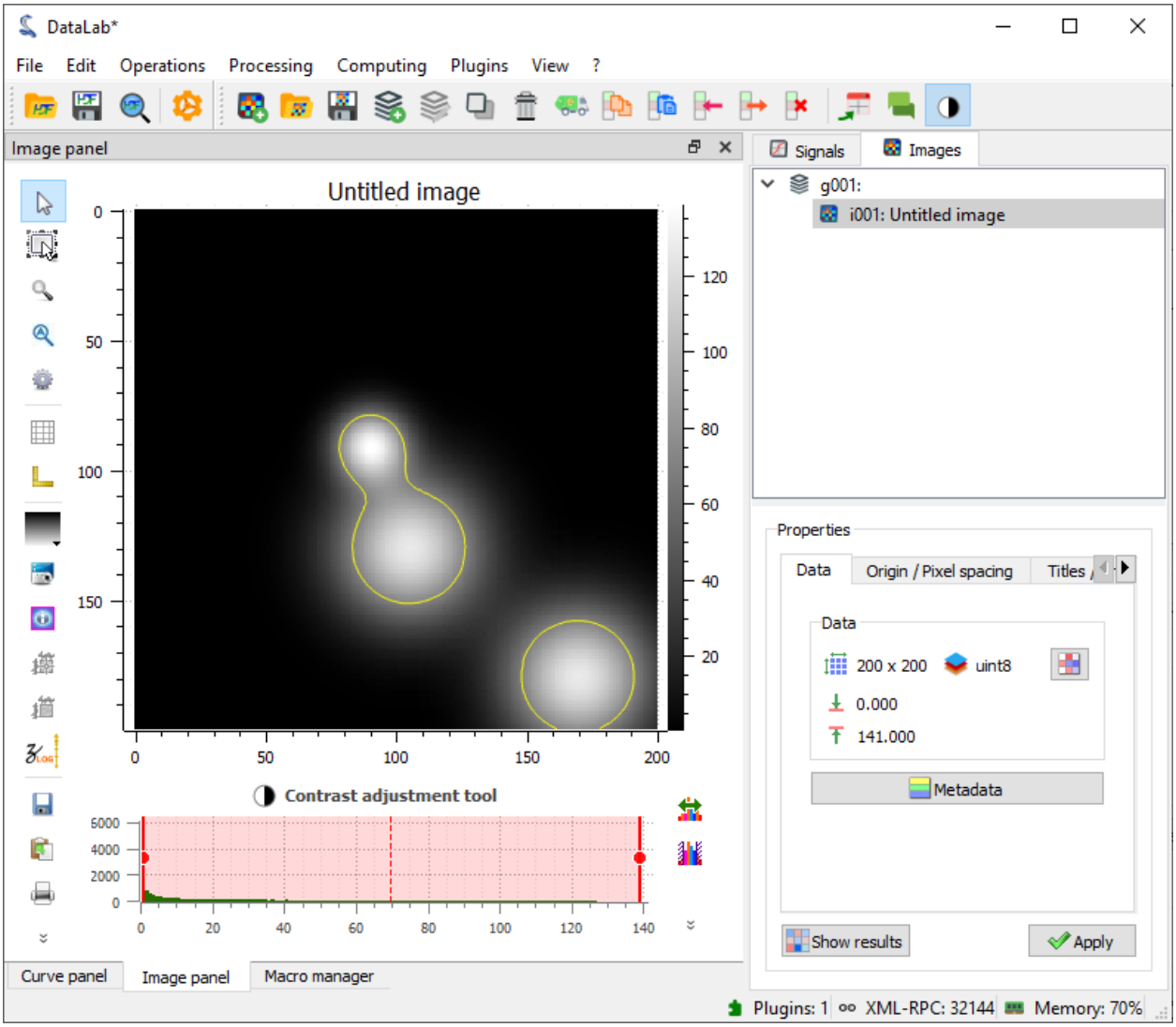


Fig. 72 – Exemple de détection de contours.

(suite de la page précédente)

```
                if r > 1.0:
                    coords.append([xc, yc, r])
        elif shape == ContourShape.ELLIPSE:
            result = fit_ellipse_model(contour)
            if result:
                xc, yc, a, b, theta = result
                if a > 1.0 and b > 1.0:
                    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 shape {shape}")
    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)
```

3.4 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).

3.4.1 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.

3.4.2 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.

3.4.3 Etat de validation de DataLab

Validation fonctionnelle

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

Package	Couverture	Nombre de tests
DataLab	~82%	>130
Sigima	~90%	>570

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 4 – Statistiques de validation

Catégorie	Signal	Image	Total
Nombre de fonctions de calcul	98	117	215
Nombre de fonctions de calcul validées	98	117	215
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 5 : Etat de validation de DataLab

Fonctions de calcul	Description	Fonction de test
`absolute`	Calculer la valeur absolue avec `numpy.absolute`	test_signal_absolute
`add_gaussian_noise`	Ajouter un bruit normal au signal d'entrée	test_signal_add_gaussian_noise
`add_poisson_noise`	Ajouter un bruit de Poisson au signal d'entrée	test_signal_add_poisson_noise
`add_uniform_noise`	Ajouter un bruit uniforme au signal d'entrée	test_signal_add_uniform_noise
`addition`	Calculer la somme élément par élément de plusieurs signaux	test_signal_addition
`addition_constant`	Calculer la somme d'un signal et d'une valeur constante	test_signal_addition_constant
`allan_deviation`	Calculer la déviation d'Allan	test_signal_allan_deviation
`allan_variance`	Calculer la variance d'Allan	test_signal_allan_variance
`apply_window`	Calcul d'un fenêtrage	test_signal_apply_window
`arithmetic`	Effectuer une opération arithmétique sur deux signaux	test_signal_arithmetic
`astype`	Convertir le type de données	test_signal_astype

suite sur la page suivante

Tableau 5 – suite de la page précédente

Fonctions de calcul	Description	Fonction de test
average	Calculer la moyenne élément par élément de plusieurs signaux	test_signal_average
bandpass	Calculer le filtre passe-bande	test_signal_bandpass
bandstop	Calculer le filtre coupe-bande	test_signal_bandstop
bandwidth_3db	Calculer la largeur de bande à -3 dB	test_signal_bandwidth_3db
calibration	Calculer l'étalonnage linéaire	test_signal_calibration
cdf_fit	Calculer l'ajustement CDF	test_signal_cdf_fit
clip	Calculer le découpage maximal des données	test_signal_clip
complex_from_magnitude_phase	Combiner les signaux de magnitude et de phase en un signal complexe	test_signal_complex_from_magnitude_phase
complex_from_real_imag	Combiner deux signaux réels en un signal complexe en utilisant réel + i * imag	test_signal_complex_from_real_imag
contrast	Calculer le contraste	test_signal_contrast
convolution	Calculer la convolution de deux signaux	test_signal_convolution
deconvolution	Calculer la déconvolution de deux signaux	test_signal_deconvolution
derivative	Calculer la dérivée	test_signal_derivative
detrending	Supprimer la tendance des données	test_signal_detrending
difference	Calculer la différence élément par élément entre deux signaux	test_signal_difference
difference_constant	Calculer la différence entre un signal et une valeur constante	test_signal_difference_constant
division	Calculer la division élément par élément entre deux signaux	test_signal_division
division_constant	Calculer la division d'un signal par une valeur constante	test_signal_division_constant
dynamic_parameters	Calculer les paramètres dynamiques	test_dynamic_parameters
evaluate_fit	Évaluer la fonction d'ajustement de src1 sur l'axe des x de src2	test_signal_evaluate_fit
exp	Calculer l'exponentielle avec numpy.exp	test_signal_exp
exponential_fit	Calculer l'ajustement exponentiel	test_signal_exponential_fit
extract_peak_positions	Extraire les positions des pics sous forme de tableau de marqueurs XY	test_signal_extract_peak_positions
extract_pulse_features	Extraire les caractéristiques des impulsions	test_signal_extract_pulse_features
extract_roi	Extraire une seule région d'intérêt à partir des données	test_signal_extract_roi
extract_rois	Extraire plusieurs régions d'intérêt à partir des données	test_signal_extract_rois
fft	Calculer la FFT	test_signal_fft
full_width_at_y		test_signal_full_width_at_y
fw1e2	Calculer la largeur à $1/e^2$	test_signal_fw1e2
fwhm	Calculer la largeur à mi-hauteur	test_signal_fwhm
gaussian_filter	Calculer le filtre gaussien	test_signal_gaussian_filter
gaussian_fit	Calculer l'ajustement gaussien	test_signal_gaussian_fit
hadamard_variance	Calculer la variance d'Hadamard	test_signal_hadamard_variance
highpass	Calculer le filtre passe-haut	test_signal_highpass
histogram	Calculer l'histogramme	test_signal_histogram
ifft	Calculer la FFT inverse	test_signal_ifft
imag	Calculer la partie imaginaire	test_signal_imag
integral	Calculer l'intégrale	test_signal_integral
interpolate	Interpoler les données	test_signal_interpolate
inverse	Calculer l'inverse élément par élément d'un signal	test_signal_inverse

suite sur la page suivante

Tableau 5 – suite de la page précédente

Fonctions de calcul	Description	Fonction de test
linear_fit	Calculer l'ajustement linéaire	test_signal_linear_fit
log10	Calculer Log10 avec numpy.log10	test_signal_log10
lorentzian_fit	Calculer l'ajustement lorentzien	test_signal_lorentzian_fit
lowpass	Calculer le filtre passe-bas	test_signal_lowpass
magnitude_spectrum	Calculer le spectre de magnitude	test_signal_magnitude_spectrum
modified_allan_variance	Calculer la variance d'Allan modifiée	test_signal_modified_allan_variance
moving_average	Calculer la moyenne mobile	test_signal_moving_average
moving_median	Calculer la médiane mobile	test_signal_moving_median
normalize	Normaliser les données	test_signal_normalize
offset_correction	Corriger le décalage : soustraire la valeur moyenne du signal dans la plage spécifiée	test_signal_offset_correction
overlapping_allan_variance	Calculer la variance d'Allan chevauchante	test_signal_overlapping_allan_variance
peak_detection	Détection des pics	test_signal_peak_detection
phase	Calculer la phase (argument) d'un signal complexe	test_signal_phase
phase_spectrum	Calculer le spectre de phase	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	Calculer la puissance avec numpy.power	test_signal_power
product	Calculer le produit élément par élément de plusieurs signaux	test_signal_product
product_constant	Calculer le produit d'un signal et d'une valeur constante	test_signal_product_constant
psd	Calculer la densité spectrale de puissance	test_signal_psd
quadratic_difference	Calculer la différence normalisée entre deux signaux	test_signal_quadratic_difference
real	Calculer la partie réelle	test_signal_real
replace_special_values	Remplacer les valeurs NaN, +Inf et -Inf dans un signal	test_signal_replace_special_values
replace_x_by_other_y	Créer un nouveau signal en utilisant Y de src1 et Y de src2 comme coordonnées X	test_replace_x_by_other_y
resampling	Rééchantillonner les données	test_signal_resampling
reverse_x	Inverser l'axe des x	test_signal_reverse_x
sampling_rate_period	Calculer la fréquence d'échantillonnage et la période	test_signal_sampling_rate_period
sigmoid_fit	Calculer l'ajustement sigmoïde	test_signal_sigmoid_fit
signals_to_image	Combiner plusieurs signaux en une image	test_signal_signals_to_image
sinusoidal_fit	Calculer l'ajustement sinusoïdal	test_sinusoidal_fit
sqrt	Calculer la racine carrée avec numpy.sqrt	test_signal_sqrt
standard_deviation	Calculer l'écart-type élément par élément de plusieurs signaux	test_signal_standard_deviation
stats	Calculer les statistiques sur un signal	test_signal_stats_unit
time_deviation	Calculer la déviation temporelle (TDEV)	test_signal_time_deviation
to_cartesian	Convertir des coordonnées polaires en coordonnées cartésiennes	test_signal_to_cartesian
to_polar	Convertir des coordonnées cartésiennes en coordonnées polaires	test_signal_to_polar
total_variance	Calculer la variance totale	test_signal_total_variance

suite sur la page suivante

Tableau 5 – suite de la page précédente

Fonctions de calcul	Description	Fonction de test
transpose	Transposer le signal (échanger les axes X et Y)	test_signal_transpose
twohalfgaussian_fit	Calculer l'ajustement à deux demi-Gaussiennes	test_signal_twohalfgaussian_fit
voigt_fit	Calculer l'ajustement de Voigt	test_signal_voigt_fit
wiener	Calculer le filtre de Wiener	test_signal_wiener
x_at_minmax		test_signal_x_at_minmax
x_at_y		test_signal_x_at_y
xy_mode	Simuler le mode X-Y d'un oscilloscope	test_signal_xy_mode
y_at_x		test_signal_y_at_x
zero_padding	Calculer le remplissage nul	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 6 : Etat de validation des fonctions de calcul image

Fonctions de calcul	Description	Fonction de test
absolute	Calculer la valeur absolue avec numpy.absolute	test_image_absolute
add_gaussian_noise	Ajouter du bruit gaussien (normal) à l'image d'entrée	test_image_add_gaussian_noise
add_poisson_noise	Ajouter du bruit de Poisson à l'image d'entrée	test_image_add_poisson_noise
add_uniform_noise	Ajouter du bruit uniforme à l'image d'entrée	test_image_add_uniform_noise
addition	Ajouter les images de la liste et retourner l'objet image résultant	test_image_addition
addition_constant	Ajouter dst et une valeur constante et retourner le nouvel objet image résultant	test_image_addition_constant
adjust_brightness_contrast	Apply a clipped linear brightness and contrast adjustment	test_adjust_brightness_contrast
adjust_gamma	Correction gamma	test_adjust_gamma
adjust_log	Calculer l'ajustement logarithmique	test_adjust_log
adjust_sigmoid	Calculer l'ajustement sigmoïde	test_adjust_sigmoid
arithmetic	Effectuer une opération arithmétique sur deux images	test_image_arithmetic
astype	Convertir le type de données de l'image	test_image_astype
average	Calculer la moyenne des images dans la liste et retourner l'objet image résultant	test_image_average
average_profile	Calculer le profil moyen horizontal ou vertical	test_average_profile
binning	Regroupement : regroupement (ou agrégation) des pixels de l'image	test_binning
black_tophat	Calculer le Black Top-Hat	test_black_tophat
blob_dog	Calculer les blobs en utilisant la méthode de la différence de Gaussienne	test_image_blob_dog
blob_doh	Calculer les blobs en utilisant la méthode du déterminant de Hessian	test_image_blob_doh
blob_log	Calculer les blobs en utilisant la méthode du Laplacien de Gaussienne	test_image_blob_log
blob_opencv	Calculer les blobs en utilisant OpenCV	test_image_blob_opencv

suite sur la page suivante

Tableau 6 – suite de la page précédente

Fonctions de calcul	Description	Fonction de test
butterworth	Calculer le filtre de Butterworth	test_butterworth
calibration	Calculer l'étalonnage polynomial	test_image_calibration
canny	Calculer le filtre de Canny	test_canny
centroid	Calculer le centroïde	test_image_centroid
clip	Appliquer le clipping	test_image_clip
closing	Calculer la fermeture morphologique	test_closing
complex_from_magnitude_phase	Combiner les images de magnitude et de phase en une image complexe	test_image_complex_from_magnitude_phase
complex_from_real_imag	Combiner deux images réelles en une image complexe en utilisant réel + i * imag	test_image_complex_from_real_imag
contour_shape	Calculer la forme du contour	test_contour_shape
convolution	Convoluer une image avec un noyau	test_image_convolution
deconvolution	Déconvoluer un noyau d'une image en utilisant la Transformée de Fourier Rapide (FFT)	test_image_deconvolution
denoise_bilateral	Calculer le débruitage par filtre bilatéral	test_denoise_bilateral
denoise_tophat	Débruiter en utilisant le White Top-Hat	test_denoise_tophat
denoise_tv	Calculer le débruitage par variation totale	test_denoise_tv
denoise_wavelet	Calculer le débruitage par ondelettes	test_denoise_wavelet
difference	Calculer la différence entre deux images	test_image_difference
difference_constant	Soustraire une valeur constante d'une image et retourner le nouvel objet image résultant	test_image_difference_constant
dilation	Calculer la dilatation	test_dilation
division	Calculer la division entre deux images	test_image_division
division_constant	Diviser une image par une valeur constante et retourner le nouvel objet image résultant	test_image_division_constant
enclosing_circle	Calculer le cercle circonscrit minimum	test_image_enclosing_circle
equalize_adapthist	Égalisation adaptative de l'histogramme	test_equalize_adapthist
equalize_hist	Égalisation de l'histogramme	test_equalize_hist
erase	Effacer une zone de l'image en utilisant la valeur moyenne de l'image	test_erase
erosion	Calculer l'érosion	test_erosion
exp	Calculer l'exponentielle avec numpy.exp	test_image_exp
extract_roi	Extraire une seule ROI	test_image_extract_roi
extract_rois	Extraire plusieurs régions d'intérêt à partir des données	test_image_extract_rois
farid	Calculer le filtre de Farid	test_farid
farid_h	Calculer le filtre de Farid horizontal	test_farid_h
farid_v	Calculer le filtre de Farid vertical	test_farid_v
fft	Calculer la FFT	test_image_fft
flatfield	Calculer la correction de champ plat	test_flatfield
fliph	Retourner les données horizontalement	test_image_fliph
flipv	Retourner les données verticalement	test_image_flipv
gaussian_filter	Calculer le filtre gaussien	test_image_gaussian_filter
gaussian_freq_filter	Appliquer un filtre gaussien dans le domaine fréquentiel	test_gaussian_freq_filter
histogram	Calculer l'histogramme des données de l'image	test_image_histogram
horizontal_projection	Calculer la somme des intensités des pixels le long de chaque colonne (projection sur l'axe x)	test_image_horizontal_projection
hough_circle_peaks	Calculer les cercles de Hough	test_image_hough_circle_peaks
ifft	Calculer la transformée de Fourier inverse	test_image_ifft

suite sur la page suivante

Tableau 6 – suite de la page précédente

Fonctions de calcul	Description	Fonction de test
imag	Calculer la partie imaginaire	test_image_imag
inverse	Calculer l'inverse d'une image et retourner le nouvel objet image résultant	test_image_inverse
laplace	Calculer le filtre de Laplace	test_laplace
line_profile	Calculer le profil horizontal ou vertical	test_line_profile
log10	Calculer le logarithme en base 10 avec numpy.log10	test_image_log10
log10_z_plus_n	Calculer le logarithme en base 10 de (z+n) avec numpy.log10	test_image_log10_z_plus_n
magnitude_spectrum	Calculer le spectre de magnitude	test_image_magnitude_spectrum
moving_average	Calculer la moyenne mobile	test_image_moving_average
moving_median	Calculer la médiane mobile	test_image_moving_median
normalize		test_image_normalize
offset_correction	Appliquer la correction de l'offset	test_image_offset_correction
opening	Calculer l'ouverture morphologique	test_opening
peak_detection	Calculer la détection de pics 2D	test_image_peak_detection
phase	Calculer la phase (argument) d'une image complexe	test_image_phase
phase_spectrum	Calculer le spectre de phase	test_image_phase_spectrum
prewitt	Calculer le filtre de Prewitt	test_prewitt
prewitt_h	Calculer le filtre de Prewitt horizontal	test_prewitt_h
prewitt_v	Calculer le filtre de Prewitt vertical	test_prewitt_v
product	Multiplier les images de la liste et retourner le nouvel objet image résultant	test_image_product
product_constant	Multiplier dst par une valeur constante et retourner le nouvel objet image résultant	test_image_product_constant
psd	Calculer la densité spectrale de puissance	test_image_psd
quadratic_difference	Calculer la différence quadratique entre deux images	test_image_quadratic_difference
radial_profile	Compute radial profile around the centroid, the image center or a given point	test_radial_profile
real	Calculer la partie réelle	test_image_real
replace_special_values	Remplacer les valeurs NaN, +Inf et -Inf dans une image	test_image_replace_special_values
resampling	Rééchantillonner l'image sur une nouvelle grille de coordonnées en utilisant l'interpolation	test_image_resampling
rescale_intensity	Recaler les niveaux d'intensité de l'image	test_rescale_intensity
resize	Fonction de zoom de l'image	test_image_resize
roberts	Calculer le filtre de Roberts	test_roberts
rotate	Faire pivoter l'image	test_image_rotate
rotate270	Faire pivoter l'image de 270°	test_image_rotate270
rotate90	Faire pivoter l'image de 90°	test_image_rotate90
scharr	Calculer le filtre de Scharr	test_scharr
scharr_h	Calculer le filtre de Scharr horizontal	test_scharr_h
scharr_v	Calculer le filtre de Scharr vertical	test_scharr_v
segment_profile	Calculer le profil de segment	test_segment_profile
set_uniform_coords	Convertir l'image en un système de coordonnées uniforme	test_set_uniform_coords
sobel	Calculer le filtre de Sobel	test_sobel
sobel_h	Calculer le filtre de Sobel horizontal	test_sobel_h
sobel_v	Calculer le filtre de Sobel vertical	test_sobel_v

suite sur la page suivante

Tableau 6 – suite de la page précédente

Fonctions de calcul	Description	Fonction de test
standard_deviation	Calculer l'écart-type élément par élément de plusieurs images	test_image_standard_deviation
stats	Calculer les statistiques sur une image	test_image_stats_unit
threshold	Calculer le seuil, en utilisant l'un des algorithmes disponibles	test_threshold
threshold_isodata	Calculer le seuil en utilisant l'algorithme Isodata avec les paramètres par défaut	test_threshold_isodata
threshold_li	Calculer le seuil en utilisant l'algorithme Li avec les paramètres par défaut	test_threshold_li
threshold_mean	Calculer le seuil en utilisant l'algorithme de la moyenne	test_threshold_mean
threshold_minimum	Calculer le seuil en utilisant l'algorithme du minimum avec les paramètres par défaut	test_threshold_minimum
threshold_otsu	Calculer le seuil en utilisant l'algorithme d'Otsu avec les paramètres par défaut	test_threshold_otsu
threshold_triangle	Calculer le seuil en utilisant l'algorithme du triangle avec les paramètres par défaut	test_threshold_triangle
threshold_yen	Calculer le seuil en utilisant l'algorithme de Yen avec les paramètres par défaut	test_threshold_yen
translate	Translater les données	test_image_translate
transpose	Transposer l'image	test_image_transpose
vertical_projection	Calculer la somme des intensités des pixels le long de chaque ligne (projection sur l'axe y)	test_image_vertical_projection
white_tophat	Calculer le White Top-Hat	test_white_tophat
wiener	Calculer le filtre de Wiener	test_image_wiener
zero_padding	Zero-padding : ajouter des zéros aux bords de l'image	test_image_zero_padding

3.5 Fonctionnalités avancées

Cette section décrit les fonctionnalités avancées de DataLab.

3.5.1 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 du code Python qui est automatiquement chargé par DataLab au démarrage. Il peut définir de nouvelles fonctionnalités ou modifier des fonctionnalités existantes.

Un plugin définit une classe qui hérite de *datalab.plugins.PluginBase*, comporte un attribut de classe nommé PLUGIN_INFO (une instance de *datalab.plugins.PluginInfo*) et implémente la méthode create_actions. DataLab trouve cette classe dans un répertoire de plugins (voir Où est positionné un plugin ?), sous l'une de ces deux formes :

- Un fichier unique dont le nom commence par datalab_ (par exemple datalab_myplugin.py) et qui contient la classe. C'est la forme la plus simple, recommandée pour débuter : datalab-plugin create crée un tel fichier, prêt à l'emploi (voir Créer un plugin avec datalab-plugin create).
- Un dossier dont le nom commence par datalab_ et qui contient un fichier __init__.py (un paquet Python, par exemple datalab_myplugin/). DataLab charge le dossier, puis son fichier adapters/desktop.py s'il

existe : c'est là que les projets créés par `datalab-plugin create --project` définissent la classe (voir Créer un projet de plugin structuré en couches). Sinon, la classe est définie dans `__init__.py`.

Rien n'a besoin d'être installé : le fichier ou le dossier est chargé tel quel.

Les plugins distribués sous forme de paquets Python installés, comme les wheels du catalogue de plugins, déclarent leur sous-classe de `PluginBase` via le groupe de points d'entrée `datalab.plugins`, sans dépendre du préfixe `datalab_`. Un plugin trouvé des deux façons, par exemple un projet installé dont le dossier est aussi un répertoire de plugins, n'est chargé qu'une seule fois.

L'objet `PLUGIN_INFO` doit définir un `id` unique, préfixé par un espace de noms, qui reste stable lorsque le nom affiché ou la classe d'implémentation changent. DataLab utilise cet identifiant pour l'enregistrement et pour les paramètres d'activation persistants. Les plugins sans identifiant restent pris en charge grâce à un repli fondé sur le module et la classe, pour assurer la rétrocompatibilité, mais les nouveaux plugins ne doivent pas s'appuyer sur ce repli.

```
from datalab.plugins import PluginCapability, PluginInfo

PLUGIN_INFO = PluginInfo(
  id="org.example.my-plugin",
  name="My Plugin",
  version="1.0.0",
  capabilities=(
    PluginCapability.APPLICATION,
    PluginCapability.PROCESSING,
  ),
)
```

`capabilities` déclare comment DataLab peut présenter et utiliser un plugin. Les valeurs prises en charge sont `PROCESSING`, `IO`, `VISUALIZATION` et `APPLICATION`. Un plugin peut combiner plusieurs valeurs ; les applications métier déclarent généralement `APPLICATION` avec `PROCESSING`. La boîte de dialogue de configuration des plugins affiche les capacités déclarées dans un ordre stable. Les plugins existants qui omettent ce champ restent valides et sont affichés sans libellé de capacité.

> **Note**
>
> Seuls les fichiers et dossiers dont les noms commencent par `datalab_` sont 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.

Organisation du code source et distribution

L'organisation du code source, le format de distribution et les capacités fonctionnelles sont des notions indépendantes. Un plugin en fichier unique et un plugin organisé en paquet Python utilisent la même API *`datalab.plugins.PluginBase`* ; aucune de ces formes ne détermine s'il fournit des fonctions de traitement, d'entrée/sortie, de visualisation ou d'application.

- Plugin en fichier unique : un module `datalab_<nom>.py`, facile à modifier et à partager sans projet de packaging. Cette forme historique est aussi celle produite par défaut par `datalab-plugin create` ; elle n'est pas obsolète.

— Paquet ou projet : des modules Python et, éventuellement, des ressources, des tests et des métadonnées de packaging. `datalab-plugin create --project` propose une organisation en couches, pas une architecture obligatoire pour tous les plugins.
— Distribution wheel : une archive `.whl` installable, contenant des métadonnées normalisées et des points d'entrée de plugins. Un wheel peut contenir un seul module ou un paquet plus important ; c'est un format de distribution, pas une catégorie fonctionnelle de plugin.

Les modules en fichier unique et les wheels peuvent être installés avec Installer depuis un fichier… ou distribués dans les catalogues en ligne et locaux. Les catalogues n'imposent pas le projet en couches généré par `--project`. Empaqueter un module unique en wheel reste possible, mais est facultatif pour Desktop.

Tableau 7 – Formes de développement et distribution

Forme de développement	Chargement direct du code source	Distribution gérée
Fichier Python unique	`datalab_<nom>.py` dans un répertoire de plugins	Fichier `.py` brut, ou wheel facultatif
Paquet ou projet Python	Dossier du paquet, y compris l'organisation `src` d'un projet	Wheel contenant le paquet et ses ressources

Les deux formats gérés fonctionnent dans les catalogues locaux et en ligne de Desktop. La forme historique en fichier unique n'est pas une catégorie fonctionnelle distincte et son code source n'a pas besoin d'être modifié pour être référencé dans un catalogue. Une entrée dans un catalogue Desktop n'implique pas la prise en charge de DataLab-Web.

Un fichier source chargé directement reste dans son répertoire de développement. Un fichier installé par DataLab est copié dans son stockage utilisateur géré. Un plugin n'est pas une macro : un plugin enregistre des fonctions persistantes dans l'application, tandis qu'une macro est un script exécuté pour automatiser une tâche. La prise en charge de la forme historique en fichier unique n'implique pas celle des API obsolètes de DataLab v0.20.

Compatibilité avec l'aperçu interactif

> **Note**
>
> Les fonctions paramétrées standard enregistrées avec `register_1_to_1` peuvent utiliser l'aperçu interactif sans moteur de rendu personnalisé. Elles doivent retourner un `SignalObj` ou un `ImageObj`, être importables dans un processus démarré par spawn et accepter des copies sérialisables des sources et paramètres. L'aperçu utilise un processus privé et ne publie aucun résultat dans l'espace de travail. Il ne protège pas des effets externes sur les fichiers, le réseau ou d'autres ressources. Enregistrez les fonctions qui ne doivent pas être exécutées provisoirement avec `preview_enabled=False`. Les erreurs de sérialisation ou de calcul sont signalées dans l'aperçu ; le chemin normal via OK reste disponible. Les implémentations personnalisées de `DataSet.edit` non prises en charge et les backends alternatifs de guidata ne sont pas remplacés.
>
> Le processus dédié à l'aperçu peut être réutilisé entre les boîtes de dialogue. Les fonctions des plugins ne doivent pas supposer qu'un nouvel interpréteur Python est créé pour chaque aperçu : l'état local des modules dans le processus peut persister jusqu'à ce qu'un aperçu en cours soit annulé, que les plugins soient rechargés ou que DataLab se ferme.

Où est positionné un plugin ?

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

— Un paquet Python installé peut déclarer sa classe de plugin via le groupe standard de points d'entrée `datalab.plugins` :

```
[project.entry-points."datalab.plugins"]
my-plugin = "my_plugin.plugin:MyPlugin"
```

La cible doit être une sous-classe de *datalab.plugins.PluginBase*. Il s'agit du mécanisme de distribution recommandé pour les plugins installés avec `pip`.

— Le répertoire des plugins utilisateur : Typiquement ~/.DataLab_v1/plugins sur Linux/macOS ou C:/Users/YourName/.DataLab_v1/plugins sur Windows.
— Répertoires de plugins supplémentaires : Tout nombre de répertoires supplémentaires peuvent être déclarés depuis Plugins > Configurer les plugins…, dans l'onglet Avancé (voir Onglet Avancé). Ils sont enregistrés dans la configuration de DataLab et analysés à chaque démarrage.
— Le répertoire de distribution autonome : Si vous utilisez une version gelée (autonome), le dossier plugins situé à côté de l'exécutable est analysé.
— Les plugins installés : wheels et modules installés depuis Plugins > Configurer les plugins…, soit depuis un fichier dans l'onglet Depuis un fichier (voir Onglet Depuis un fichier), soit depuis un catalogue de plugins dans l'onglet Catalogue (voir Onglet Catalogue).
— Le dossier interne datalab/plugins/builtin (non recommandé pour les plugins utilisateur) : Cet emplacement est réservé aux plugins intégrés ou fournis et ne doit pas être modifié manuellement.
— Répertoires supplémentaires listés dans la variable d'environnement `DATALAB_PLUGINS` : Un ou plusieurs répertoires peuvent être spécifiés, séparés par le séparateur de chemin du système d'exploitation (`;` sous Windows, `:` sous Linux/macOS), selon la même convention que `PYTHONPATH`. Tous les répertoires listés sont ajoutés aux chemins de recherche des plugins au démarrage ; les chemins inexistants sont ignorés silencieusement (un avertissement est consigné dans le fichier journal). Exemples :

```
# Linux/macOS
export DATALAB_PLUGINS="/opt/my_plugins:/home/alice/datalab_plugins"
```

```
:: Windows
set DATALAB_PLUGINS=C:\my_plugins;D:\shared\datalab_plugins
```

Les modifications apportées à cette variable ne sont prises en compte qu'au démarrage de DataLab.

Un répertoire de plugins peut aussi être le dossier d'un projet de plugin, comme ceux créés par `datalab-plugin create --project` : DataLab recherche alors aussi dans son sous-dossier `src`, qui contient le dossier `datalab_*` du plugin. Les paquets Python dont dépend le plugin doivent être disponibles dans l'environnement de DataLab.

Gestion des plugins dans DataLab

Le menu Plugins propose deux actions dédiées :

— Configurer les plugins… Ouvre la boîte de dialogue de configuration des plugins, organisée en quatre onglets : Installés, Catalogue, Depuis un fichier et Avancé. Le survol du titre d'un onglet décrit son contenu. La boîte de dialogue se rouvre avec la taille qu'elle avait à sa dernière fermeture.
— Recharger les plugins Recharge les modules de plugins depuis le disque sans redémarrer DataLab.

Lorsque les plugins sont globalement activés, un simple clic gauche sur le libellé Plugins : de la barre de statut ouvre également la même boîte de dialogue de configuration. L'icône n'est pas cliquable ; le raccourci reste disponible lorsqu'aucun plugin n'est chargé ou que certains plugins n'ont pas pu être chargés.

Onglet Installés

Cet onglet liste tous les plugins découverts avec leur icône, nom, version, description et chemin de fichier, ainsi que les plugins qui n'ont pas pu être importés. Pour chaque entrée :

— une case à cocher individuelle active ou désactive le plugin,
— Ouvrir le fichier ouvre la source du plugin dans votre éditeur,
— Afficher dans le dossier révèle le fichier du plugin dans le gestionnaire de fichiers du système.

Une case à cocher Activer tous les plugins et une boîte combinée Filtrer (Tous les plugins, Plugins activés, Plugins désactivés, Plugins avec erreurs) facilitent la navigation dans une grande collection de plugins. Un indicateur Dernier chargement : … montre quand les plugins ont été chargés pour la dernière fois pendant la session en cours, et le bouton Appliquer et recharger les plugins applique les modifications sans quitter la boîte de dialogue.

Onglet Catalogue

Cet onglet liste les plugins du catalogue de plugins DataLab et des catalogues locaux de plugins (voir Catalogues locaux de plugins), avec leur description, leur licence et un lien vers leur code source. Les plugins maintenus par l'équipe DataLab sont marqués Officiel ; les autres sont des plugins de la Communauté, maintenus par leurs auteurs, ou des plugins d'un Catalogue local, fournis par votre organisation (l'infobulle d'un tel plugin indique le dossier de son catalogue). Un champ de recherche filtre la liste.

Installer copie ou télécharge le wheel ou le module en fichier unique disponible pour Desktop et la version de Python utilisée, vérifie sa taille et son empreinte SHA-256, puis l'installe comme décrit dans Onglet Depuis un fichier, après confirmation. Les versions numérotées peuvent proposer Mettre à jour vers…. Pour un module sans version comparable, un contenu modifié propose plutôt Remplacer… ; une empreinte identique indique qu'il est déjà installé. Les plugins installés avec `pip` ou fournis par des répertoires sources non gérés sont identifiés, mais ne sont pas modifiés.

Les catalogues ne sont lus que lorsque l'onglet est affiché ou actualisé, ainsi que lorsque les dossiers des catalogues locaux changent. Un catalogue illisible, par exemple un partage réseau non connecté, est signalé sous le nombre de plugins sans masquer les plugins des autres catalogues. L'adresse du catalogue de plugins DataLab est l'option `plugins_catalog_url` du fichier de configuration de DataLab : elle peut désigner une copie privée du catalogue, ou être vidée pour le désactiver, par exemple sur des ordinateurs sans accès à Internet. Pour un catalogue stocké dans un dossier, y compris un dossier partagé sur le réseau, utilisez plutôt un catalogue local de plugins.

Catalogues locaux de plugins

Un catalogue local de plugins est un dossier, généralement partagé sur le réseau (par exemple `\\fileserver\datalab\plugins`), à partir duquel une organisation diffuse ses propres plugins, ou les versions de plugins publics qu'elle a validées, sans les publier. Les catalogues locaux complètent le catalogue de plugins DataLab et sont prioritaires sur lui : lorsque plusieurs catalogues proposent le même plugin (même nom de distribution ou même identifiant de plugin), seul le premier est affiché, afin que les utilisateurs installent la version choisie par l'organisation. Si le catalogue de plugins DataLab retire un plugin ainsi masqué, l'onglet le signale.

Les catalogues locaux sont déclarés :

- dans l'onglet Avancé (voir Onglet Avancé) : ces dossiers sont enregistrés dans la configuration de DataLab (option `plugins_catalog_path_list`) ;
- par la variable d'environnement `DATALAB_PLUGINS_CATALOG`, par exemple pour configurer tous les ordinateurs d'une organisation : les dossiers sont séparés par le séparateur de chemins du système (`;` sous Windows, `:` sous Linux/macOS) et sont prioritaires sur les dossiers de la configuration. Contrairement à `DATALAB_PLUGINS`, les dossiers indisponibles au démarrage de DataLab ne sont pas ignorés : ils sont signalés dans l'onglet Catalogue, et utilisés dès qu'ils deviennent disponibles et que les catalogues sont actualisés.

```
:: Windows
set DATALAB_PLUGINS_CATALOG=\\fileserver\datalab\plugins;D:\approved_plugins
```

Un dossier de catalogue local contient :

- des fichiers de plugins sans index : DataLab recherche les wheels en Python pur déclarant un point d'entrée `datalab.plugins` et les fichiers source nommés `datalab_<nom>.py` à la racine du dossier, sans parcourir les sous-dossiers. Les wheels sont regroupés par distribution et version ; chaque module constitue un fichier installable, même s'il déclare plusieurs classes de plugins. Les métadonnées des modules sont lues statiquement lorsque c'est possible ; un nom inconnu est remplacé par le nom du module et une version inconnue reste vide, avec un suivi par empreinte. Les fichiers invalides sont signalés. Copier un module modifié ou une nouvelle version d'un wheel suffit pour le proposer ;
- soit un fichier `catalog.json`, au format du catalogue de plugins DataLab, dont les URL des versions sont relatives au dossier : seuls les plugins qu'il liste sont affichés. Il peut s'agir d'une copie du catalogue de plugins DataLab avec ses wheels, ou d'un catalogue construit à partir des wheels du dossier avec `datalab-plugin catalog build`, puis modifié pour donner un nom d'affichage aux plugins, les déprécier ou les révoquer (champs `status` et `status_reason`), ou retirer une version (champ `yanked`).

Quel que soit le catalogue, DataLab vérifie la taille et l'empreinte SHA-256 du fichier avant son installation, l'inspecte sans l'exécuter et demande confirmation ; la boîte de dialogue indique le dossier du catalogue comme source. Pour un module, la validité du nom importable, l'encodage Python et la syntaxe sont vérifiés, avec une taille maximale de 4 Mio. L'inspection n'importe pas le code et ne certifie pas son comportement. L'organisation responsable d'un catalogue partagé doit en contrôler le contenu et limiter les droits d'écriture.

La commande `datalab-plugin` gère le fichier `catalog.json` d'un dossier :

```
:: Write catalog.json, listing the plugin files of the folder
datalab-plugin catalog build \\fileserver\datalab\plugins

:: Check catalog.json against the plugin files of the folder
datalab-plugin catalog check \\fileserver\datalab\plugins
```

`build` vérifie les wheels et les modules source sans les exécuter et n'écrit pas l'index si un fichier est refusé. Les wheels conservent leurs vérifications de Python pur, de points d'entrée et de dépendances ; les modules suivent les règles ci-dessous. Les métadonnées administrées, les motifs de retrait et les entrées révoquées sont conservés. Le fichier JSON est remplacé de manière atomique. `check` signale les fichiers de plugins et de dépendances manquants, modifiés, refusés ou non listés. Relancez `build` après modification des fichiers à la racine. Les deux commandes renvoient un code de sortie non nul si elles détectent un problème.

Modules en fichier unique dans un catalogue

Pour un catalogue hors ligne simple, placez le fichier `datalab_<nom>.py` inchangé à sa racine et ajoutez ce dossier dans Catalogues locaux de plugins. Aucun `pyproject.toml`, wheel ou accès à Internet n'est nécessaire. Pour générer un index administrable, exécutez `datalab-plugin catalog build <dossier>`, puis `check`. Un dossier mixte peut contenir des modules et des wheels :

```
approved_plugins\
    datalab_spectroscopy.py
    datalab_other_plugin-1.0-py3-none-any.whl
    catalog.json
    dependencies\
        additional_wheel_dependency-1.0-py3-none-any.whl
```

Le JSON généré utilise `kind: "module"` et le vrai nom d'importation dans `module` ; il n'attribue pas de distribution Python fictive au script. Sa version contient `filename`, une `url` relative, `size`, `sha256`, `targets: ["desktop"]` et `version` (éventuellement une chaîne vide). `kind` est omis ou vaut `"wheel"` pour les entrées wheel existantes. Ajoutez une contrainte facultative `requires_python` à une version de module lorsque sa compatibilité est connue.

Un administrateur peut modifier le nom d'affichage, le résumé, la licence et les autres champs descriptifs du JSON, et fournir une identité de catalogue et une version lorsqu'elles ne sont pas accessibles statiquement dans le code source. `build` conserve ces champs. Une identité ou une version connue dans le code source doit correspondre au catalogue. En présence de métadonnées dynamiques ou de plusieurs classes de plugins, le catalogue décrit le module complet sans l'exécuter. Son identité de catalogue ne réécrit pas le `PluginInfo` utilisé à l'exécution et n'impose pas de modifier un fichier historique.

Si un module n'a pas de version exploitable, laissez `version` vide : DataLab compare alors les empreintes SHA-256. La confirmation et la liste des fichiers installés distinguent une version connue d'une révision. Un fichier sans version modifié, ou un contenu différent portant la même version déclarée, demande un accord explicite via Remplacer… ; il n'est pas présenté comme une version plus récente. Les modules déjà installés manuellement par DataLab sont aussi reconnus. Les fichiers des répertoires de développement ne sont pas écrasés ; résolvez un conflit de nom ou retirez la copie non gérée avant d'utiliser l'installation gérée.

Les modules en fichier unique utilisent les paquets déjà disponibles dans DataLab. Leurs instructions `import` ne sont pas interprétées comme des exigences de paquets, aucune dépendance n'est téléchargée et un sous-dossier `dependencies`

n'est pas installé pour eux. `prepare` inclut les modules dans le catalogue, mais ne télécharge de dépendances que pour les wheels. Les importations qui ne peuvent pas être vérifiées statiquement peuvent encore échouer au chargement du plugin ; ces erreurs sont alors affichées dans l'onglet Installés.

Pour le catalogue public, publiez le fichier inchangé comme fichier joint nommé d'une release GitHub et indiquez sa version, sa licence, sa description et son SHA-256 dans la fiche YAML. Le code source n'est pas obligé d'exposer ces valeurs statiquement. Une fiche a la structure suivante (remplacez le dépôt, le fichier et l'empreinte d'exemple par les informations de la release réelle) :

```
id: io.github.alice.spectroscopy
name: Spectroscopy tools
kind: module
summary: A single-file Desktop plugin
license: BSD-3-Clause
repository: https://github.com/alice/spectroscopy
capabilities: [processing]
releases:
  - version: "1.0.0"
    filename: datalab_spectroscopy.py
    sha256: <SHA-256 of the released file>
    requires_python: ">=3.10"
```

Le catalogue public vérifie le script et le met en miroir dans `modules/<sha256>/...`. Pour utiliser ce catalogue publié hors ligne, copiez son JSON et toute son arborescence relative, y compris `modules` et `wheels` lorsqu'ils sont présents. Ne reconstruisez pas un miroir inchangé dont les fichiers se trouvent dans ces sous-dossiers : les commandes locales de préparation parcourent la racine. Les URL sources, les vérifications de révision, les retraits et la priorité des catalogues locaux suivent les mêmes règles pour les deux formats. Les modules ciblent uniquement Desktop, pas DataLab-Web ni PyPI.

Dépendances supplémentaires et installation hors ligne

Les plugins Desktop peuvent nécessiter des paquets qui ne sont pas fournis avec DataLab. Conservez le plugin lui-même sous forme de paquet wheel en Python pur et fournissez les paquets supplémentaires, y compris toutes leurs dépendances transitives, sous forme de wheels ordinaires dans un sous-dossier `dependencies` du catalogue local. Ces paquets de dépendances peuvent contenir des bibliothèques natives, à condition de correspondre à la version de Python, à l'ABI et à la plateforme de l'environnement hôte. Ils n'ont pas besoin de point d'entrée de plugin DataLab.

Sur un ordinateur disposant d'un accès à Internet, la commande de préparation constitue le dépôt de paquets wheel et écrit le catalogue :

```
datalab-plugin catalog prepare D:\approved_plugins
```

La commande télécharge les paquets wheel pour l'environnement courant sans les installer ni modifier ses paquets. Elle utilise les dépendances d'exécution déclarées de DataLab comme référence : les plugins sans rapport déjà installés sur l'ordinateur de l'auteur ne sont donc pas confondus avec les paquets livrés avec DataLab. Préparez les paquets dans chaque environnement cible lorsque des versions de Python ou des plateformes différentes nécessitent des fichiers différents, puis copiez le dossier complet sur l'ordinateur hors ligne ou le partage réseau. Les commandes `build` et `check` ne téléchargent pas elles-mêmes les paquets manquants.

Par exemple, un catalogue pour CPython 3.11 sous Windows peut contenir :

```
approved_plugins\
    catalog.json
    datalab_tango-0.1.0-py3-none-any.whl
    dependencies\
```

(suite sur la page suivante)

(suite de la page précédente)

```
        pytango-10.3.1-cp311-cp311-win_amd64.whl
        docstring_parser-0.18.0-py3-none-any.whl
```

L'onglet Catalogue copie les fichiers de dépendances vérifiés de la version choisie. Pour Installer depuis un fichier…, placez le dossier `dependencies` à côté du paquet wheel du plugin. DataLab résout l'installation à partir de ces seuls fichiers locaux, sans repli vers un index de paquets et sans compilation. Avant confirmation, il affiche les paquets supplémentaires, leurs versions, leurs tailles, le dossier source et leurs empreintes SHA-256.

Les paquets déjà fournis par l'environnement Python ou par DataLab ne sont jamais remplacés, mis à jour ou rétrogradés. Les paquets supplémentaires sont partagés entre les plugins d'un même environnement d'exécution Python. Une version installée incompatible, un paquet transitif manquant ou un paquet natif non pris en charge entraîne un refus explicite ; aucune installation partielle n'est activée. Les vérifications des métadonnées du catalogue ne remplacent pas cette résolution dans l'environnement hôte réel.

Une installation qui ajoute des dépendances est activée après le redémarrage de DataLab. Un plugin nouvellement installé est alors activé, y compris lorsqu'une liste explicite de plugins autorisés est configurée. Désinstaller le plugin conserve les dépendances partagées. Les environnements Python/ABI/plateforme différents utilisent des répertoires de dépendances séparés ; les dépendances incompatibles avec un environnement hôte modifié sont signalées plutôt que réparées par téléchargement.

Il n'y a pas d'environnement par plugin : des plugins exigeant des versions contradictoires ne peuvent pas coexister. Les paquets nécessitant des bibliothèques système ou des pilotes externes exigent toujours l'installation séparée de ces prérequis. Dans l'application autonome, les dépendances sont destinées à être importées comme bibliothèques ; elles ne fournissent pas d'interpréteur Python externe et le fonctionnement indépendant de leurs scripts en ligne de commande n'est pas garanti. L'installateur Web reste inchangé.

Sous Windows, les chemins des bibliothèques natives doivent également respecter la limite du chargeur de DLL. DataLab utilise des noms courts pour les répertoires de générations et refuse une installation dont les chemins de DLL resteraient trop longs, avant de la publier ; utilisez alors un répertoire de configuration dont le chemin est plus court.

Déploiement hors ligne, pas à pas

Cette procédure est autonome : l'ordinateur déconnecté n'a besoin ni d'un index de paquets, ni d'un environnement de développement Python, ni d'un accès au site public des plugins. Un partage réseau local, une clé USB ou un simple dossier local convient. Préparer les dépendances manquantes des wheels nécessite un ordinateur de référence connecté ; un catalogue ne contenant que des modules source peut être constitué entièrement hors ligne (voir Modules en fichier unique dans un catalogue).

> **Important**
>
> Un catalogue de plugins est une source de fichiers d'installation, pas un répertoire de recherche de plugins. Ajoutez-le dans Catalogues locaux de plugins, et non dans Répertoires supplémentaires de plugins. `DATALAB_PLUGINS_CATALOG` indique des catalogues ; `DATALAB_PLUGINS` indique du code chargé directement. L'installation copie les fichiers approuvés dans le répertoire de configuration DataLab de l'utilisateur : le dossier source ou le partage réseau n'est donc plus nécessaire pour exécuter ensuite le plugin installé.

1. Préparer un catalogue complet sur un ordinateur connecté

Le fournisseur doit disposer de DataLab installé dans un environnement Python, avec la commande `datalab-plugin` disponible. L'application autonome ne fournit pas cette commande de préparation ; ses utilisateurs n'ont besoin que des fichiers préparés. Utilisez un environnement de référence correspondant à la version de DataLab, à la version de Python, au système d'exploitation, à l'architecture et aux versions des paquets de l'ordinateur cible. **?** > Installation et configuration… permet d'identifier l'installation cible. Un paquet pour CPython 3.11 sous Windows 64 bits n'est pas interchangeable avec un paquet pour Linux, une architecture 32 bits ou CPython 3.12.

Créez un dossier, placez le fichier `.whl` ou `datalab_*.py` du plugin à sa racine, puis exécutez :

```
datalab-plugin catalog prepare D:\approved_plugins
datalab-plugin catalog check D:\approved_plugins
```

Pour les wheels, `prepare` résout les exigences manquantes par rapport à l'environnement hôte de référence, télécharge leurs wheels et dépendances transitives dans `dependencies`, puis écrit `catalog.json`. La commande ne les installe pas dans l'environnement de référence et ne compile pas de distributions source. Les modules sont inclus sans téléchargement de dépendances. Utilisez `build` pour un catalogue reposant uniquement sur des fichiers ; si la préparation signale un wheel manquant ou un conflit, corrigez le catalogue avant de le transférer.

Conservez tous les fichiers produits, y compris les paquets de dépendances. Ne copiez pas uniquement `catalog.json` ou uniquement le paquet wheel du plugin. Pour desservir plusieurs environnements cibles, préparez leurs paquets dans des environnements de référence correspondants, rassemblez les fichiers nécessaires, puis exécutez `datalab-plugin catalog build D:\approved_plugins`, suivi de `check`. Un paquet en Python pur peut lui aussi restreindre les versions de Python prises en charge dans ses métadonnées.

`check` vérifie les fichiers du catalogue, leur intégrité et leurs métadonnées. Cette commande ne remplace pas un essai d'installation sur la cible réelle : la résolution complète des dépendances se fait lors de l'installation. Avant un déploiement plus large, testez le dossier complet sur une installation équivalente dont l'accès à Internet est indisponible.

2. Transférer le dossier en conservant son organisation

Copiez le dossier complet sur une clé USB, un disque local ou un partage réseau accessible depuis l'ordinateur cible. Conservez les chemins relatifs et les noms de fichiers. Un miroir public peut stocker les fichiers dans `wheels\<digest>\...` et `modules\<digest>\...` : conservez aussi ces répertoires, car `catalog.json` les référence. Le seul fichier JSON n'est pas un miroir et DataLab n'exporte pas automatiquement un catalogue en ligne pour un usage hors ligne.

Pour un catalogue JSON strictement hors ligne, l'URL d'installation (champ `url`) de chaque version et de chaque fichier de dépendance doit désigner un fichier local existant. Une URL absolue `https://` entraîne toujours un téléchargement depuis son serveur, même si le catalogue est local ; placer un paquet wheel à côté de ce JSON ne remplace pas son URL. Les liens informatifs tels que `repository`, `documentation` et `source_url` ne remplacent pas les fichiers d'installation. Les catalogues générés par `build` utilisent des URL de fichiers relatives ; les miroirs publiés doivent conserver l'arborescence relative correspondante.

L'utilisateur doit disposer d'un accès en lecture à cette source, et d'un accès en écriture à ses propres répertoires de configuration et de fichiers temporaires. Les privilèges d'administrateur ne sont pas nécessaires pour installer un plugin de cette manière. Réservez l'accès en écriture au catalogue partagé à ses responsables ; une vérification SHA-256 détecte les fichiers modifiés, mais ne certifie pas l'innocuité d'un plugin ou d'une dépendance.

3. Configurer l'ordinateur déconnecté

Ouvrez Plugins > Configurer les plugins… > Avancé. Dans Catalogues locaux de plugins, cliquez sur Ajouter, sélectionnez la racine du catalogue (le dossier contenant `catalog.json` ou les paquets des plugins), puis appliquez la configuration. Ne sélectionnez pas le sous-dossier `dependencies`. Ouvrez l'onglet Catalogue et cliquez sur Actualiser si nécessaire.

Les administrateurs peuvent aussi définir la variable d'environnement suivante avant de lancer DataLab :

```
$env:DATALAB_PLUGINS_CATALOG = 'D:\approved_plugins;\\fileserver\datalab\plugins'
```

Cet exemple s'applique à DataLab démarré depuis la même fenêtre PowerShell ; un raccourci lancé séparément n'hérite pas de l'environnement de cette fenêtre. Une organisation peut également déployer la variable par son mécanisme habituel de configuration des postes. Les dossiers de la variable d'environnement sont prioritaires sur ceux enregistrés dans les paramètres de DataLab ; les deux sont prioritaires sur le catalogue en ligne. Le premier identifiant de plugin ou nom de distribution correspondant l'emporte : les versions de sources différentes ne sont pas fusionnées.

> **Avertissement**
>
> Ajouter un catalogue local ne désactive pas le catalogue en ligne. Si son URL reste configurée, DataLab tente toujours de le lire lors de l'actualisation, ce qui peut entraîner une attente réseau et une erreur sur un ordinateur déconnecté. Les autres catalogues restent utilisables après cette erreur.

Pour désactiver cette recherche, fermez DataLab, sauvegardez son fichier de configuration et donnez une valeur vide à `plugins_catalog_url` dans la section `[main]` existante. Avec le profil utilisateur Windows standard, DataLab 1.3 et les versions suivantes utilisent `%USERPROFILE%\.DataLab_v1\DataLab_v1_typed.ini` (sous Linux/macOS, normalement `~/.DataLab_v1/DataLab_v1_typed.ini`). Les profils personnalisés peuvent utiliser un autre dossier de base. Ne modifiez pas à sa place l'ancien fichier `DataLab_v1.ini`.

```
[main]
plugins_catalog_url =
```

Conservez le reste du fichier. Ne supprimez pas cette option pour la désactiver : une option absente rétablit l'URL publique par défaut. Redémarrez DataLab après modification. Une URL HTTPS privée nécessite toujours un accès à son serveur ; pour une installation hors ligne reposant uniquement sur des fichiers, utilisez un dossier de catalogue local.

4. Installer, confirmer et redémarrer si nécessaire

Dans l'onglet Catalogue, trouvez le plugin et cliquez sur Installer. DataLab choisit la version la plus récente proposée pour Desktop et pour la version de Python utilisée, vérifie la taille et les empreintes SHA-256 du plugin et des fichiers de dépendances compatibles listés, puis prépare toute installation supplémentaire à partir des seuls fichiers locaux. La sélection d'une version ne garantit pas que toutes ses dépendances puissent être installées : ce point est vérifié dans l'environnement hôte réel avant confirmation.

Lisez la confirmation : elle indique l'identité du plugin, ses classes, ses exigences et son empreinte, ainsi que les paquets supplémentaires, leurs versions, leurs tailles, leurs empreintes et le dossier source si nécessaire. Annuler laisse l'installation inchangée. Après accord, DataLab installe les fichiers dans son stockage utilisateur privé ; il ne modifie pas les versions des paquets de l'environnement et ne recharge pas le code depuis le catalogue partagé à chaque démarrage.

Redémarrez DataLab si des dépendances ont été ajoutées ou si vous remplacez un paquet déjà chargé. Sinon, DataLab peut proposer de recharger les plugins. Après redémarrage, vérifiez l'état du plugin et les éventuelles erreurs d'importation dans l'onglet Installés, puis essayez l'une de ses actions. Le catalogue source peut être déconnecté après une installation réussie. Pour installer un fichier ponctuellement, Depuis un fichier > Installer depuis un fichier… effectue les mêmes vérifications ; conservez `dependencies` à côté du paquet wheel du plugin.

5. Maintenir et dépanner le catalogue hors ligne

Lorsque `catalog.json` existe, il fait autorité : DataLab ne liste et ne copie que les versions et fichiers de dépendances qui y sont déclarés. Pour un catalogue géré à partir de fichiers à sa racine, relancez `build` et `check` après ajout ou modification d'un module, d'un wheel de plugin ou de dépendance, puis redistribuez le dossier complet. Copier un nouveau fichier à côté d'un JSON existant ne met pas cet index à jour. `prepare` peut nécessiter Internet pour les dépendances des wheels ; `build` et `check` ne récupèrent jamais les paquets manquants.

N'exécutez pas `build` sur un miroir inchangé dont les plugins se trouvent uniquement dans `wheels\<digest>\...` ou `modules\<digest>\...` : la commande parcourt la racine du catalogue, pas l'arborescence récursive, et supprimerait les entrées non représentées. Mettez à jour un miroir en transférant un catalogue publié complet et cohérent avec ses fichiers. Pour le gérer avec les commandes locales de préparation, disposez d'abord les wheels et modules des plugins à la racine et les paquets supplémentaires dans `dependencies`.

Les utilisateurs actualisent l'onglet Catalogue pour découvrir les mises à jour, puis cliquent explicitement sur Mettre à jour vers…. Les mises à jour ne sont pas installées automatiquement. Les plugins installés avec `pip` dans l'environnement

Python sont identifiés, mais ne sont pas mis à jour par cette boîte de dialogue. Les dépendances supplémentaires partagées sont conservées lorsqu'un plugin est désinstallé, et leurs versions existantes restent protégées lors des installations suivantes.

Si une installation échoue, lisez l'erreur dans la boîte de dialogue des plugins avant de modifier les fichiers :

- Catalogue introuvable ou inaccessible : vérifiez la racine sélectionnée, la lettre de lecteur, la clé USB, la connexion réseau et les droits de lecture. Un partage manquant n'est pas un problème d'accès à Internet. Actualisez lorsqu'il est de nouveau disponible.
- Taille ou SHA-256 incorrect : les fichiers copiés diffèrent du catalogue. Demandez au fournisseur une copie complète et cohérente ou régénérez le catalogue à partir des paquets souhaités ; ne contournez pas cette vérification.
- Aucun paquet compatible ou exigence manquante : le dépôt peut ne pas contenir une dépendance transitive ou le paquet correspondant à ce Python, à cette ABI ou à cette plateforme. Demandez au fournisseur d'exécuter `prepare` dans un environnement de référence correspondant. L'installateur ne se replie jamais vers PyPI.
- Version installée incompatible : l'environnement hôte ou un autre plugin fournit déjà une version de paquet en conflit. Demandez une version ou une installation de référence compatible ; ne forcez pas de mise à jour ou de rétrogradation pour contourner le refus.
- Chemin de DLL Windows trop long : utilisez un chemin de configuration plus court. Erreur d'importation de DLL ou bibliothèque système manquante : vérifiez les prérequis externes du paquet natif avec le fournisseur ; DataLab n'installe ni pilotes ni paquets système.
- Installé mais inactif : redémarrez si cela est demandé, vérifiez l'activation globale et individuelle des plugins et consultez l'erreur d'importation affichée dans l'onglet Installés. Un simple rechargement ne remplace pas une bibliothèque native déjà importée.

Pour l'aide locale, utilisez **?** > Documentation PDF, puis recherchez Déploiement hors ligne, pas à pas ou `DATALAB_PLUGINS_CATALOG`. Ce PDF est un fichier local et ne nécessite pas Internet. Documentation en ligne, le lien vers la documentation de la page d'accueil et les liens externes vers le code source ou la documentation ouvrent, eux, des sites web. L'action de menu du PDF n'apparaît que si un PDF local est présent ; DataLab essaie d'abord la langue du système, puis l'anglais. S'il est absent ou illisible, demandez à l'administrateur le PDF livré avec la version correspondante de DataLab plutôt que de dépendre d'un lien en ligne.

Onglet Depuis un fichier

Cet onglet installe un plugin partagé sous forme de fichier, sans aucun outil Python : c'est ainsi que l'on ajoute des plugins à la version autonome de DataLab. Installer depuis un fichier… accepte deux types de fichiers :

- un paquet wheel (`.whl`) construit à partir d'un projet de plugin, par exemple avec `python -m build` : il doit être en Python pur (`py3-none-any`), déclarer sa classe de plugin dans le groupe de points d'entrée `datalab.plugins`, et ses exigences doivent être satisfaites par l'environnement hôte ou le dépôt de dépendances adjacent (voir Dépendances supplémentaires et installation hors ligne) ;
- un module Python unique nommé `datalab_<nom>.py`.

Avant l'installation, DataLab vérifie le fichier sans l'exécuter. Pour un wheel, la confirmation affiche la distribution, la version, les points d'entrée et les exigences issus de ses métadonnées. Pour un module en fichier unique, elle affiche son nom d'importation, le fichier, la taille, le SHA-256 et les métadonnées lisibles statiquement ou explicites du catalogue. Une version inconnue reste inconnue et est affichée comme une révision ; aucune liste de classes ou de dépendances n'est devinée. Un plugin s'exécute avec les mêmes droits que DataLab : n'installez que des plugins provenant d'auteurs de confiance.

Les plugins installés sont conservés dans le dossier `installed_plugins` du répertoire de configuration DataLab (par exemple `~/.DataLab_v1/installed_plugins`) et listés dans l'onglet, où Désinstaller les supprime. Les modules peuvent être rechargés après installation ou remplacement ; un plugin nouvellement installé est activé. Les dépendances supplémentaires et les nouvelles versions de wheels déjà chargés ne sont activées qu'au prochain démarrage de DataLab.

Lors du rechargement des plugins, DataLab effectue les étapes suivantes :

1. Désenregistrer les plugins actuellement actifs et leurs traitements,
2. Effacer les actions de plugins des panneaux signal et image,
3. Redécouvrir et recharger les modules de plugins,
4. Réenregistrer les plugins activés,
5. Réenregistrer les traitements possédés,

6. Recréer les actions des plugins et rafraîchir les menus.

Ce flux permet un développement itératif des plugins pendant que DataLab est en cours d'exécution.

> **Note**
>
> L'état d'activation/désactivation des plugins est conservé dans les paramètres de DataLab. Les plugins désactivés restent listés dans la boîte de dialogue de configuration et peuvent être réactivés plus tard. Le réglage global des plugins tiers dans les préférences est lui aussi appliqué immédiatement : le désactiver supprime les actions des plugins et désactive l'indicateur d'état, qui affiche Plugins : -, tandis que le réactiver recharge automatiquement les plugins. L'action du menu Plugins > Configurer les plugins… reste accessible lorsque l'indicateur d'état est désactivé.

Onglet Avancé

Cet onglet liste tous les répertoires scannés au démarrage, en deux groupes :

— Répertoires de plugins par défaut (lecture seule) : le répertoire de plugins utilisateur, le dossier datalab/plugins/builtin inclus, le dossier de distribution autonome le cas échéant, et les répertoires déclarés via la variable d'environnement `DATALAB_PLUGINS` (identifiés par un suffixe *(from* DATALAB_PLUGINS)).

— Répertoires de plugins supplémentaires : vos propres répertoires, ajoutés avec le bouton Ajouter et gérés avec les boutons Modifier le répertoire et Supprimer le répertoire. Contrairement à la variable d'environnement, ces répertoires sont enregistrés dans la configuration de DataLab et persistent donc entre les sessions.

Il liste aussi les Catalogues locaux de plugins : des dossiers, éventuellement partagés sur le réseau, dont les plugins sont proposés dans l'onglet Catalogue (voir Catalogues locaux de plugins). Ils se gèrent avec les mêmes boutons, et les dossiers déclarés par la variable d'environnement `DATALAB_PLUGINS_CATALOG` apparaissent en premier, en lecture seule. Modifier ces dossiers ne nécessite pas de recharger les plugins.

L'onglet contient également une option Avertissements de compatibilité pour masquer les avertissements concernant les plugins DataLab v0.20 incompatibles.

Rechargement à chaud pour le développement de plugins

La fonctionnalité de rechargement à chaud est conçue pour accélérer le cycle de développement des plugins. Voici le workflow recommandé :

1. Démarrer DataLab normalement.
2. Créer ou modifier votre fichier plugin (par exemple `datalab_myplugin.py`) dans l'un des répertoires de plugins (par exemple `~/.DataLab_v1/plugins`).
3. Dans DataLab, utiliser Plugins > Recharger les plugins pour prendre en compte les modifications instantanément.
4. Tester les actions du plugin directement dans l'application en cours d'exécution.
5. Itérer : modifier le fichier, recharger, tester — sans redémarrer DataLab.

Pour activer ou désactiver sélectivement des plugins pendant le développement, utiliser Plugins > Configurer les plugins… et son bouton Appliquer et recharger les plugins, afin que les modifications prennent effet sans quitter la boîte de dialogue.

Aides de l'API plugin

Les plugins qui héritent de *`datalab.plugins.PluginBase`* ont un accès direct à des aides utiles :

— `self.signalpanel` et `self.imagepanel` : accès aux API de panneaux et aux gestionnaires d'actions,

— `self.proxy` : une instance de *`datalab.control.proxy.LocalProxy`* pour créer des objets et lancer des traitements,

— `show_warning`, `show_error`, `show_info`, `ask_yesno` : méthodes pratiques pour les boîtes de dialogue,

— `edit_new_signal_parameters` et `edit_new_image_parameters` : aides pour les boîtes de dialogue de paramètres d'objet.

Ces aides simplifient le code des plugins et le maintiennent cohérent avec le comportement de DataLab.

Les plugins de traitement doivent surcharger `register_computations()` et enregistrer chaque fonctionnalité avec un `feature_id` stable de la forme `<plugin_id>:<local_id>` et `owner_plugin_id=self.plugin_id`. DataLab appelle cette méthode une fois les panneaux Signal et Image créés. Les fonctionnalités détenues par le plugin sont supprimées automatiquement lorsque celui-ci est désactivé, rechargé ou désinstallé. `create_actions()` peut ensuite faire référence à la fonctionnalité enregistrée par son identifiant stable.

Une fonctionnalité enregistrée exécute une fonction de calcul Sigima, décorée par `sigima.proc.decorator.computation_function()`. Pour une fonctionnalité 1 vers 1 sans paramètre, le plus simple est d'envelopper une fonction traitant des tableaux NumPy avec `sigima.proc.signal.Wrap1to1Func` (valeurs Y d'un signal) ou `sigima.proc.image.Wrap1to1Func` (données d'une image), comme dans l'exemple de plugin de traitement (voir Exemple : plugin de traitement). DataLab refuse les autres fonctions dès leur enregistrement.

Recettes sans interface graphique

Les plugins peuvent exposer des workflows scientifiques versionnés via le tuple de classe `RECIPES`. Chaque *`datalab.plugins.recipes.RecipeDescriptor`* déclare un ID stable dans l'espace de noms du plugin, des emplacements d'entrée typés, une classe de paramètres guidata `DataSet` facultative et une fonction appelable sans interface graphique :

Les descripteurs de recettes restent détenus par la classe de plugin courante et sont lus via *`datalab.plugins.PluginBase.get_recipes()`* ; ils ne sont pas copiés dans un registre mutable distinct. Le rechargement de la classe de plugin remplace donc ses déclarations de recettes en même temps que son implémentation.

```
from datalab.plugins import PluginBase, PluginInfo
from datalab.plugins.recipes import (
  RecipeDescriptor,
  RecipeInputSlot,
  RecipeObjectOutput,
  RecipeOutcome,
)

def run_quick_check(inputs, parameters, context):
  source = inputs["source"][0]
  context.raise_if_cancelled()
  output = source.copy()
  context.report_progress(1.0, "Quick check complete")
  return RecipeOutcome(
    objects=(RecipeObjectOutput("checked-signal", output),),
  )

class MyPlugin(PluginBase):
  PLUGIN_INFO = PluginInfo(
    id="org.example.my-plugin",
    name="My Plugin",
    version="1.0.0",
  )
  RECIPES = (
    RecipeDescriptor(
      recipe_id="org.example.my-plugin:quick-check",
      plugin_version="1.0.0",
      title="Quick check",
      version="1.0.0",
      inputs=(RecipeInputSlot("source", "signal", "one"),),
      run=run_quick_check,
```

(suite sur la page suivante)

(suite de la page précédente)

```
    ),
  )

  def create_actions(self):
    pass
```

La valeur `plugin_version` du descripteur doit correspondre à `PLUGIN_INFO.version`. L'explicitation des versions du plugin et de la recette permet aux traces d'exécution d'identifier à la fois l'implémentation installée et le workflow versionné indépendamment.

DataLab valide les déclarations d'un plugin lors de son chargement : recettes, lanceurs de recettes, exemples, outils et tuiles de la page d'accueil, décrits ci-dessous. Un plugin dont une déclaration est invalide, par exemple une recette dont le `plugin_version` ne correspond plus à la version du plugin, n'est pas chargé : son erreur est affichée dans la console interne et consignée dans le fichier journal, et le plugin est listé avec cette erreur dans l'onglet Installés de Configurer les plugins… (filtre Plugins avec erreurs). Les autres plugins sont chargés normalement.

La fonction de la recette reçoit une correspondance entre les ID d'emplacements et des tuples d'instances Sigima `SignalObj` ou `ImageObj`, l'instance de paramètres `DataSet` (ou `None`) et un *`datalab.plugins.recipes.RecipeExecutionContext`*. Le contexte fournit des fonctions de rappel de progression et d'annulation indépendantes de la technologie et ne dépend pas de Qt.

Lorsque `parameter_class` vaut `None`, la recette ne déclare aucun paramètre et la fonction reçoit `None`. Dans le cas contraire, les consommateurs doivent fournir une instance de la sous-classe `DataSet` déclarée. Le runner de recettes applique cette règle avant l'exécution.

Un *`datalab.plugins.recipes.RecipeOutcome`* contient des sorties d'objets nommées, des diagnostics structurés et des résultats scalaires facultatifs. Un `TableResult` ou `GeometryResult` est encapsulé dans un *`datalab.plugins.recipes.RecipeResultOutput`* et utilise `anchor_id` pour référencer la sortie d'objet nommée qui en sera propriétaire. Le contrat valide ces références sans attribuer d'UUID du workspace DataLab. La mutation du workspace et le commit atomique relèvent du runner de recettes, pas de la fonction de la recette.

Décrire les entrées d'une recette

DataLab montre aux utilisateurs ce qu'attend chaque recette et si la sélection courante lui convient. Décrivez chaque emplacement d'entrée en conséquence :

```
from datalab.plugins.recipes import RecipeInputSlot, RecipeMetadataRequirement

FLAT_FRAMES = RecipeInputSlot(
  "flat_frames",
  "image",
  "many",
  title="Flat frames",
  description="Uniformly illuminated frames, two or more per exposure level",
  min_count=4,
  metadata=(
    RecipeMetadataRequirement(
      "org.example.my-plugin.exposure_time_s", "Exposure time in seconds"
    ),
    RecipeMetadataRequirement(
      "org.example.my-plugin.frame_role", "dark or flat", required=False
    ),
  ),
)
```

`min_count` est le nombre minimal d'objets affectés à un emplacement `many`. Les clés de métadonnées requises doivent être présentes sur chaque objet affecté ; les clés facultatives sont des indications montrées à l'utilisateur. L'exécuteur de recettes applique ces deux règles avant d'appeler la fonction de la recette.

Deux points d'extension facultatifs de *datalab.plugins.recipes.RecipeDescriptor* affinent ces déclarations :

- `suggest_bindings(candidates)` renvoie une correspondance entre identifiants d'emplacements et une partie des objets candidats, par exemple pour répartir des images selon leur rôle à partir de leurs métadonnées. Sans lui, les candidats compatibles vont au seul emplacement acceptant leur type, et les emplacements partageant un même type sont laissés à l'affectation de l'utilisateur.
- `check_inputs(inputs, parameters)` renvoie une liste de *datalab.plugins.recipes.RecipeDiagnostic* pour des vérifications rapides propres à la recette, comme les dimensions, les effectifs par niveau ou les unités. Il ne doit pas calculer de résultats. Les diagnostics d'erreur bloquent l'exécution et les avertissements sont affichés avant celle-ci. Une exception levée par ce point d'extension est signalée comme une erreur `input-check-failed`.

`datalab.plugins.recipe_binding.assess_recipe_inputs()` combine les déclarations des emplacements et ces deux points d'extension. Elle renvoie un `datalab.plugins.recipe_binding.RecipeReadiness` comprenant un statut (`no_input`, `needs_assignment`, `not_ready`, `warnings` ou `ready`), les affectations proposées, des problèmes structurés que les hôtes traduisent, et les diagnostics de la recette. Le module ne dépend pas de Qt : DataLab-Web utilise donc le même code.

Exécuter une recette sur Desktop

La plupart des plugins n'appellent pas directement l'exécuteur. *datalab.plugins.PluginBase.start_recipe()* exécute une recette sur la sélection courante avec le lanceur générique de DataLab. Il évalue la sélection, n'ouvre une boîte de dialogue d'affectation des entrées que si les affectations sont ambiguës ou invalides, édite les paramètres déclarés, puis appelle l'exécuteur. Une action de menu de plugin se résume donc à :

```
def run_quick_check(self):
  return self.start_recipe("org.example.my-plugin:quick-check")
```

Les boutons Exécuter… du Panneau Applications utilisent le même lanceur. Un plugin qui a besoin d'une interface dédiée pour une recette peut associer son identifiant de recette au nom d'une méthode du plugin dans la correspondance de classe `RECIPE_LAUNCHERS` ; le panneau appelle alors cette méthode, sans argument, à la place du lanceur générique.

Utilisez *datalab.gui.plugins.recipe_runner.RecipeRunner* depuis le thread de l'interface graphique pour exécuter une recette sur des entrées explicites. Il valide les entrées et les paramètres, exécute la fonction sans interface et committe son résultat dans le workspace DataLab :

```
from datalab.gui.plugins.recipe_runner import RecipeRunner

descriptor = self.get_recipes()[0]
outcome = RecipeRunner(self.main).run(
  descriptor,
  inputs={"source": (source_signal,)},
)
```

Le runner rejette les emplacements d'entrée manquants, supplémentaires, mal typés ou de cardinalité incorrecte avant d'appeler le code de la recette. Il applique également `min_count`, les métadonnées requises et les diagnostics d'erreur de `check_inputs`, valide l'instance `DataSet` facultative et vérifie l'annulation avant l'exécution, après l'exécution et juste avant le commit. Le code de la recette doit rester sans interface et ne doit pas modifier lui-même le workspace.

Seul un *datalab.plugins.recipes.RecipeOutcome* validé atteint la phase de commit. Le runner Desktop crée un groupe par panneau de sortie, en utilisant par défaut le titre de la recette, puis ajoute toutes les sorties signal et image. Les ID des résultats scalaires sont conservés sous forme de noms de fonction `<recipe-id>:<result-id>` sur leurs objets ancres nommés, afin que plusieurs tables ou géométries coexistent sans collision de clés de métadonnées. Un échec

pendant l'insertion supprime chaque objet et groupe créé par cette invocation, puis restaure l'état modifié précédent du workspace et le panneau courant.

Provenance locale de l'exécution

Après une exécution sans interface réussie, le runner Desktop enregistre le même *datalab.plugins.recipes.RecipeRunRecord* dans les métadonnées de chaque objet de sortie. Son contenu versionné et compatible JSON comprend un UUID d'exécution partagé, les ID et versions du plugin et de la recette, les paramètres résolus au format JSON, les UUID nommés des entrées et sorties, les versions de DataLab et Sigima, le statut terminé ainsi que les horodatages UTC de début et de fin.

Les métadonnées sont conservées lors de l'export JSON et des allers-retours des workspaces HDF5. L'UUID d'exécution partagé relie les sorties entre les panneaux Signal et Image sans ajouter de métadonnées aux groupes ni d'historique global de workflow. Les invocations en échec ou annulées ne committent aucune sortie et ne laissent donc aucune trace d'exécution dans le workspace.

Exemples packagés

Les plugins peuvent exposer des workspaces DataLab natifs via le tuple de classe `EXAMPLES`. Chaque *datalab.plugins.examples.PluginExample* utilise une ressource `package:relative/path` plutôt qu'un chemin de développement du système de fichiers :

```
from datalab.plugins.examples import PluginExample

class MyPlugin(PluginBase):
  EXAMPLES = (
    PluginExample(
      id="quickstart",
      title="Quick start",
      description="Small deterministic workspace",
      resource="datalab_my_plugin:examples/quickstart.h5",
      recipe_ids=("org.example.my-plugin:quick-check",),
      expected_checks=("summary-table",),
    ),
  )

  def create_actions(self):
    pass
```

Le package doit inclure la ressource dans sa wheel. `resolve()` renvoie un objet traversable `importlib.resources` qui fonctionne également pour les packages importés depuis une archive ZIP. Pour les API exigeant un chemin du système de fichiers, utilisez `as_file()` comme gestionnaire de contexte et ne conservez pas le chemin renvoyé après la sortie du contexte.

datalab.plugins.PluginBase.get_examples() valide l'unicité des ID locaux et les références aux recettes. Un plugin enregistré peut appeler `open_example("quickstart")` pour matérialiser et charger un workspace HDF5 DataLab natif. L'ouverture efface le workspace courant par défaut ; passez `reset_all=False` pour le fusionner à la place.

`recipe_ids` liste les recettes pour lesquelles un exemple est conçu. Un exemple peut servir plusieurs recettes, par exemple une rampe d'obscurité analysée par deux méthodes. Un plugin peut aussi générer un exemple en mémoire en surchargeant `materialize_example()` pour renvoyer un *datalab.plugins.examples.PluginExampleData* ; sa correspondance `parameter_values` donne, pour chaque identifiant de recette, les valeurs de paramètres adaptées aux objets générés.

Le Panneau Applications présente chaque recette comme une méthode, avec un statut mis à jour pour la sélection courante et un bouton Exécuter… ; ses détails donnent les entrées attendues et les exemples conçus pour elle. Essayer avec cet

exemple appelle *datalab.plugins.PluginBase.try_example()* : il ouvre l'exemple, préremplit les paramètres de la recette avec les valeurs de l'exemple et exécute la recette une fois qu'ils sont acceptés par l'utilisateur. Les exemples sans `recipe_ids` sont listés comme jeux de données, qui s'ouvrent seulement.

Outils de plugin

Une recette couvre une analyse sans interface exécutée par DataLab. Toute autre interaction propre à un plugin d'application, comme un assistant, un éditeur interactif, une étape de préparation des données ou un instrument simulé, est un outil. Déclarez les outils dans le tuple de classe `TOOLS` de valeurs *datalab.plugins.tools.PluginTool* :

```
from datalab.plugins.recipes import RecipeObjectType
from datalab.plugins.tools import PluginTool, ToolSelection

class MyPlugin(PluginBase):
  TOOLS = (
    PluginTool(
      id="annotate",
      title="Annotate frames...",
      launcher="annotate_frames",
      description="Set the acquisition metadata of the selected frames",
      object_type=RecipeObjectType.IMAGE,
      selection=ToolSelection.AT_LEAST_ONE,
    ),
  )

  def annotate_frames(self):
    ...
```

DataLab liste chaque outil à deux endroits : dans la section Outils du Panneau Applications, et dans le sous-menu du plugin du menu Plugins, après les actions propres au plugin. `object_type` indique le panneau dont le menu affiche l'outil ; avec `None`, les deux panneaux l'affichent. `selection` indique la sélection dont l'outil a besoin : `NONE` (par défaut), `EXACTLY_ONE`, `AT_LEAST_ONE` ou `AT_LEAST_TWO` objets de type `object_type`. Lorsque la sélection ne convient pas, DataLab désactive l'entrée de menu et le bouton du panneau Applications, et l'info-bulle du bouton indique quoi sélectionner. `icon` accepte une ressource `package:path`, une image `module:filename` enregistrée auprès de guidata (par exemple `datalab:delete.svg`) ou un nom de fichier d'icône DataLab ; par défaut, l'outil utilise l'icône du plugin.

Le lanceur est appelé sans argument. *datalab.plugins.PluginBase.get_tools()* vérifie l'unicité des identifiants locaux et les méthodes des outils, et rejette les outils déclarés par des plugins sans la capacité `APPLICATION`. *datalab.plugins.PluginBase.assess_tool()* renvoie la raison pour laquelle un outil ne peut pas être ouvert pour la sélection courante, ou `None`.

Instruments

Un outil peut désigner une méthode `instrument` au lieu d'un `launcher`. Cette méthode renvoie un *datalab.plugins.instruments.PluginInstrument*, et DataLab ouvre une fenêtre pour lui : une vue en direct à gauche, les réglages de l'instrument à droite, un bouton En direct qui rafraîchit la vue en continu, une option Nouveau groupe et un bouton Acquérir. Le plugin n'écrit aucun code d'interface utilisateur :

```
import guidata.dataset as gds
from datalab.plugins.instruments import (
  InstrumentAcquisition,
  InstrumentFrame,
  PluginInstrument,
)
```

(suite sur la page suivante)

(suite de la page précédente)

```
class SourceSettings(gds.DataSet):
  amplitude = gds.FloatItem("Amplitude", default=1.0, unit="V")
  count = gds.IntItem("Signals per acquisition", default=10, min=1)

class Source(PluginInstrument):
  live_interval_ms = 200

  def __init__(self):
    super().__init__(SourceSettings())

  def preview(self):
    signal = self.make_signal()
    return InstrumentFrame([signal], summary="...", value_range=(-2.0, 2.0))

  def acquire(self):
    signals = [self.make_signal() for _ in range(self.settings.count)]
    return InstrumentAcquisition("Source - acquisition 001", signals)

class MyPlugin(PluginBase):
  TOOLS = (
    PluginTool(
      id="source",
      title="Signal source...",
      instrument="signal_source",
      object_type=RecipeObjectType.SIGNAL,
    ),
  )

  def signal_source(self):
    return Source()
```

DataLab modifie `settings` sur place, en respectant l'état actif, les groupes et les onglets du DataSet guidata, puis appelle *`preview()`* après chaque modification et, en mode direct, toutes les `live_interval_ms` millisecondes. Un *`InstrumentFrame`* contient des signaux tracés ensemble ou une seule image, un court résumé affiché sous la vue, et une plage fixe facultative en Y (signaux) ou de couleurs (image). *`acquire()`* renvoie un *`InstrumentAcquisition`* : DataLab ajoute ses objets dans un nouveau groupe du panneau correspondant, portant le titre de l'acquisition, et les sélectionne, afin qu'une recette puisse s'exécuter aussitôt sur eux. Les deux méthodes lèvent `ValueError` avec un message destiné à l'utilisateur lorsque les réglages ne peuvent pas être utilisés ; DataLab affiche ce message dans la fenêtre.

Lorsque l'utilisateur désactive l'option Nouveau groupe de la fenêtre, DataLab ajoute plutôt les objets acquis au groupe courant du panneau : comme ils sont sélectionnés, les acquisitions suivantes vont dans le même groupe, jusqu'à ce que l'utilisateur sélectionne un autre groupe. L'option et le groupe courant sont pris en compte au clic sur Acquérir : les modifier pendant l'acquisition ne change pas sa destination, et un nouveau groupe est créé si le groupe a été supprimé entre-temps. Un nouveau groupe est tout de même créé lorsque le panneau n'a pas de groupe courant, par exemple lorsqu'il est vide. L'attribut de classe *`new_group`* définit l'état par défaut de l'option (`True` par défaut) : un instrument dont les acquisitions sont généralement des objets uniques, comme un attribut d'appareil lu à la demande, peut le mettre à `False`. Une fois fait, le choix de l'utilisateur est mémorisé pour chaque instrument et prévaut sur cette valeur par défaut.

DataLab crée l'instrument une seule fois par outil et le conserve jusqu'au rechargement des plugins : les réglages sont ainsi conservés d'une ouverture à l'autre.

DataLab Desktop appelle l'instrument en dehors du fil d'exécution de l'interface utilisateur, un appel à la fois, même lorsque sa fenêtre est fermée puis rouverte pendant un appel, et ne modifie ses réglages qu'entre deux appels : un instrument lent,

comme un appareil attendant ses données, ne fige pas DataLab. Les méthodes facultatives *open()* et *close()* sont appelées à l'ouverture et à la fermeture de la fenêtre, y compris lors du rechargement des plugins ou de la fermeture de DataLab, par exemple pour se connecter à un appareil puis le libérer. Si `open` lève une erreur, son message reste affiché dans la fenêtre et l'instrument n'est pas appelé : le rafraîchissement ou l'acquisition suivant, par exemple après une modification des réglages, appelle d'abord `open` à nouveau. `close` est appelée même si `open` a échoué, et ne doit libérer que ce qui est encore détenu. Les données reçues par l'instrument dans ses propres fils d'exécution, comme les événements d'un appareil, sont conservées par l'instrument jusqu'à son prochain appel de `preview` ou `acquire`.

Les connecteurs d'instruments doivent attacher une entrée de métadonnées `experimental_source` aux objets acquis avant de les renvoyer ; DataLab ne la remplit pas automatiquement. Utilisez les fonctions `make_experimental_source` et `set_experimental_source` de Sigima pour enregistrer, selon le cas, l'adresse de l'appareil, l'attribut, la sélection et les horodatages. Cela identifie l'origine de la mesure, pas une acquisition reproductible. Le guide des données expérimentales de Sigima fournit un exemple et décrit le format portable des métadonnées.

DataLab-Web prend aussi en charge les outils : il les liste aux mêmes endroits, ouvre les instruments dans la même fenêtre et appelle les lanceurs dans l'hôte Pyodide du plugin.

Tuiles de la page d'accueil

Chaque plugin actif déclarant la capacité `APPLICATION` ajoute une tuile à la section Applications, en haut de la Page d'accueil. Par défaut, cette tuile est construite à partir du nom, de la description et de l'icône du `PluginInfo` du plugin. Un clic sur cette tuile affiche la page du plugin dans le Panneau Applications.

L'icône du plugin est déclarée avec `PluginInfo.icon`, soit sous la forme d'une ressource `package:relative/path` (image SVG ou bitmap fournie dans la wheel du plugin), soit sous la forme d'une image `module:filename` enregistrée par un module auprès de guidata (par exemple `datalab:delete.svg`), soit sous la forme du nom de fichier d'une icône de DataLab. Elle est affichée sur la tuile par défaut, dans le panneau Applications et dans la boîte de dialogue de configuration des plugins ; les plugins sans icône utilisent une icône de plugin générique. Pour proposer d'autres points d'entrée, par exemple pour ouvrir directement un exemple, déclarez le tuple de classe `WELCOME_TILES` de valeurs *datalab.plugins.tiles.WelcomeTile* :

```
from datalab.plugins.tiles import WelcomeTile

class MyPlugin(PluginBase):
  PLUGIN_INFO = PluginInfo(
    id="org.example.my-plugin",
    name="My Plugin",
    icon="datalab_my_plugin:icons/my_plugin.svg",
    capabilities=(PluginCapability.APPLICATION,),
  )
  WELCOME_TILES = (
    WelcomeTile(id="application", title="My Plugin"),
    WelcomeTile(
      id="quickstart",
      title="Open quick start",
      description="Open the packaged quick start workspace",
      icon="datalab_my_plugin:icons/quickstart.svg",
      launcher="open_quickstart",
    ),
  )

  def open_quickstart(self):
    self.launch_example("quickstart")

  def create_actions(self):
```

(suite sur la page suivante)

(suite de la page précédente)

```
    pass
```

Les tuiles déclarées remplacent la tuile par défaut. La première est la tuile principale du plugin ; les autres sont affichées à côté d'elle lorsque la section Applications a la place nécessaire, et deviennent sinon des actions de son menu, ouvert depuis son bouton « … » ou par un clic droit sur la tuile. Une tuile sans `launcher` affiche la page du plugin dans le panneau Applications ; sinon, la méthode du plugin indiquée est appelée sans argument. Une tuile sans `icon` utilise l'icône du plugin. *`datalab.plugins.PluginBase.get_welcome_tiles()`* valide l'unicité des ID locaux et les méthodes de lancement, et rejette les tuiles déclarées par des plugins sans la capacité `APPLICATION`. La page d'accueil actualise ses tuiles lorsque les plugins sont chargés, rechargés ou désactivés, et signale les erreurs levées par une méthode de lancement sans fermer DataLab.

L'utilisateur choisit les applications affichées et leur ordre : la page d'accueil affiche un nombre limité de lignes de tuiles, avec d'abord les applications épinglées, puis celles utilisées récemment, puis les autres par ordre alphabétique (voir Page d'accueil). Un plugin ne peut donc pas compter sur la visibilité de sa tuile : ses fonctionnalités doivent rester accessibles depuis le panneau Applications.

Créer un plugin avec `datalab-plugin create`

La commande `datalab-plugin` est installée avec DataLab sous forme de paquet Python ; la version autonome de DataLab ne la fournit pas. Exécutez-la sans option pour créer un plugin en fichier unique :

```
datalab-plugin create
```

La commande demande seulement le nom du plugin et sa description, puis écrit un fichier `datalab_<nom>.py` dans le répertoire de plugins de l'utilisateur (voir Où est positionné un plugin ?). Sélectionnez Plugins > Recharger les plugins dans DataLab, ou redémarrez DataLab : le plugin ajoute son sous-menu au menu Plugins, avec un traitement d'exemple des signaux sélectionnés. Pour réaliser votre propre traitement, modifiez la fonction `sample_processing` du fichier, qui reçoit les valeurs Y d'un signal sous forme de tableau NumPy et en renvoie de nouvelles, puis rechargez les plugins.

Des options adaptent le fichier généré :

- `--object-kind image` : le traitement d'exemple s'applique aux images, et sa fonction reçoit les données de l'image.
- `--capability` : sans la capacité `processing`, le sous-menu contient une action affichant un message, à remplacer par la vôtre.
- Un dossier de destination, donné en argument : il doit alors être ajouté aux répertoires de plugins (voir Onglet Avancé).
- `--package` et `--plugin-id` : à défaut, le nom du module et l'ID du plugin sont déduits du nom du plugin. Le nom du module doit commencer par `datalab_`.

Un fichier existant n'est jamais écrasé. Pour partager le plugin, envoyez son fichier : les autres utilisateurs l'installent avec Plugins > Configurer les plugins… > Depuis un fichier (voir Onglet Depuis un fichier).

Le plugin brut en fichier unique et le wheel peuvent tous deux être publiés dans un catalogue Desktop (voir Modules en fichier unique dans un catalogue). Pour développer un plugin plus important avec le packaging et les tests déjà configurés, créez plutôt un projet structuré en couches avec `--project` (voir Créer un projet de plugin structuré en couches).

Créer un projet de plugin structuré en couches

Avec `--project`, la commande installée `datalab-plugin` crée un petit projet installable qui respecte les contrats stables des plugins et de propriété des fonctionnalités. Exécutez-la sans autre option pour une configuration interactive :

```
datalab-plugin create --project
```

La commande demande le nom d'affichage et le compte GitHub (utilisateur ou organisation) qui hébergera le projet, et propose des valeurs par défaut pour le package Python, l'ID de plugin en notation de domaine inversé, la description et la destination. L'ID de plugin par défaut est `io.github.<compte>.<nom>` ; les ID `org.datalab.` sont réservés à

l'organisation DataLab-Platform, et `org.example.` est utilisé lorsqu'aucun compte n'est indiqué. Pour les scripts ou une configuration reproductible, indiquez explicitement les valeurs :

```
datalab-plugin create --project datalab-camera-characterization \
  --name "Camera Characterization" \
  --github-account DataLab-Platform \
  --package datalab_camera_characterization \
  --plugin-id org.datalab.camera-characterization \
  --description "Characterize scientific cameras" \
  --capability application \
  --capability processing \
  --object-kind image
```

Le projet généré avec une organisation `src` applique l'architecture validée par le pilote Camera :

```
package root (host-independent identity)
├── core
├── workflow
└── adapters
    ├── desktop.py
    └── web.py
```

`core` contient le comportement métier indépendant de l'hôte, `workflow` peut utiliser les contrats de recettes sans interface de DataLab et `adapters` gère l'intégration aux hôtes. Le point d'entrée `datalab.plugins` cible l'adaptateur Desktop ; la prise en charge Web démarre explicitement avec le statut `unsupported`. Un test AST généré empêche `core` et `workflow` d'importer des modules d'interface graphique, de navigateur ou d'adaptateur.

Le projet comprend également un *`datalab.plugins.PluginInfo`* stable, un traitement d'exemple possédé lorsque la capacité `processing` est sélectionnée, une configuration Ruff, une documentation de l'architecture et des contributions, un changelog, un README et une licence BSD-3-Clause. La commande refuse d'écraser une destination existante.

Le modèle reste volontairement réduit. Ajoutez des recettes métier, des paramètres DataSet, des exemples empaquetés et une intégration continue spécialisée uniquement lorsque le plugin en a besoin.

Pour essayer le plugin, ajoutez le répertoire généré comme répertoire de plugins (voir Onglet Avancé) et rechargez les plugins : DataLab charge la classe du plugin depuis `adapters/desktop.py` sans installer le projet. Pour valider le projet, installez-le avec ses dépendances de test et lancez ses vérifications depuis le répertoire généré :

```
python -m pip install -e ".[test]"
python -m pytest
python -m ruff check .
```

Le `pyproject.toml` généré exige Python 3.10 ou ultérieur et la version majeure courante de DataLab (par exemple `datalab-platform >= 1.4, < 2`), et déclare les URL du projet lorsque le compte GitHub est connu. Le fichier `.github/workflows/release.yml` généré publie le plugin sans compte PyPI : pousser une étiquette comme `v0.1.0` construit le wheel, vérifie que l'étiquette correspond à la version du package, joint le wheel à une release GitHub et affiche la fiche à soumettre au catalogue de plugins DataLab, avec l'empreinte SHA-256 du wheel. Les utilisateurs installent ensuite le wheel avec Plugins > Configurer les plugins… > Depuis un fichier (voir Onglet Depuis un fichier). L'inscription du plugin au catalogue est facultative ; une fois inscrit, il peut aussi être installé depuis l'onglet Catalogue (voir Onglet Catalogue).

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/builtin 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

Cet exemple enregistre une fonctionnalité de traitement dotée d'une identité stable et d'un propriétaire, puis crée une action qui l'exécute via le registre du processeur :

```
# -*- 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
```

(suite sur la page suivante)

(suite de la page précédente)

```
    <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"""

    FEATURE_ID = "org.datalab.examples.custom-filters:weighted-average-denoise"
    PLUGIN_INFO = datalab.plugins.PluginInfo(
        id="org.datalab.examples.custom-filters",
        name="My custom filters",
        version="1.0.0",
        description="This is an example plugin",
        capabilities=(datalab.plugins.PluginCapability.PROCESSING,),
    )

    def register_computations(self) -> None:
        """Register computations owned by this plugin."""
        wrapped_func = sipi.Wrap1to1Func(weighted_average_denoise)
        self.imagepanel.processor.register_1_to_1(
            wrapped_func,
            "Weighted average denoise",
            feature_id=self.FEATURE_ID,
            owner_plugin_id=self.plugin_id,
        )

    def create_actions(self) -> None:
        """Create actions"""
        acth = self.imagepanel.acthandler
```

(suite sur la page suivante)

(suite de la page précédente)

```
        with acth.new_menu(self.PLUGIN_INFO.name):
            acth.action_for(self.FEATURE_ID)
```

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())
```

Exemples de modèles utilisés par la suite de tests

DataLab fournit aussi des modèles de plugins utilisés par les tests d'intégration dans `datalab/tests/features/plugins/templates`. Ils servent de références de développement pour :

- une structure de plugin valide de base,
- des menus de plugin imbriqués,
- des plugins avec actions de boîtes de dialogue,
- des plugins avec de nombreuses actions,
- des plugins avec de longues descriptions.

Les tests fonctionnels correspondants sont dans `datalab/tests/features/plugins/plugins_app_test.py` et couvrent le cycle de vie des plugins, le rechargement à chaud, la gestion des erreurs, les noms en doublon et le filtrage de configuration.

Autres exemples

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

Plugins et DataLab-Web

Les plugins sont en grande partie portables entre l'application de bureau et DataLab-Web, l'édition native pour navigateur de la plateforme. La même sous-classe *`datalab.plugins.PluginBase`* peut fonctionner dans les deux, car l'API de plugin est partagée. Quelques contraintes s'appliquent toutefois à l'environnement d'exécution du navigateur :

- Les boîtes de dialogue de paramètres doivent être ouvertes de manière asynchrone. Dans le navigateur, l'appel synchrone `param.edit(self.main)` ne peut pas bloquer la boucle d'événements ; utilisez plutôt `await param.edit_async(self.main)`. Un plugin écrit de cette façon fonctionne sans modification sur le bureau.

— Pas d'interfaces graphiques propres à Qt. Les plugins qui intègrent des widgets Qt personnalisés ou s'appuient sur les outils interactifs de PlotPy sont réservés au bureau ; leurs parties graphiques n'ont pas d'équivalent dans le navigateur.
— L'exécution a lieu dans le navigateur (WebAssembly), sans accès natif au système de fichiers au-delà du système de fichiers en mémoire.

Par conséquent, un plugin qui repose sur une interface graphique personnalisée peut ne pas être entièrement compatible avec DataLab-Web. Le catalogue de plugins DataLab indique lesquels de ses plugins prennent aussi en charge DataLab-Web. Reportez-vous à la documentation de DataLab-Web pour le guide pratique de chargement des plugins dans le navigateur.

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.**ClassicsImageFormat**

Objet représentant les types de fichiers image classiques

static read_data(filename: str) → ndarray

Lire les données et les renvoyer

Paramètres
filename – Nom du fichier

Renvoie
Données du tableau de l'image

static write_data(filename: str, data: ndarray) → None

Écrire les données dans un fichier

Paramètres
— **filename** – Nom du fichier
— **data** – Données du tableau de l'image

write_with_options(filename: str, obj: ImageObj, writer_options: dict[str, object]) → None

Écrire des données d'image classiques avec des options propres au format validées.

class datalab.plugins.**FailedPluginInfo**(name: str, filepath: str, traceback: str, source: str = '')

Informations sur un plugin qui n'a pas pu être chargé ou instancié.

class datalab.plugins.**FormatInfo**(name: str = None, extensions: str = None, readable: bool = False, writeable: bool = False, requires: list[str] = None)

Informations sur le format

to_rst_table_row() → str

Renvoyer une ligne de tableau reStructuredText décrivant le format (tableau `.. list-table::` à 5 colonnes)

class datalab.plugins.**ImageFormatBase**

Objet de base pour les formats d'image.

Cette classe définit l'interface des formats d'entrée/sortie des images. C'est une classe de base abstraite qui définit les méthodes devant être implémentées par toute classe de format d'image.

validate_read_param(param: DataSet | None) → None

Refuser les paramètres de lecture pour les formats qui ne les prennent pas en charge.

abstractmethod read(filename: str, worker: CallbackWorkerProtocol | None = None, * *(Séparateur de paramètres par mot-clé uniquement (PEP 3102))*, param: DataSet | None = None) → Sequence[ImageObj]

Lire une liste d'objets image dans un fichier

Paramètres
- **filename** – Nom du fichier
- **worker** – Objet de traitement des rappels
- **param** – Paramètres de lecture facultatifs propres au format

Renvoie
Liste d'objets image

abstractmethod write(filename: str, obj: ImageObj) → None

Écrire les données dans un fichier

Paramètres
- **filename** – nom du fichier
- **obj** – objet natif (signal ou image)

Lève
NotImplementedError – si le format n'est pas pris en charge

write_with_options(filename: str, obj: ImageObj, writer_options: dict[str, object]) → None

Écrire une image avec des options propres au format.

Paramètres
- **filename** – Nom du fichier
- **obj** – Objet image
- **writer_options** – Options d'écriture propres au format

Lève
ValueError – Si ce format ne prend pas en charge les options d'écriture

class datalab.plugins.**PluginBase**

Classe de base du plugin

last_example_data: *PluginExampleData* | None = None

Données du dernier exemple généré ouvert (None pour les exemples empaquetés)

classmethod get_plugin_id() → str

Renvoie l'identifiant stable du plugin ou un identifiant de repli hérité déterministe.

property plugin_id: str

Renvoie l'identifiant stable du plugin ou un identifiant de repli hérité déterministe.

classmethod get_recipes() → tuple[RecipeDescriptor, ...]

Renvoie les descripteurs de recettes validés exposés par ce plugin.

classmethod get_recipe(recipe_id: str) → RecipeDescriptor

Renvoie un descripteur de recette à partir de son identifiant qualifié.

classmethod get_recipe_launchers() → Mapping[str, str]

Renvoie les associations validées entre recettes et méthodes remplaçant l'interface Desktop.

launch_recipe(recipe_id: str) → RecipeOutcome | None

Lance une recette sur la sélection courante.

Une méthode du plugin déclarée dans `RECIPE_LAUNCHERS` remplace l'interaction générique de *start_recipe()*.

get_selected_objects() → list[SignalObj | ImageObj]

Renvoie les objets sélectionnés dans le panneau Signal ou Image courant.

example_parameter_values(recipe_id: str, objects: Sequence[SignalObj | ImageObj]) → Mapping[str, object]

Renvoie les valeurs de paramètres du dernier exemple pour une recette et des objets.

Les valeurs ne sont renvoyées que si tous les objets proviennent du dernier exemple généré ouvert : elles ne conviennent qu'à ces données.

assess_recipe(recipe_id: str, objects: Sequence[SignalObj | ImageObj] | None = None) → RecipeReadiness

Évalue si une recette peut s'exécuter sur des objets (par défaut : la sélection).

start_recipe(recipe_id: str, objects: Sequence[SignalObj | ImageObj] | None = None, parameter_values: Mapping[str, object] | None = None) → RecipeOutcome | None

Exécute une recette via l'interaction générique de DataLab.

DataLab affecte les objets (par défaut : la sélection) aux entrées de la recette, en interrogeant l'utilisateur si nécessaire, les vérifie, édite les paramètres, puis exécute la recette.

Paramètres
- **recipe_id** – identifiant qualifié de la recette
- **objects** – objets candidats (par défaut : sélection courante)
- **parameter_values** – valeurs initiales des paramètres (par défaut : celles du dernier exemple ouvert, lorsque les objets en proviennent)

Renvoie
Résultat de la recette, ou None en cas d'annulation ou d'échec

try_example(example_id: str, recipe_id: str) → RecipeOutcome | None

Ouvre un exemple, puis lance l'une des recettes pour lesquelles il est conçu.

classmethod get_tools() → tuple[PluginTool, ...]

Renvoie les outils validés listés dans le panneau Applications.

classmethod get_tool(tool_id: str) → PluginTool

Renvoie un outil à partir de son identifiant local au plugin.

assess_tool(tool_id: str, objects: Sequence[SignalObj | ImageObj] | None = None) → str | None

Renvoie la raison pour laquelle un outil ne peut pas être ouvert sur des objets (par défaut : la sélection).

Renvoie
Message indiquant quels objets sélectionner, ou None si l'outil peut être ouvert

launch_tool(tool_id: str) → object

Ouvre un outil du plugin : appelle sa méthode de lancement, ou affiche son instrument.

Lève
ValueError – si la sélection ne permet pas d'ouvrir l'outil

get_instrument(tool_id: str) → PluginInstrument

Renvoie l'instrument d'un outil, créé une seule fois par enregistrement.

classmethod get_examples() → tuple[PluginExample, ...]

Renvoie les exemples empaquetés validés exposés par ce plugin.

classmethod get_example(example_id: str) → PluginExample

Renvoie un exemple empaqueté à partir de son identifiant local au plugin.

classmethod materialize_example(example_id: str) → PluginExampleData | None

Génère un exemple en mémoire, ou s'en remet à sa ressource de paquet.

launch_example(example_id: str) → PluginExample | None

Demande confirmation, ouvre un exemple généré ou empaqueté, puis le sélectionne.

open_example(example_id: str, reset_all: bool = True) → PluginExample

Ouvre un exemple HDF5 généré ou empaqueté dans l'espace de travail Desktop.

Les exemples générés (lorsque `materialize_example()` renvoie des données) sont chargés directement dans les panneaux Signal et Image ; les exemples empaquetés sont ouverts via leur ressource HDF5 native. Les données matérialisées sont conservées dans *last_example_data*, afin que les recettes lancées sur ces objets réutilisent les valeurs de paramètres de l'exemple (voir *example_parameter_values()*).

classmethod get_welcome_tiles() → tuple[WelcomeTile, ...]

Renvoie les tuiles validées de la page d'accueil exposées par ce plugin d'application.

Sans `WELCOME_TILES` déclarées, un plugin d'application expose une tuile par défaut construite à partir de son *PluginInfo*. Les tuiles déclarées sans icône héritent de l'icône du plugin.

launch_welcome_tile(tile_id: str) → object

Lance une tuile de la page d'accueil.

Une tuile sans méthode de lancement affiche la page du plugin dans le panneau Applications ; sinon, sa méthode de plugin est appelée sans argument.

Renvoie
Valeur renvoyée par la méthode de lancement (None pour le panneau Applications)

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

Les déclarations du plugin (recettes, exemples, outils, tuiles de la page d'accueil) sont d'abord validées : un plugin invalide n'est pas enregistré, de sorte que le panneau Applications, la page d'accueil et les menus ne reçoivent que des déclarations valides.

Lève
- **TypeError** – si une déclaration n'est pas du bon type
- **ValueError** – si une déclaration est invalide ou si l'ID du plugin est déjà utilisé

unregister()

Désenregistrer le plugin

remove_owned_features() → None

Supprime les fonctionnalités de calcul de ce plugin des processeurs disponibles.

register_hooks()

Enregistre les hooks du plugin.

Appelée par *register()* au démarrage de l'application ou lors du rechargement des plugins, avant que la catégorie d'actions PLUGINS des panneaux n'existe : à utiliser uniquement pour des effets de bord sans interface graphique (enregistrement de formats d'E/S, écouteurs). `self.main` et `self.proxy` sont déjà définis. Une exception levée ici annule tout l'enregistrement (le plugin reste non enregistré).

unregister_hooks()

Désenregistre les hooks du plugin.

Appelée par *unregister()* ; doit défaire *register_hooks()*. Les fonctionnalités de calcul détenues par le plugin sont ensuite supprimées automatiquement.

register_computations() → None

Enregistrer les traitements possédés après la création des panneaux signal et image.

Appelée une fois par plugin enregistré, juste avant *create_actions()* ; une exception levée ici supprime les fonctionnalités détenues par le plugin mais le laisse enregistré.

abstractmethod create_actions()

Crée les actions.

Appelée juste après *register_computations()*, dans la catégorie d'actions PLUGINS des panneaux : les actions créées ici alimentent le menu Plugins.

class datalab.plugins.**PluginBaseMeta**(name, bases, namespace, / *(Séparateur de paramètres positionnels uniquement (PEP 570))*, **kwargs)

Métaclasse mixte pour éviter les conflits

class datalab.plugins.**PluginCapability**(value)

Capacité exposée par un plugin aux consommateurs de DataLab.

class datalab.plugins.**PluginInfo**(name: str = None, version: str = '0.0.0', description: str = '', icon: str = None, id: str | None = None, capabilities: Collection[PluginCapability] = <factory>, documentation_url: str | None = None)

Informations sur le plugin

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

Renvoie un plugin à partir de son identifiant stable, de son ancien nom ou de sa classe.

Les anciens noms d'affichage ne sont acceptés que s'ils désignent un seul 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 clear_plugin_classes() → None

Effacer les classes de plugins enregistrées

Ceci est principalement utile lors du rechargement des modules de plugins à l'exécution.

classmethod add_discovery_error(tb_text: str) → None

Enregistrer une trace d'erreur survenue lors de la découverte des plugins.

Paramètres
tb_text – Chaîne de trace d'erreur formatée

classmethod get_discovery_errors() → list[str]

Retourner les traces d'erreur collectées lors de la découverte des plugins.

Renvoie
Liste des chaînes de trace d'erreur formatées (peut être vide)

classmethod clear_discovery_errors() → None

Effacer les erreurs de découverte enregistrées.

classmethod add_failed_plugin(name: str, filepath: str, tb_text: str, source: str = '') → None

Enregistrer un plugin qui n'a pas pu être chargé ou instancié.

Paramètres
- **name** – Nom du module ou de la classe du plugin
- **filepath** – Chemin d'accès du module du plugin
- **tb_text** – Chaîne de trace d'erreur formatée
- **source** – Description de la source de découverte

classmethod get_failed_plugins() → list[FailedPluginInfo]

Renvoie des informations structurées sur les plugins qui n'ont pas pu être chargés.

Renvoie
Liste des objets FailedPluginInfo (peut être vide)

classmethod clear_failed_plugins() → None

Effacer les informations sur les plugins ayant échoué.

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.**SignalFormatBase**

Classe représentant un type de fichier de signal

static create_object(filename: str, index: int | None = None) → SignalObj

Créer un objet vide

Paramètres
- **filename** – Nom du fichier
- **index** – Indice de l'objet dans le fichier

Renvoie
Objet signal

create_signal(xydata: ndarray, filename: str, index: int | None = None) → SignalObj

Créer un objet signal à partir de xydata et du nom de fichier.

Paramètres
- **xydata** – Données XY
- **filename** – Nom du fichier
- **index** – Indice de l'objet dans le fichier

Renvoie
Objet signal

create_signals_from(xydata: ndarray, filename: str) → list[SignalObj]

Créer des objets signal à partir de xydata et du nom de fichier

Paramètres
- **xydata** – Données XY
- **filename** – Nom du fichier

Renvoie
Liste d'objets signal

read(filename: str, worker: CallbackWorkerProtocol | None = None) → Sequence[SignalObj]

Lire une liste d'objets signal dans un fichier

Paramètres
- **filename** – Nom du fichier
- **worker** – Objet de traitement des rappels

Renvoie
Liste d'objets signal

read_xydata(filename: str) → ndarray

Lire les données et métadonnées d'un fichier, écrire les métadonnées dans l'objet et renvoyer xydata

Paramètres
filename – Nom du fichier

Renvoie
Données XY

datalab.plugins.**discover_plugins**() → list[ModuleType]

Découvrir les plugins via les points d'entrée de packages et la convention de nommage.

Les packages installés peuvent exposer une sous-classe de *PluginBase* via le groupe de points d'entrée `datalab.plugins`. Cette fonction recharge ou importe également les modules et packages qui respectent la convention de nommage historique `"{MOD_NAME}_*"`, ainsi que le module `adapters.desktop` de ces packages, qui définit la classe du plugin des packages organisés comme ceux créés par `datalab-plugin create`.

Les erreurs d'importation pour les plugins individuels sont capturées et enregistrées afin qu'un plugin défectueux n'empêche pas les autres de se charger. Les traces d'erreurs sont accumulées dans les attributs de classe *PluginRegistry* afin que les appelants (par exemple, la fenêtre principale) puissent les rejouer dans la console interne une fois qu'elle est prête.

Cette fonction modifie `sys.modules` et le registre des plugins sans synchronisation : ne l'appelez que depuis le thread principal Qt.

Renvoie
Modules importés ou rechargés contenant les plugins découverts

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.**format_tool_requirement**(tool: PluginTool) → str

Renvoie le message indiquant quels objets sélectionner pour ouvrir un outil.

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

Instancier et obtenir les plugins disponibles

Renvoie
Liste des plugins disponibles (en tant qu'instances)

datalab.plugins.**migrate_enabled_plugin_ids**(enabled_plugins: list[str] | None, plugin_classes: list[type[PluginBase]] | None = None) → list[str] | None

Convertit les noms d'affichage des plugins activés en identifiants stables.

Les entrées inconnues sont conservées afin que les plugins temporairement indisponibles ne perdent pas leur état d'activation. Un ancien nom partagé par plusieurs plugins active tous les identifiants correspondants, car l'ancien paramètre ne permet pas de les distinguer.

Paramètres
— **enabled_plugins** – Identifiants stables ou anciens noms d'affichage configurés
— **plugin_classes** – Classes disponibles pour la conversion (registre par défaut)

Renvoie
Identifiants stables activés et entrées inconnues conservées, ou None pour tous les plugins

datalab.plugins.**reload_plugin_modules**() → None

Recharger les modules de plugins et réinitialiser les classes de plugins.

Cet assistant est destiné au rechargement à chaud des plugins à l'exécution. Il :
— Met à jour le chemin de recherche des plugins

— Efface le registre des classes de plugins
— Recharge ou importe tous les modules correspondant à la convention de nommage des plugins
Comme *discover_plugins()*, cette fonction n'est pas thread-safe : ne l'appelez que depuis le thread principal Qt.

Contrats sans interface graphique pour les recettes scientifiques fournies par les plugins.

exception datalab.plugins.recipes.**RecipeCancellationError**

Levée lorsque l'exécution d'une recette constate une demande d'annulation.

class datalab.plugins.recipes.**RecipeCardinality**(value)

Nombre d'objets acceptés par un emplacement d'entrée de recette.

class datalab.plugins.recipes.**RecipeDescriptor**(recipe_id: str, plugin_version: str, title: str, version: str, run: Callable[[Mapping[str, tuple[SignalObj | ImageObj, ...]], DataSet | None, RecipeExecutionContext], RecipeOutcome], description: str = '', inputs: Sequence[RecipeInputSlot] = (), parameter_class: type[DataSet] | None = None, suggest_bindings: Callable[[tuple[SignalObj | ImageObj, ...]], Mapping[str, Sequence[SignalObj | ImageObj]]] | None = None, check_inputs: Callable[[Mapping[str, tuple[SignalObj | ImageObj, ...]], DataSet | None], Sequence[RecipeDiagnostic]] | None = None)

Déclaration typée et versionnée d'une recette sans interface graphique fournie par un plugin.

L'absence de `parameter_class` déclare une recette sans paramètre dont la fonction d'exécution reçoit `None` comme second argument. Dans le cas contraire, cet argument doit être une instance de la sous-classe `DataSet` déclarée.

`suggest_bindings` reçoit les objets candidats et renvoie une proposition d'affectation de certains d'entre eux aux emplacements d'entrée (par exemple à partir des métadonnées). Les hôtes l'utilisent pour affecter les entrées sans interroger l'utilisateur.

`check_inputs` reçoit des entrées complètes et les paramètres (ceux par défaut lors de l'évaluation d'une sélection) et renvoie des diagnostics : les erreurs bloquent l'exécution, les avertissements sont affichés. Il doit rester rapide (métadonnées, dimensions, effectifs) car les hôtes l'appellent à chaque changement de sélection.

property plugin_id: str

Renvoie l'identifiant du plugin propriétaire encodé dans l'identifiant de la recette.

property local_id: str

Renvoie l'identifiant local de la recette encodé dans l'identifiant de la recette.

get_input(slot_id: str) → RecipeInputSlot

Renvoie un emplacement d'entrée déclaré à partir de son identifiant.

class datalab.plugins.recipes.**RecipeDiagnostic**(level: RecipeDiagnosticLevel, code: str, message: str, details: Mapping[str, object]=<factory>)

Diagnostic structuré et identifiable par une machine, émis par une recette.

class datalab.plugins.recipes.**RecipeDiagnosticLevel**(value)

Gravité d'un diagnostic structuré de recette.

class datalab.plugins.recipes.**RecipeExecutionContext**(progress_callback: Callable[[float, str | None], None] | None = None, cancellation_callback: Callable[[], bool] | None = None)

Fonctions de rappel de progression et d'annulation indépendantes de la technologie, pour une recette.

property is_cancelled: bool

Indique si une annulation a été demandée.

raise_if_cancelled() → None

Lève *RecipeCancellationError* lorsqu'une annulation est demandée.

report_progress(progress: float, message: str | None = None) → None

Signale une progression normalisée entre zéro et un.

class datalab.plugins.recipes.**RecipeInputSlot**(id: str, object_type: RecipeObjectType, cardinality: RecipeCardinality, required: bool = True, title: str = '', description: str = '', min_count: int = 1, metadata: Sequence[RecipeMetadataRequirement] = ())

Emplacement d'entrée typé déclaré par une recette.

Paramètres
- **id** – identifiant de l'emplacement, local à la recette
- **object_type** – type des objets acceptés
- **cardinality** – un objet, ou plusieurs objets
- **required** – True si l'emplacement ne peut pas rester vide
- **title** – nom lisible de l'emplacement (dérivé de `id` s'il est vide)
- **description** – ce que les objets doivent représenter et comment ils sont acquis
- **min_count** – nombre minimal d'objets d'un emplacement `MANY` non vide
- **metadata** – entrées de métadonnées attendues sur chaque objet affecté

property display_title: str

Renvoie le titre déclaré, ou un titre dérivé de l'identifiant de l'emplacement.

class datalab.plugins.recipes.**RecipeMetadataRequirement**(key: str, description: str = '', required: bool = True)

Entrée de métadonnées attendue sur les objets affectés à un emplacement d'entrée de recette.

Paramètres
- **key** – clé de métadonnée
- **description** – signification de la valeur (unité, valeurs autorisées…)
- **required** – True si les objets dépourvus de la clé ne peuvent pas être analysés ; False documente une indication facultative, utilisée par exemple pour affecter les objets aux emplacements

class datalab.plugins.recipes.**RecipeObjectOutput**(id: str, value: SignalObj | ImageObj)

Signal ou image nommé produit par une recette.

class datalab.plugins.recipes.**RecipeObjectType**(value)

Type d'objet scientifique accepté par un emplacement d'entrée de recette.

class datalab.plugins.recipes.**RecipeOutcome**(objects: Sequence[RecipeObjectOutput] = (), results: Sequence[RecipeResultOutput] = (), diagnostics: Sequence[RecipeDiagnostic] = ())

Objets, résultats scalaires ancrés et diagnostics produits par une recette.

class datalab.plugins.recipes.**RecipeResultOutput**(id: str, value: TableResult | GeometryResult, anchor_id: str)

Résultat scalaire nommé rattaché à une sortie d'objet nommée.

class datalab.plugins.recipes.**RecipeRunRecord**(run_id: str, plugin_id: str, plugin_version: str, recipe_id: str, recipe_version: str, parameters_json: str, input_uuids: Mapping[str, Sequence[str]], output_uuids: Mapping[str, str], datalab_version: str, sigima_version: str, status: RecipeRunStatus, started_at: str, finished_at: str, schema_version: int = 1)

Provenance locale versionnée partagée par toutes les sorties d'une exécution de recette.

L'enregistrement est conservé sous forme de métadonnées JSON (voir *to_dict()*) dans l'option de métadonnées RECIPE_RUN_RECORD_OPTION de chaque objet de sortie, afin que les consommateurs externes (DataLab-Web, espaces de travail HDF5, outils) puissent s'appuyer sur le schéma suivant (version 1) :

— schema_version (int) : toujours RECIPE_RUN_RECORD_SCHEMA_VERSION
— run_id (str) : UUID partagé par toutes les sorties d'un même appel de recette
— plugin_id / plugin_version (str) : plugin propriétaire et sa version
— recipe_id / recipe_version (str) : recette (<plugin_id>:<name>) et version de son contrat
— parameters_json (str) : DataSet guidata sérialisé avec dataset_to_json ("null" pour les recettes sans paramètre)
— input_uuids (dict[str, list[str]]) : UUID des objets consommés par emplacement d'entrée — un dictionnaire (et non une liste à plat) afin que les rôles des emplacements soient conservés lors des allers-retours
— output_uuids (dict[str, str]) : UUID de l'objet produit pour chaque identifiant de sortie
— datalab_version / sigima_version (str) : versions de l'environnement
— status (str) : une valeur de RecipeRunStatus
— started_at / finished_at (str) : ISO 8601 avec fuseau horaire

Politique de compatibilité : les lecteurs doivent rejeter les enregistrements dont ils ne connaissent pas la schema_version (comme le fait *from_dict()*) ; tout ajout, suppression ou changement sémantique de champ impose d'incrémenter RECIPE_RUN_RECORD_SCHEMA_VERSION et de fournir un chemin de migration explicite pour les espaces de travail enregistrés.

to_dict() → dict[str, object]

Renvoie une représentation composée uniquement de valeurs compatibles JSON.

classmethod from_dict(data: Mapping[str, Any]) → RecipeRunRecord

Restaure un enregistrement d'exécution de recette à partir des métadonnées enregistrées.

class datalab.plugins.recipes.**RecipeRunStatus**(value)

Statut final enregistré sur les sorties d'une recette réussie.

exception datalab.plugins.recipes.**RecipeValidationError**

Levée lorsque les entrées ou les paramètres d'une recette ne respectent pas son descripteur.

Déclarations portables des espaces de travail d'exemple fournis par les plugins.

class datalab.plugins.examples.**PluginExample**(id: str, title: str, resource: str | None = None, description: str = '', recipe_ids: Sequence[str] = (), expected_checks: Sequence[str] = ())

Exemple empaqueté ou généré, et recettes pour lesquelles il est conçu.

Un exemple sans recette est présenté comme un simple jeu de données.

property package: str

Renvoie le paquet importable contenant la ressource de l'exemple.

property resource_path: str

Renvoie le chemin de style POSIX relatif au paquet de ressources.

resolve() → Traversable

Résout la ressource déclarée sans nécessiter de chemin dans le système de fichiers.

as_file() → Iterator[Path]

Matérialise la ressource pour les API qui nécessitent un chemin dans le système de fichiers.

class datalab.plugins.examples.**PluginExampleData**(objects: ~collections.abc.Sequence[~sigima.objects.signal.object.SignalObj | ~sigima.objects.image.object.ImageObj], parameter_values: ~collections.abc.Mapping[str, ~collections.abc.Mapping[str, object]] = <factory>)

Objets scientifiques et valeurs par défaut par recette générés par un plugin.

Paramètres
- **objects** – signaux et images générés
- **parameter_values** – identifiant de recette vers les valeurs de paramètres adaptées à ces objets

values_for(recipe_id: str) → Mapping[str, object]

Renvoie les valeurs de paramètres adaptées à une recette (vide s'il n'y en a pas).

Déclarations portables des tuiles de la page d'accueil fournies par les plugins.

class datalab.plugins.tiles.**WelcomeTile**(id: str, title: str, description: str = '', icon: str | None = None, launcher: str | None = None)

Tuile de la page d'accueil exposée par un plugin d'application

Paramètres
- **id** – ID de la tuile, local au plugin (lettres minuscules, chiffres, ., _, -)
- **title** – titre de la tuile
- **description** – description courte affichée sous le titre
- **icon** – ressource `package:path` (SVG ou bitmap) ou nom de fichier d'une icône de DataLab ; `None` utilise l'icône du plugin
- **launcher** – nom de la méthode du plugin appelée lors d'un clic sur la tuile ; `None` affiche la page du plugin dans le panneau Applications

property is_package_icon: bool

Renvoie True si l'icône est une ressource `package:path`

Déclarations des outils propres aux plugins listés dans le catalogue Applications.

DataLab Desktop et DataLab-Web partagent ce module.

class datalab.plugins.tools.**PluginTool**(id: str, title: str, launcher: str | None = None, description: str = '', icon: str | None = None, instrument: str | None = None, object_type: RecipeObjectType | None = None, selection: ToolSelection = ToolSelection.NONE)

Outil doté de sa propre interface utilisateur, exposé par un plugin d'application

Les recettes couvrent les analyses sans interface exécutées par DataLab ; un outil couvre toute autre interaction propre au plugin (assistant, éditeur interactif, préparation des données, instrument…). DataLab liste les outils dans le catalogue Applications et dans le sous-menu du plugin du menu Plugins.

Un outil est ouvert soit par une méthode `launcher`, appelée sans argument, soit par une méthode `instrument` renvoyant un *datalab.plugins.instruments.PluginInstrument*, que DataLab affiche dans une fenêtre avec une vue en direct et les réglages de l'instrument.

Paramètres
- **id** – identifiant de l'outil local au plugin (lettres minuscules, chiffres, ., _, -)
- **title** – titre de l'outil
- **launcher** – nom de la méthode du plugin appelée, sans argument, pour ouvrir l'outil
- **description** – courte description de ce que fait l'outil
- **icon** – ressource `package:path` (SVG ou bitmap) ou nom de fichier d'une icône de DataLab ; `None` utilise l'icône du plugin

— **instrument** – nom de la méthode du plugin renvoyant l'instrument affiché par DataLab (exclusif avec `launcher`)
— **object_type** – type des objets sur lesquels l'outil travaille ou qu'il crée ; l'outil est listé dans le menu de ce panneau (`None` : les deux panneaux)
— **selection** – sélection nécessaire pour ouvrir l'outil, en ne comptant que les objets de type `object_type`

class datalab.plugins.tools.**ToolSelection**(value)

Sélection nécessaire à l'ouverture d'un outil (les noms suivent les conditions de DataLab).

datalab.plugins.tools.**tool_accepts_selection**(tool: PluginTool, objects: Sequence[SignalObj | ImageObj]) → bool

Renvoie True si les objets sélectionnés permettent d'ouvrir l'outil.

Paramètres
— **tool** – outil à ouvrir
— **objects** – objets sélectionnés

Renvoie
True si le nombre d'objets sélectionnés du type de l'outil correspond à la sélection de l'outil

Instruments exposés comme outils par les plugins d'application.

Un instrument est un objet sans interface : il porte des réglages (un DataSet guidata), renvoie une image en direct pour ces réglages et acquiert des objets. DataLab l'affiche dans une fenêtre avec la vue en direct à gauche et les réglages à droite. DataLab Desktop et DataLab-Web partagent ce module.

class datalab.plugins.instruments.**InstrumentAcquisition**(group_title: str, objects: Sequence[SignalObj | ImageObj])

Objets acquis par un instrument, ajoutés par DataLab à l'espace de travail.

DataLab les ajoute à un nouveau groupe, ou au groupe courant du panneau lorsque l'utilisateur désactive l'option Nouveau groupe de la fenêtre de l'instrument (voir *PluginInstrument.new_group*).

Paramètres
— **group_title** – titre du groupe créé pour l'acquisition, le cas échéant
— **objects** – signaux acquis, ou images acquises

class datalab.plugins.instruments.**InstrumentFrame**(objects: Sequence[SignalObj | ImageObj], summary: str = '', value_range: tuple[float, float] | None = None)

Image en direct affichée par DataLab pour les réglages courants de l'instrument.

Paramètres
— **objects** – signaux tracés ensemble, ou une seule image
— **summary** – court texte affiché sous la vue (par exemple les niveaux mesurés)
— **value_range** – plage fixe en Y des signaux, ou plage de couleurs de l'image (`None` : automatique)

class datalab.plugins.instruments.**PluginInstrument**(settings: DataSet)

Instrument ouvert par un outil de plugin.

DataLab écrit les valeurs modifiées dans `settings` avant d'appeler *preview()* ou *acquire()*. Les deux méthodes lèvent `ValueError` avec un message destiné à l'utilisateur lorsque les réglages ne peuvent pas être utilisés. Le plugin conserve une instance par outil pendant la session : les réglages sont ainsi conservés d'une ouverture à l'autre.

DataLab appelle *open()* à l'ouverture de la fenêtre de l'instrument et *close()* à sa fermeture (y compris lorsque le plugin est rechargé ou que DataLab se ferme). Il appelle ces méthodes, *preview()* et *acquire()* une à la fois, en dehors du fil d'exécution de l'interface graphique, même lorsque la fenêtre est fermée puis rouverte pendant un appel, et ne modifie `settings` qu'entre deux appels : un instrument lent ne fige pas DataLab. Les données

reçues par l'instrument dans ses propres fils d'exécution (par exemple les événements d'un appareil) doivent être conservées jusqu'au prochain appel de *preview()* ou *acquire()*.

Paramètres
settings – réglages de l'instrument, modifiés par DataLab

live_interval_ms: int = 500

Intervalle entre deux images en mode direct, en millisecondes

new_group: bool = True

État par défaut de l'option Nouveau groupe de la fenêtre de l'instrument (si True, chaque acquisition est ajoutée à un nouveau groupe, sinon au groupe courant du panneau). Une fois fait, le choix de l'utilisateur prévaut.

open() → None

Préparer l'instrument à l'ouverture de sa fenêtre (par exemple s'y connecter).

Lève `ValueError` avec un message destiné à l'utilisateur si l'instrument ne peut pas être utilisé : DataLab affiche ce message, et n'appelle pas *preview()* ou *acquire()* tant qu'une nouvelle demande (par exemple après une modification des réglages) n'a pas ouvert l'instrument avec succès. Ne fait rien par défaut.

close() → None

Libérer l'instrument à la fermeture de sa fenêtre (par exemple s'en déconnecter).

Appelée même si *open()* a échoué, et éventuellement plusieurs fois : elle ne doit libérer que ce qui est encore détenu. Ne fait rien par défaut.

abstractmethod preview() → InstrumentFrame

Renvoie une image en direct pour les réglages courants.

abstractmethod acquire() → InstrumentAcquisition

Acquiert des objets avec les réglages courants.

Exécuteur transactionnel Desktop pour les recettes scientifiques fournies par les plugins.

exception datalab.gui.plugins.recipe_runner.**RecipeCommitError**(message: str, rollback_error: Exception | None = None)

Levée après l'annulation d'une validation (commit) échouée d'une recette dans l'espace de travail.

rollback_error

Exception levée par l'annulation elle-même, ou None si l'espace de travail a été restauré avec succès.

class datalab.gui.plugins.recipe_runner.**RecipeRunner**(mainwindow: DLMainWindow)

Valide, exécute et valide de manière transactionnelle une recette sur Desktop.

run(descriptor: RecipeDescriptor, inputs: Mapping[str, Sequence[SignalObj | ImageObj]], parameters: DataSet | None = None, context: RecipeExecutionContext | None = None, group_title: str | None = None) → RecipeOutcome

Valide, exécute et valide de manière atomique un appel de recette.

3.5.2 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
  - Convention de nommage des fichiers de plugin
  - Nouveaux objets résultat
  - Accès à l'UUID de l'objet
  - 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
```

Convention de nommage des fichiers de plugin

Les noms de fichiers de plugin doivent suivre la nouvelle convention de nommage :

Important

Les fichiers de plugin doivent être renommés de `cdl_*.py` en `datalab_*.py`.

Exemples :

— `cdl_myplugin.py` → `datalab_myplugin.py`
— `cdl_custom_filters.py` → `datalab_custom_filters.py`
— `cdl_example.py` → `datalab_example.py`

Cette convention de nommage est requise pour que DataLab découvre et charge automatiquement les plugins au démarrage. Les plugins qui ne suivent pas ce schéma de nommage ne seront pas chargés.

Note

Le mécanisme de découverte des plugins recherche les fichiers Python correspondant au motif `datalab_*.py` dans le répertoire des plugins. Assurez-vous de mettre à jour les noms de fichiers de vos plugins en conséquence.

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`).

Accès à l'UUID de l'objet

La manière d'accéder à l'UUID d'un objet a changé :

```
# v0.20 - Direct attribute access
obj_uuid = obj.uuid

# v1.0 - Use helper function from datalab.objectmodel
from datalab.objectmodel import get_uuid
obj_uuid = get_uuid(obj)

# Alternative in v1.0 - Access metadata directly
obj_uuid = obj.get_metadata_option("uuid")
```

Ce changement s'applique aux objets `SignalObj` et `ImageObj`. L'UUID est désormais stocké dans les métadonnées de l'objet plutôt que comme un attribut direct.

> **Note**
>
> La fonction `get_uuid()` est définie dans `datalab.objectmodel`, et non comme une méthode de l'objet lui-même. Importez-la avant utilisation : `from datalab.objectmodel import get_uuid`.

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 8 : 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	

suite sur la page suivante

Tableau 8 – suite de la page précédente

DataLab v0.20	DataLab v1.0
`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)`
`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_signa`	`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=`	`edit_new_image_parameters(hide_dtype=True)`
`edit_new_image_parameters(hide_image_type=T`	`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()`

suite sur la page suivante

Tableau 8 – suite de la page précédente

DataLab v0.20	DataLab v1.0
`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
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):
```

(suite sur la page suivante)

(suite de la page précédente)

```
            for name, func in (("Weighted average denoise", weighted_average_denoise),
↪):
                # Wrap function to handle ImageObj objects
                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)


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
        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
```

(suite sur la page suivante)

(suite de la page précédente)

```
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
        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
```

(suite sur la page suivante)

(suite de la page précédente)

```
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"),
    )

    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(
        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",
```

(suite sur la page suivante)

(suite de la page précédente)

```
            )
```

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
```

(suite sur la page suivante)

(suite de la page précédente)

```
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)

# 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)
```

(suite sur la page suivante)

(suite de la page précédente)

```
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()

# 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)
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()

# Peak detection (signal): ``compute_peak_detection`` still exists and
# opens the interactive threshold dialog as before, but it now stores
# the result as an XY-markers table (no sticks signal is created
# automatically). To rebuild the legacy sticks signal, chain with
# ``compute_markers_to_signal`` (or call ``run_feature("extract_peak_positions",
# param)`` directly when no interactive dialog is needed):
proc.compute_peak_detection(param)        # interactive, XY-markers result
proc.compute_markers_to_signal()          # optional: rebuild sticks signal
```

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)
```

(suite sur la page suivante)

(suite de la page précédente)

```
# 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/builtin`

3.5.3 Macros

Généralités

Plusieurs méthodes permettent d'étendre les fonctionnalités de DataLab (voir Plugins ou Contrôle à distance (XML-RPC)). 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.

> **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.

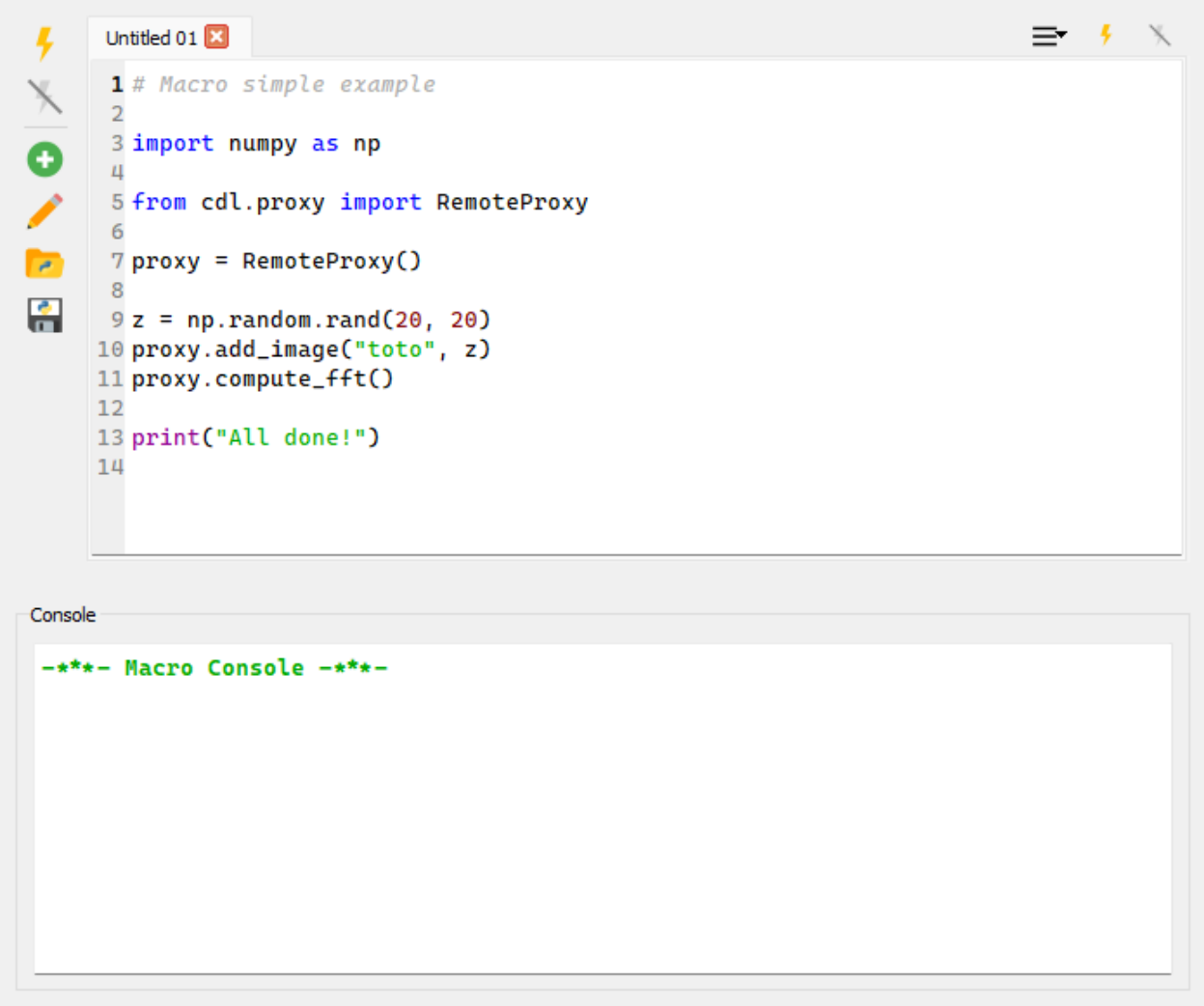


Fig. 73 – Le Gestonnaire de Macros dans DataLab.

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 (XML-RPC).

Fonctionnalités principales

Le Gestionnaire de Macros est une interface simple pour :

- Créer de nouvelles macros, en utilisant le bouton « Nouvelle macro » . Le bouton expose également un menu déroulant proposant une macro vierge et les modèles fournis (voir Modèles).
- Renommer des macros existantes, en utilisant le bouton « Renommer macro »  (ou en double-cliquant sur un onglet).
- Dupliquer la macro courante, en utilisant le bouton « Dupliquer macro » .
- Importer/exporter des macros depuis/vers des fichiers, en utilisant les boutons « Importer macro »  et « Exporter macro » .
- Rouvrir une macro d'une session précédente, en utilisant le bouton « Macros récentes… »  (la liste des macros récentes est conservée d'une session à l'autre).
- 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 .

Fonctionnalités de l'éditeur

L'éditeur de macros offre quelques fonctionnalités qui facilitent l'écriture de scripts :

- Sauvegarde automatique : chaque frappe est temporisée et persistée dans l'espace de travail courant, afin que les modifications non sauvegardées survivent à un arrêt inattendu. Une invite de récupération est proposée au démarrage lorsqu'une version sauvegardée automatiquement est détectée.
- Persistance des onglets : l'ensemble des macros ouvertes et l'onglet actif sont restaurés au démarrage suivant de DataLab, vous permettant de reprendre exactement là où vous vous étiez arrêté.
- Rechercher / Remplacer (`Ctrl+F` / `Ctrl+H`) : une barre intégrée en bas de l'éditeur prend en charge la recherche sensible à la casse, la correspondance de mots entiers, les expressions régulières et le remplacement (ou le remplacement total) dans la macro active.
- L'autocomplétion Python est activée par défaut (`Ctrl+Espace` pour la déclencher manuellement), fournie par l'éditeur de code intégré.

Console

Le panneau Console en bas capture les flux de sortie standard et d'erreur standard produits par la macro en cours d'exécution. Un clic droit dans la console expose :

— Effacer la console — efface le contenu actuel.
— Enregistrer le journal de l'historique… — exporte l'historique de la console vers un fichier texte.

Le nombre maximum de lignes conservées par la console peut être ajusté via le paramètre `Conf.macro.console_max_lines` (par défaut `5000`).

Modèles

Le menu déroulant « Nouvelle macro » liste un ensemble de modèles prêts à l'emploi couvrant des cas courants (création de signal, traitement d'image, boucles de traitement par lots, etc.). Vous pouvez fournir vos propres modèles en plaçant des fichiers `*.py` dans le répertoire des modèles utilisateur — par défaut `~/.DataLab/macro_templates/`, configurable via `Conf.macro.templates_path`. La première ligne non vide de la docstring de chaque modèle est utilisée comme titre de l'entrée du menu. Les modèles fournis avec DataLab ont priorité sur les modèles utilisateur en cas de conflit de noms.

Exemple

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

3.5.4 Assistant IA

DataLab intègre un assistant IA optionnel : un panneau de discussion ancrable, connecté à un grand modèle de langage (LLM), capable d'inspecter votre espace de travail, de créer et de traiter des signaux et des images, et d'écrire puis d'exécuter des macros Python à votre place.

> **Avertissement**
>
> L'assistant IA transmet vos messages — et, lorsque vous le lui demandez, des informations sur les objets de votre espace de travail — au fournisseur configuré. Rien n'est envoyé tant que l'assistant est désactivé ou tant que vous n'avez pas démarré de conversation. Consultez la section Confidentialité et sécurité avant d'utiliser un fournisseur distant sur des données sensibles.

Premiers pas

1. Ouvrez `Fichier > Préférences`, allez dans l'onglet Assistant IA et activez l'assistant (voir Assistant IA pour la liste complète des options).
2. Choisissez un fournisseur et un modèle, puis renseignez la clé d'API — ou, de préférence, laissez le champ vide et définissez la variable d'environnement standard du fournisseur.
3. Cliquez sur Tester la connexion pour vérifier le point d'accès avant de commencer.
4. Affichez le panneau Assistant IA depuis le menu `Affichage` s'il est masqué. Il est ancré sur le côté droit de la fenêtre principale, en onglet avec le panneau de macros.

> **Astuce**
>
> Pour découvrir l'assistant sans compte ni clé d'API, sélectionnez le fournisseur `mock` : il répond par des réponses scriptées déclenchées par de simples mots-clés, et met en œuvre toute la chaîne de traitement (appels d'outils, boîtes de dialogue de confirmation, stockage des conversations) hors ligne.

Fournisseurs pris en charge

L'assistant dialogue avec tout service exposant une API de complétion de conversation compatible OpenAI. Cela couvre notamment :

- OpenAI (`https://api.openai.com/v1`)
- GitHub Models (`https://models.github.ai/inference`)
- Azure OpenAI
- Des environnements d'exécution locaux tels que Ollama, LM Studio, llama.cpp ou vLLM

Le bouton Charger un préréglage… de la boîte de dialogue des préférences pré-remplit les champs URL de base et Modèle pour les points d'accès les plus courants.

Utilisation du panneau de discussion

Le panneau se compose d'une barre d'outils, de la vue de conversation et d'une zone de saisie :

- Nouvelle conversation : démarrer une nouvelle conversation. La précédente est conservée dans l'espace de stockage des conversations.
- Historique… : parcourir les conversations passées, les charger, les renommer ou les supprimer, et en exporter une au format Markdown. Les conversations sont stockées dans le répertoire de configuration utilisateur de DataLab.
- Envoyer / Arrêter : envoyer le message courant, ou interrompre une requête en cours. Les requêtes s'exécutent dans un fil d'exécution en arrière-plan, si bien que l'interface reste réactive.
- Le compteur de jetons à droite de la barre d'outils indique la taille du contexte de la dernière requête et la consommation cumulée de la conversation.

La zone de saisie conserve un historique des messages précédemment envoyés, navigable au clavier comme dans un terminal.

Lorsque l'assistant propose une macro, le code généré est temporaire : il est exécuté sans encombrer le panneau de macros. Un lien Enregistrer dans les macros est proposé après l'exécution si vous souhaitez la conserver.

Lorsque vous demandez à l'assistant d"écrire une macro plutôt que de l'exécuter, le code est seulement proposé : la boîte de dialogue de confirmation indique que la macro ne sera pas exécutée, et le même lien Enregistrer dans les macros est proposé. Un tel lien est également proposé sous chaque bloc de code Python des réponses de l'assistant (sauf si la création de macros est désactivée), et tous ces liens sont de nouveau disponibles lorsqu'une conversation est rouverte depuis Historique…. Une fois la macro enregistrée, un lien Afficher dans le panneau de macros fait passer ce panneau au premier plan.

Outils à la disposition de l'assistant

L'assistant n'agit pas directement sur DataLab : il ne peut appeler qu'un ensemble fixe d"outils déclarés. Les outils marqués comme étant en lecture seule ne modifient jamais l'espace de travail et peuvent être approuvés automatiquement (voir l'option correspondante) ; tous les autres outils exigent une confirmation explicite.

Outil	Lecture seule	Rôle
`list_objects`	✓	Lister les signaux ou les images d'un panneau
`get_current_panel`	✓	Renvoyer le panneau actuellement actif
`get_object_info`	✓	Inspecter un objet particulier (dimensions, unités, métadonnées, etc.)
`list_available_operations`	✓	Explorer le catalogue de traitements exposé par Sigima
`list_plugin_actions`	✓	Lister les actions apportées par les plugins tiers
`get_macro_console_output`	✓	Relire la console du panneau de macros
`get_api_help`	✓	Renvoyer l'API publique du proxy, de `SignalObj`, d'`ImageObj` ou de `sigima.params`, afin que le modèle n'invente pas de noms de méthodes
`capture_view`	✓	Capturer une image de la vue graphique courante et l'injecter dans la conversation, afin qu'un modèle multimodal puisse inspecter visuellement les données
`trigger_plugin_action`		Déclencher une action de plugin par son chemin de menu
`create_synthetic_signal`		Créer un signal synthétique (sinus, cosinus, gaussienne, bruit, rampe)
`create_synthetic_image`		Créer une image synthétique (gaussienne 2D, rampe, bruit, damier)
`load_file`		Charger un fichier dans un panneau
`apply_operation`		Exécuter n'importe quelle fonctionnalité de traitement enregistrée
`create_and_run_macro`		Créer et exécuter une macro Python, ou seulement la proposer à l'enregistrement

> **Note**
>
> L'outil `create_and_run_macro` n'est exposé au modèle que si l'option Autoriser l'IA à créer et exécuter des macros est activée. Lorsqu'elle est désactivée, l'assistant ne peut même pas proposer l'exécution de code arbitraire.

Confirmation des appels d'outils

Chaque fois que l'assistant souhaite exécuter un outil modifiant l'espace de travail, une boîte de dialogue de confirmation affiche le nom de l'outil et la valeur des paramètres. Lorsqu'une macro est proposée, la boîte de dialogue affiche également son code source avec coloration syntaxique, afin que le code puisse être relu avant d'être exécuté. Refuser l'appel renvoie ce refus au modèle, qui peut alors proposer autre chose.

Confidentialité et sécurité

— Aucune télémétrie : DataLab n'enregistre ni ne transmet le contenu de vos conversations ailleurs qu'au fournisseur que vous avez configuré.
— Fournisseurs locaux : pour que tout reste sur votre machine, faites pointer l'"URL de base vers un environnement d'exécution local (Ollama, LM Studio, llama.cpp, vLLM). Les données ne quittent alors jamais votre ordinateur.
— Stockage de la clé d'API : la clé est stockée en clair dans le fichier de configuration de DataLab. Préférez la variable d'environnement du fournisseur.
— Confirmation explicite : toute action modifiant l'espace de travail — et toute exécution de macro — exige par défaut une confirmation explicite de l'utilisateur.
— Limite d'itérations : le nombre d'appels d'outils enchaînés est borné par l'option Nombre maximum d'itérations d'appels d'outils, de sorte qu'un modèle défaillant ne puisse pas boucler indéfiniment.

Voir aussi

Le système de macros utilisé par l'assistant est décrit dans Macros, et l'API de contrôle sous-jacente dans Contrôle à distance (XML-RPC).

3.5.5 API Web

L'API Web DataLab fournit une interface moderne HTTP/JSON pour un accès externe à l'espace de travail DataLab. C'est la voie d'intégration recommandée pour les nouveaux projets, en particulier pour les notebooks Jupyter et les outils basés sur le web.

Note

L'API Web est disponible à partir de DataLab 1.1. Les dépendances requises (FastAPI, Uvicorn, Pydantic) sont incluses dans l'installation standard de DataLab.

Aperçu

L'API Web permet :

- Intégration de notebooks Jupyter via DataLab-Kernel
- Compatibilité WASM/Pyodide (pas de dépendance XML-RPC)
- Intégration d'outils externes (tout client HTTP)

Contrairement à l'interface XML-RPC, l'API Web :

- Utilise JSON pour les métadonnées et NPZ pour les données binaires (transfert efficace de grands tableaux)
- Fournit une authentification par jeton bearer pour la sécurité
- Suit les conventions REST avec documentation OpenAPI
- Fonctionne dans les environnements navigateur (WASM/Pyodide)

Tableau 9 – Comparaison avec XML-RPC

Fonctionnalité	XML-RPC	API Web
Protocole	XML	JSON + Binaire (NPZ)
Support WASM	×	✓
Grands tableaux	Lent (base64)	Rapide (NPZ)
Authentification	Aucune	Jeton Bearer
Standards	XML-RPC	REST + OpenAPI

Démarrage rapide

Activer l'API Web

Il existe plusieurs façons d'activer le serveur API Web :

1. Via l'interface : Fichier → API Web → Démarrer le serveur API Web
2. Via variable d'environnement : Définir `DATALAB_WEBAPI_ENABLED=1` avant de démarrer DataLab

Au démarrage, DataLab affiche l'URL du serveur et le jeton d'authentification dans une boîte de dialogue. La barre d'état affiche également le port de l'API Web lorsque le serveur est en cours d'exécution.

Le port par défaut est 18080 (ou le prochain port disponible s'il est occupé).

Découverte automatique

DataLab-Kernel peut automatiquement découvrir et se connecter à une instance DataLab en cours d'exécution sans aucune configuration manuelle. Lorsque l'API Web démarre, DataLab écrit les informations de connexion dans un fichier que DataLab-Kernel lit automatiquement.

Chargez simplement l'extension — aucune variable d'environnement ni connexion explicite requise :

```
# In your notebook (JupyterLab, VS Code, etc.)
%load_ext datalab_kernel

# DataLab-Kernel automatically finds and connects to DataLab
workspace.list()  # Already connected!
```

Le mécanisme de découverte automatique essaie les méthodes suivantes dans l'ordre :

1. Variables d'environnement (`DATALAB_WORKSPACE_URL`, `DATALAB_WORKSPACE_TOKEN`)
2. Fichier de connexion écrit par DataLab (Python natif uniquement)
3. Paramètres d'URL (JupyterLite : `?datalab_url=...&datalab_token=...`)
4. Sondage du port bien connu (`http://127.0.0.1:18080`)

Si la découverte échoue, DataLab-Kernel démarre en mode autonome et vous pouvez vous connecter plus tard avec `workspace.connect()`.

Connexion manuelle (historique)

Si la découverte automatique ne fonctionne pas (par exemple, en cas d'exécution sur une machine différente), vous pouvez définir des variables d'environnement avant de démarrer le noyau de votre notebook :

```
export DATALAB_WORKSPACE_URL=http://127.0.0.1:18080
export DATALAB_WORKSPACE_TOKEN=<your-token>
```

Puis dans votre notebook utilisant DataLab-Kernel :

```
from datalab_kernel import workspace

# List objects in DataLab
workspace.list()

# Retrieve an object
signal = workspace.get("my_signal")

# Add a processed result back to DataLab
workspace.add("processed", result)
```

Référence de l'API

URL de base : `http://127.0.0.1:<port>/api/v1`

Point d'accès Status (sans authentification)

Méthode	Point d'accès	Description
GET	`/status`	État du serveur et informations de version

Opérations sur les objets (authentification requise)

Tous les points d'accès ci-dessous nécessitent l'en-tête `Authorization: Bearer <token>`.

Méthode	Point d'accès	Description
GET	`/objects`	Lister tous les objets avec métadonnées (JSON)
GET	`/objects/{name}`	Obtenir les métadonnées d'un objet (JSON)
DELETE	`/objects/{name}`	Supprimer un objet
PATCH	`/objects/{name}/metadata`	Mettre à jour les métadonnées d'un objet

Transfert de données binaires (authentification requise)

Méthode	Point d'accès	Description
GET	`/objects/{name}/data`	Télécharger l'objet sous forme d'archive NPZ (prend en charge `?compress=false`)
PUT	`/objects/{name}/data`	Créer ou remplacer un objet à partir d'une archive NPZ (prend en charge `?overwrite=true`)
PUT	`/objects/{name}`	Mettre à jour sur place un objet existant à partir d'une archive NPZ, en préservant son identité, son appartenance à un groupe et sa position

Le point d'accès `GET /objects/{name}/data` diffuse une archive NPZ sous le type `application/x-npz`. Passez `?compress=false` pour désactiver la compression deflate du ZIP lorsque le client (par exemple un notebook) préfère un transfert plus rapide au prix d'une charge utile plus volumineuse. Les noms d'objets contenant des caractères non ASCII sont renvoyés dans l'en-tête `Content-Disposition` au moyen de la forme `filename*` de la RFC 5987.

Le point d'accès `PUT /objects/{name}/data` crée un nouvel objet portant le nom indiqué, ou renvoie `409 Conflict` s'il existe déjà. Passez `?overwrite=true` pour remplacer l'objet existant de manière atomique.

Calcul (authentification requise)

Méthode	Point d'accès	Description
POST	`/select`	Sélectionner des objets dans un panneau pour les calculs ultérieurs
POST	`/calc`	Exécuter une fonction de calcul DataLab sur la sélection courante

Format des données

Les objets sont transférés au format NPZ de NumPy pour plus d'efficacité. Le format est implémenté par Sigima (`sigima.io.serialize_object_to_npz` et `sigima.io.deserialize_object_from_npz`), de sorte que les clients Python peuvent lire et écrire ces archives avec le même code que DataLab.

Structure NPZ de SignalObj

```
my_signal.npz
├── x.npy        # X coordinates (float64)
├── y.npy        # Y data (float64)
├── dx.npy       # X uncertainties (optional)
├── dy.npy       # Y uncertainties (optional)
└── metadata.json  # Labels, units, title
```

Structure NPZ de ImageObj

```
my_image.npz
├── data.npy      # 2D image array (preserves dtype)
├── xcoords.npy   # X pixel centers (only for non-uniform coordinates)
├── ycoords.npy   # Y pixel centers (only for non-uniform coordinates)
└── metadata.json  # Labels, units, coordinates
                   # (x0, y0, dx, dy, is_uniform_coords)
```

Authentification

Tous les points d'accès de l'API sauf `/status` nécessitent un jeton bearer :

```
Authorization: Bearer <token>
```

Le jeton est généré au démarrage du serveur et affiché dans la boîte de dialogue de connexion.

Variables d'environnement

Variable	Description	Valeur par défaut
`DATALAB_WEBAPI_HOST`	Adresse de liaison	`127.0.0.1`
`DATALAB_WEBAPI_PORT`	Port du serveur	`18080` (ou le suivant disponible)
`DATALAB_WEBAPI_TOKEN`	Jeton d'authentification	Généré

Modèle de sécurité

L'API Web implémente les mesures de sécurité suivantes :

1. Liaison localhost : Par défaut, le serveur n'accepte que les connexions provenant de la machine locale (127.0.0.1).
2. Authentification par jeton : Chaque requête (sauf status) doit inclure un jeton bearer valide.
3. Opt-in explicite : La liaison distante (0.0.0.0) nécessite une configuration explicite.

Contournement du jeton pour localhost

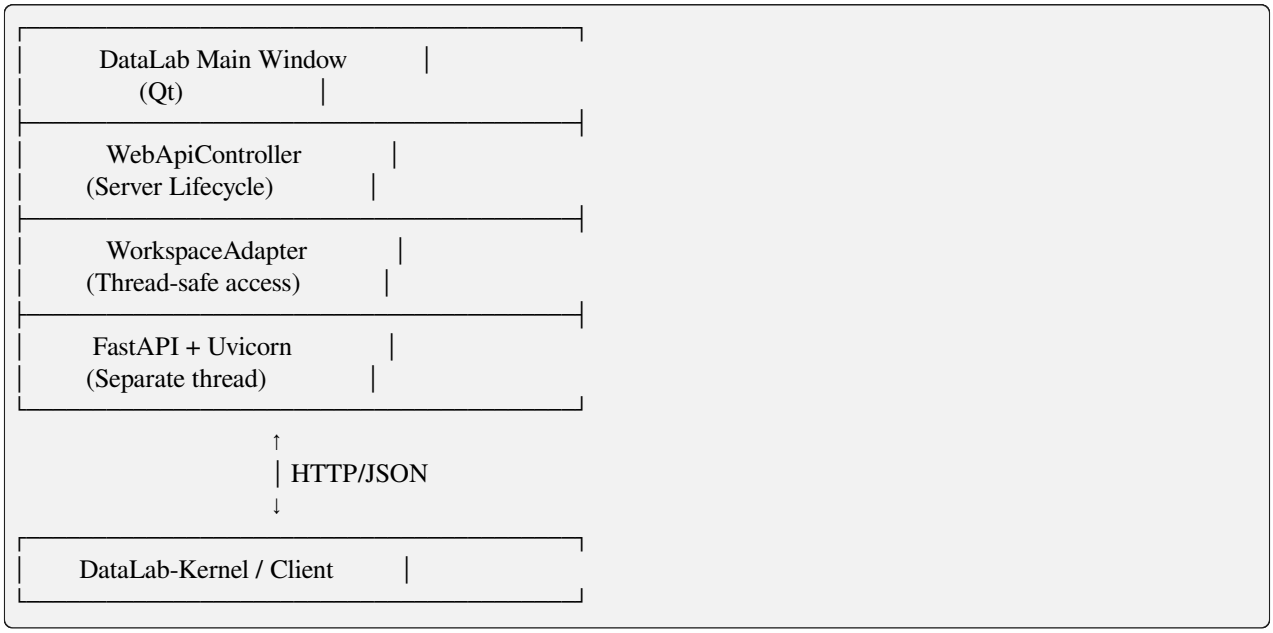
Pour simplifier le développement local, vous pouvez désactiver la vérification du jeton pour les connexions localhost dans Édition → Paramètres → Contournement localhost de l'API Web.

Lorsque cette option est activée, les clients se connectant depuis `127.0.0.1` n'ont pas besoin de fournir de jeton. Cela permet à la découverte automatique de fonctionner de manière transparente même lorsque le fichier de connexion ne peut pas être lu (par exemple, dans JupyterLite ou les environnements sandboxés).

> **Avertissement**
>
> N'activez le contournement localhost que lorsque vous contrôlez toutes les applications s'exécutant sur votre machine. Un logiciel malveillant local pourrait accéder à vos données sans jeton.

> **Avertissement**
>
> Exposer l'API sur le réseau permet à quiconque possédant le jeton d'accéder à vos données. À utiliser avec précaution uniquement sur des réseaux de confiance.

Architecture



L'API Web s'exécute dans un thread séparé de l'interface Qt. Le `WorkspaceAdapter` assure un accès thread-safe aux objets DataLab en acheminant les appels vers le thread principal via le mécanisme signal/slot de Qt.

Installation

Les dépendances de l'API Web sont incluses dans l'installation standard de DataLab :

- FastAPI (framework web)
- Uvicorn (serveur ASGI)
- Pydantic (validation de données)

Aucune étape d'installation supplémentaire n'est nécessaire.

Contrôle programmatique

L'API Web peut également être démarrée et contrôlée par programmation via l'interface XML-RPC, ce qui est utile pour les flux de travail automatisés :

```
from datalab.control.remote import RemoteClient

proxy = RemoteClient()
proxy.connect()

# Start the Web API server
info = proxy.start_webapi_server()
print(f"URL: {info['url']}")
print(f"Token: {info['token']}")

# Check status
status = proxy.get_webapi_status()
print(f"Running: {status['running']}")

# Stop the server
proxy.stop_webapi_server()
```

Voir aussi

— Contrôle à distance (XML-RPC) pour l'interface XML-RPC (historique)
— Documentation DataLab-Kernel pour l'intégration Jupyter

3.5.6 Contrôle à distance (XML-RPC)

> **Note**
>
> Pour les nouveaux projets, nous recommandons d'utiliser l'" API Web au lieu de XML-RPC. L'API Web offre de meilleures performances pour les grands tableaux, une authentification basée sur des jetons et une compatibilité avec les environnements de navigateur (WASM/Pyodide).
>
> XML-RPC reste disponible pour la compatibilité avec les intégrations existantes.

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 alternative légère pour contrôler DataLab à distance (c'est-à-dire sans installer l'ensemble du package DataLab et ses dépendances dans votre environnement), utilisez la classe `sigima.client.SimpleRemoteProxy` fournie par le package Sigima (`pip install sigima`). Elle expose les mêmes points d'entrée XML-RPC que la classe `RemoteProxy` décrite ci-dessous, avec un ensemble minimal de dépendances.
>
> Cette classe remplace `cdlclient.SimpleRemoteProxy` : le package `cdlclient` est obsolète et archivé depuis DataLab 1.3. Sa boîte de dialogue de connexion est désormais fournie par SigimaX (voir Boîte de dialogue de connexion ci-dessous).

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`

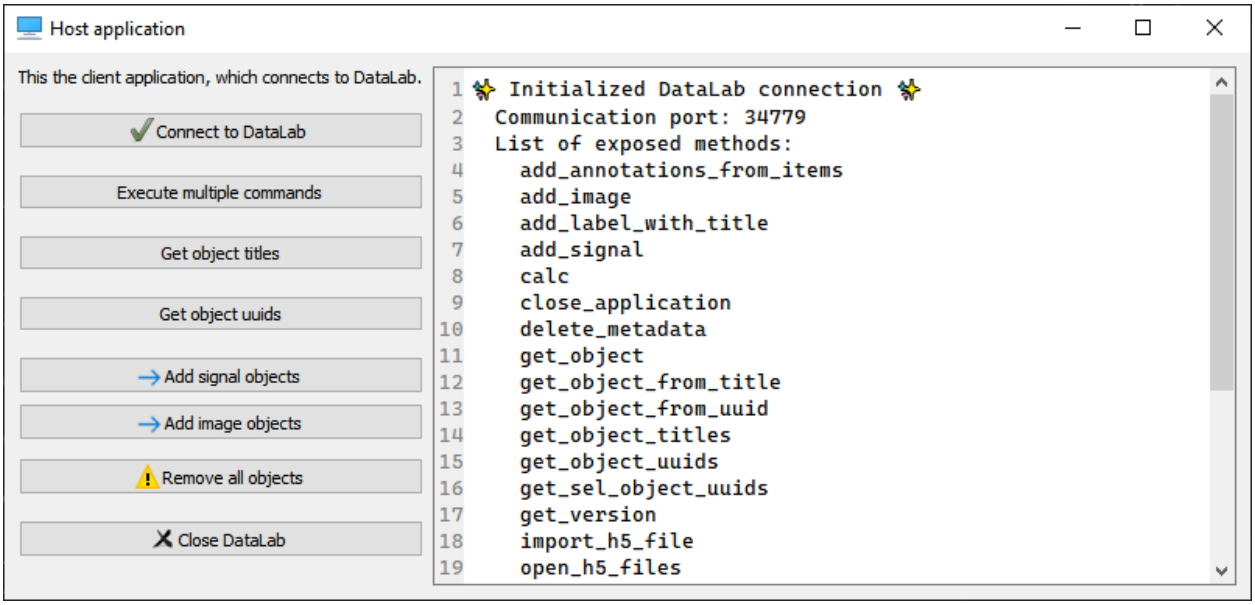


Fig. 74 – Capture d'écran du test de l'application cliente distante (`datalab.tests.features.control.remoteclient_app_test`)

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()

# %% 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)
```

(suite sur la page suivante)

(suite de la page précédente)

```
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()
    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):
```

(suite sur la page suivante)

(suite de la page précédente)

```
        """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()

    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):
```

(suite sur la page suivante)

(suite de la page précédente)

```
        """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
        )

    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,
```

(suite sur la page suivante)

(suite de la page précédente)

```
            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

Une boîte de dialogue de connexion peut être utilisée pour se connecter à une instance de DataLab en cours d'exécution. Elle est exposée par la classe `sigimax.widgets.connection.ConnectionDialog`, et accepte les arguments optionnels `icon` et `banner` pour porter l'identité visuelle de l'application cliente. Elle fonctionne avec tout proxy exposant une méthode bloquante `connect()`, c'est-à-dire à la fois `RemoteProxy` et `sigima.client.SimpleRemoteProxy`.

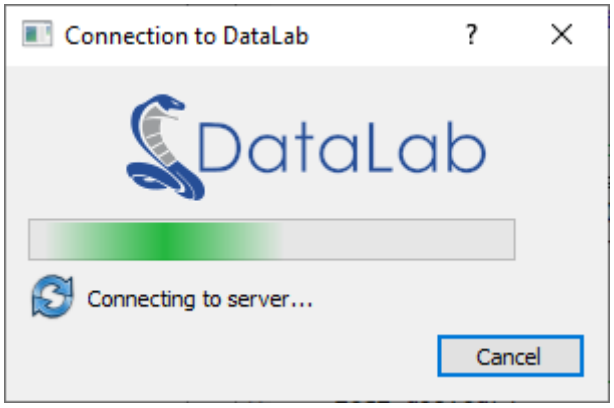


Fig. 75 – Capture d'écran de la boîte de dialogue de connexion (`sigimax.widgets.connection.ConnectionDialog`)

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.configtools import get_icon, get_image_file_path
from guidata.qthelpers import qt_app_context
from qtpy import QtGui as QG
from qtpy import QtWidgets as QW
from sigimax.widgets.connection import ConnectionDialog

from datalab.control.proxy import RemoteProxy


def test_dialog():
    """Test connection dialog"""
    proxy = RemoteProxy(autoconnect=False)
    with qt_app_context():
        dlg = ConnectionDialog(
            proxy.connect,
            icon=get_icon("DataLab.svg"),
            banner=QG.QPixmap(get_image_file_path("DataLab-Banner-200.png")),
        )
        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'
```

(suite sur la page suivante)

(suite de la page précédente)

```
>>> 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

Définir le port XML-RPC auquel se connecter.

Paramètres

port – Port XML-RPC auquel se connecter. Si None, le port est automatiquement récupéré depuis la configuration de DataLab.

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

Essayer de se connecter au serveur XML-RPC de DataLab.

Paramètres

- **port** – Port XML-RPC auquel se connecter. Si non spécifié, le port est automatiquement récupéré depuis la configuration de DataLab.
- **timeout** – Temps maximum d'attente de connexion en secondes. Par défaut 5.0. Il s'agit du temps d'attente maximum total, et non par tentative.
- **retries** – Nombre de tentatives. Par défaut 10. Ce paramètre est déprécié et sera supprimé dans une version future (conservé pour la compatibilité ascendante).

Lève

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

disconnect() → None

Se déconnecter du serveur XML-RPC de DataLab.

is_connected() → bool

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

get_method_list() → list[str]

Renvoie la liste des méthodes disponibles.

add_signal(title: str, xdata: ndarray, ydata: ndarray, xunit: str = '', yunit: str = '', xlabel: str = '', ylabel: str = '', group_id: str = '', set_current: bool = True) → bool

Ajouter des données de signal à DataLab.

Paramètres

- **title** – Titre du signal
- **xdata** – Données X
- **ydata** – Données Y
- **xunit** – Unité X. Par défaut « «
- **yunit** – Unité Y. Par défaut « «
- **xlabel** – Libellé X. Par défaut « «
- **ylabel** – Libellé Y. Par défaut « «
- **group_id** – Identifiant du groupe dans lequel ajouter le signal. Par défaut « «
- **set_current** – si True, définit le signal ajouté comme courant

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

Lève
— **ValueError** – Type de données xdata invalide
— **ValueError** – Type de données ydata invalide

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

Ajouter des données d'image à DataLab.

Paramètres
— **title** – Titre de l'image
— **data** – Données de l'image
— **xunit** – Unité X. Par défaut « «
— **yunit** – Unité Y. Par défaut « «
— **zunit** – Unité Z. Par défaut « «
— **xlabel** – Libellé X. Par défaut « «
— **ylabel** – Libellé Y. Par défaut « «
— **zlabel** – Libellé Z. Par défaut « «
— **group_id** – Identifiant du groupe dans lequel ajouter l'image. Par défaut « «
— **set_current** – si True, définit l'image ajoutée comme courante

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

Lève
ValueError – Type de données invalide

add_object(obj: SignalObj | ImageObj, group_id: str = '', set_current: bool = True) → None

Ajouter un objet à DataLab.

Paramètres
— **obj** – Objet signal ou image
— **group_id** – Identifiant du groupe dans lequel ajouter l'objet. Par défaut « «
— **set_current** – si True, définit l'objet ajouté comme courant

set_object(obj: SignalObj | ImageObj) → None

Définir les données d'un objet dans DataLab.

Mettre à jour un objet existant dans DataLab avec les nouvelles données de `obj`. L'objet est identifié par son UUID (qui est porté par `obj` depuis un appel précédent à *`get_object()`*).

Paramètres
obj – Objet signal ou image (doit avoir le même UUID qu'un objet existant dans DataLab)

Lève
KeyError – si aucun objet avec un UUID correspondant n'est trouvé

load_from_files(filenames: list[str], param: DataSet | None = None) → None

Ouvrir des objets depuis des fichiers dans le panneau courant (signaux/images).

Paramètres
— **filenames** – liste des noms de fichiers
— **param** – Paramètres d'import optionnels propres au format, appliqués à chaque fichier qui en a besoin, sans afficher de boîte de dialogue (par exemple `sigima.io.RawImageImportParam` pour les images RAW). Par défaut None.

load_from_directory(path: str, param: DataSet | None = None) → None

Ouvrir des objets depuis un répertoire dans le panneau courant (signaux/images).

Paramètres
— **path** – chemin du répertoire

— **param** – Paramètres d'import optionnels propres au format, appliqués à chaque fichier qui en a besoin, sans afficher de boîte de dialogue (par exemple `sigima.io.RawImageImportParam` pour les images RAW). Par défaut None.

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

Appeler la fonctionnalité de calcul `name`

> **Note**
>
> Ceci appelle soit la méthode `compute_<name>` du processeur (si elle existe), soit la fonctionnalité de calcul `<name>` du processeur (si elle est enregistrée, en utilisant la méthode `run_feature`). La fonction est recherchée dans tous les panneaux, en commençant par le panneau courant.

Paramètres
- **name** – Nom de la fonction de calcul
- **param** – Paramètre de la fonction de calcul. Par défaut None.

Lève
ValueError – fonction inconnue

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

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

Paramètres
- **nb_id_title** – Numéro de l'objet, identifiant de l'objet, ou titre de l'objet. Par défaut None (objet courant).
- **panel** – Nom du panneau. Par défaut None (panneau courant).

Renvoie
Objet

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

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

Obtenir les formes des éléments graphiques associées à l'objet (signal/image).

Paramètres
- **nb_id_title** – Numéro de l'objet, identifiant de l'objet, ou titre de l'objet. Par défaut None (objet courant).
- **panel** – Nom du panneau. Par défaut None (panneau courant).

Renvoie
Liste des formes des éléments graphiques

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

Ajouter des annotations à l'objet (éléments graphiques d'annotation).

Paramètres
- **items** – éléments graphiques d'annotation
- **refresh_plot** – rafraîchir le graphique. Par défaut True.
- **panel** – nom du panneau (valeurs valides : « signal », « image »). Si None, le panneau courant est utilisé.

call_method(method_name: str, *args, panel: str | None = None, **kwargs)

Appeler une méthode publique sur un panneau ou la fenêtre principale.

Ordre de résolution des méthodes quand panel est None : 1. Essayer la fenêtre principale (DLMainWindow) 2. Si non trouvé, essayer le panneau courant (BaseDataPanel)

Paramètres

— **method_name** – Nom de la méthode à appeler
— ***args** – Arguments positionnels à passer à la méthode
— **panel** – Nom du panneau (« signal », « image », ou None pour la détection automatique). Par défaut None.
— ****kwargs** – Arguments nommés à passer à la méthode

Renvoie
La valeur de retour de la méthode appelée

Lève
— **AttributeError** – Si la méthode n'existe pas ou n'est pas publique
— **ValueError** – Si le nom du panneau est invalide

start_webapi_server(host: str = '127.0.0.1', port: int = 8080) → dict[str, str | int]

Démarrer le serveur WebAPI.

Paramètres
— **host** – Adresse hôte à laquelle se lier. Par défaut « 127.0.0.1 ».
— **port** – Numéro de port. Par défaut 8080.

Renvoie
Dictionnaire avec les informations du serveur incluant "url" et "token".

stop_webapi_server() → bool

Arrêter le serveur WebAPI.

Renvoie
True si le serveur a été arrêté, False s'il n'était pas en cours d'exécution.

get_webapi_status() → dict[str, str | int | bool]

Obtenir l'état actuel du serveur WebAPI.

Renvoie
Dictionnaire avec les informations d'état incluant "running", "url" et "port".

add_group(title: str, panel: str | None = None, select: bool = False) → str

Ajouter un groupe à DataLab.

Paramètres
— **title** – Titre du groupe
— **panel** – Nom du panneau (valeurs valides : « signal », « image »). Par défaut None.
— **select** – Sélectionner le groupe après sa création. Par défaut False.

Renvoie
UUID du groupe créé

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

Ajouter un libellé avec le titre de l'objet sur le graphique associé

Paramètres
— **title** – Titre du libellé. Par défaut None. Si None, le titre est le titre de l'objet.
— **panel** – nom du panneau (valeurs valides : « signal », « image »). Si None, le panneau courant est utilisé.

close_application() → None

Fermer l'application DataLab

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

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

Renvoie
Gestionnaire de contexte

Exemple

```
>>> with proxy.context_no_refresh():
...     proxy.add_image("image1", data1)
...     proxy.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

Supprimer les métadonnées des objets sélectionnés

Paramètres
- **refresh_plot** – Rafraîchir le graphique. Par défaut True.
- **keep_roi** – Conserver les ROI. Par défaut False.

export_capsule(filename: str, name: str | None = None, description: str | None = None, as_revision: bool = True, workspace_ref: str | None = None, workspace_filename: str | None = None, attachments: list[str] | None = None) → str

Exporter l'espace de travail sous forme de capsule (espace de travail et sa provenance).

Paramètres
- **filename** – Nom du fichier de la capsule (`.dlcapsule`).
- **name** – Nom de la capsule.
- **description** – Description de la capsule.
- **as_revision** – Lorsque l'espace de travail a été ouvert depuis une capsule, désigner cette capsule comme révision parente. Par défaut True.
- **workspace_ref** – IRI absolue où l'espace de travail sera publié : la capsule référence alors l'espace de travail au lieu de l'inclure.
- **workspace_filename** – Nom du fichier de l'espace de travail publié à workspace_ref (par défaut : dernier segment de l'IRI, à côté de la capsule).
- **attachments** – Fichiers légers déclarés dans la capsule (16 Mio au plus chacun), jamais ouverts ni exécutés.

Renvoie
Empreinte du manifeste de la capsule.

get_current_object_uuid() → str | None

Renvoie l'UUID de l'objet courant dans le panneau courant.

Renvoie
UUID de l'objet courant, ou None si aucun objet n'est courant.

get_current_panel() → str

Renvoie le nom du panneau courant.

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]]]

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

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

Type renvoyé
Tuple

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

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

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

Renvoie
Liste des titres des objets

Lève
ValueError – si le panneau n'est pas trouvé

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

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

Paramètres
— **panel** – nom du panneau (valeurs valides : « signal », « image »). Si None, le panneau courant est utilisé.
— **group** – Numéro du groupe, identifiant du groupe, ou titre du groupe. Par défaut None (tous les groupes).

Renvoie
Liste des UUID des objets

Lève
ValueError – si le panneau n'est pas trouvé

classmethod get_public_methods() → list[str]

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

Renvoie
Liste des méthodes publiques

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

Renvoie les UUID des objets sélectionnés.

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

Renvoie
Liste des UUID des objets sélectionnés.

get_version() → str

Renvoie la version publique de DataLab.

Renvoie
Version de DataLab

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

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

Paramètres
— **filename** – Nom du fichier HDF5
— **reset_all** – Réinitialiser toutes les données de l'application. Par défaut None.

import_macro_from_file(filename: str) → None

Importer une macro depuis un fichier

Paramètres
filename – Nom du fichier.

load_h5_workspace(h5files: list[str], reset_all: bool = False) → None

Charger des fichiers d'espace de travail HDF5 natifs DataLab sans aucun élément d'interface graphique.

Cette méthode peut être appelée en toute sécurité depuis des scripts (par exemple, console interne, macros) car elle ne crée aucun widget Qt, boîte de dialogue ou barre de progression.

> **Avertissement**
>
> Cette méthode ne prend en charge que les fichiers HDF5 natifs DataLab. Pour importer des fichiers HDF5 arbitraires (non natifs), utilisez plutôt *open_h5_files()* ou *import_h5_file()*.

Paramètres
- **h5files** – Liste des noms de fichiers HDF5 natifs DataLab
- **reset_all** – Réinitialiser toutes les données de l'application avant l'importation. Par défaut False.

Lève
- **ValueError** – Si un fichier n'est pas un fichier HDF5 natif DataLab valide

open_capsule(filename: str, workspace: str | None = None) → None

Ouvrir une capsule comme espace de travail, en remplaçant l'espace de travail actuel.

Paramètres
- **filename** – Nom du fichier de la capsule (`.dlcapsule`).
- **workspace** – Fichier de l'espace de travail d'une capsule qui référence son espace de travail au lieu de l'inclure.

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

Ouvrir un fichier HDF5 DataLab ou importer depuis tout autre fichier HDF5.

Paramètres
- **h5files** – Liste des fichiers HDF5 à ouvrir. Par défaut None.
- **import_all** – Importer tous les objets depuis les fichiers HDF5. Par défaut None.
- **reset_all** – Réinitialiser toutes les données de l'application. Par défaut None.

raise_window() → None

Mettre la fenêtre DataLab au premier plan

remove_object(force: bool = False) → None

Supprimer l'objet courant du panneau courant.

Paramètres
- **force** – si True, supprimer l'objet sans confirmation. Par défaut False.

reset_all() → None

Réinitialiser toutes les données de l'application

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

Exécuter une macro.

Paramètres
- **number_or_title** – Numéro de la macro, ou titre de la macro. Par défaut None (macro courante).

Lève
- **ValueError** – si la macro n'est pas trouvée

save_h5_workspace(filename: str) → None

Enregistrer l'espace de travail courant dans un fichier HDF5 natif DataLab sans éléments d'interface graphique.

Cette méthode peut être appelée en toute sécurité depuis des scripts (par exemple, console interne, macros) car elle ne crée aucun widget Qt, boîte de dialogue ou barre de progression.

Paramètres
filename – Nom du fichier HDF5 dans lequel enregistrer

Lève
IOError – Si le fichier ne peut pas être enregistré

save_to_h5_file(filename: str) → None

Enregistrer dans un fichier HDF5 DataLab.

Paramètres
filename – Nom du fichier HDF5

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

Sélectionner des groupes dans le panneau courant.

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

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

Sélectionner des objets dans le panneau courant.

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

set_current_panel(panel: str) → None

Basculer vers le panneau.

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

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

Arrêter la macro.

Paramètres
number_or_title – Numéro de la macro, ou titre de la macro. Par défaut None (macro courante).

Lève
ValueError – si la macro n'est pas trouvée

toggle_auto_refresh(state: bool) → None

Basculer l'état du rafraîchissement automatique.

Paramètres
state – État du rafraîchissement automatique

toggle_show_titles(state: bool) → None

Basculer l'état d'affichage des titres.

Paramètres
state – État d'affichage des titres

3.5.7 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.

3.5.8 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.

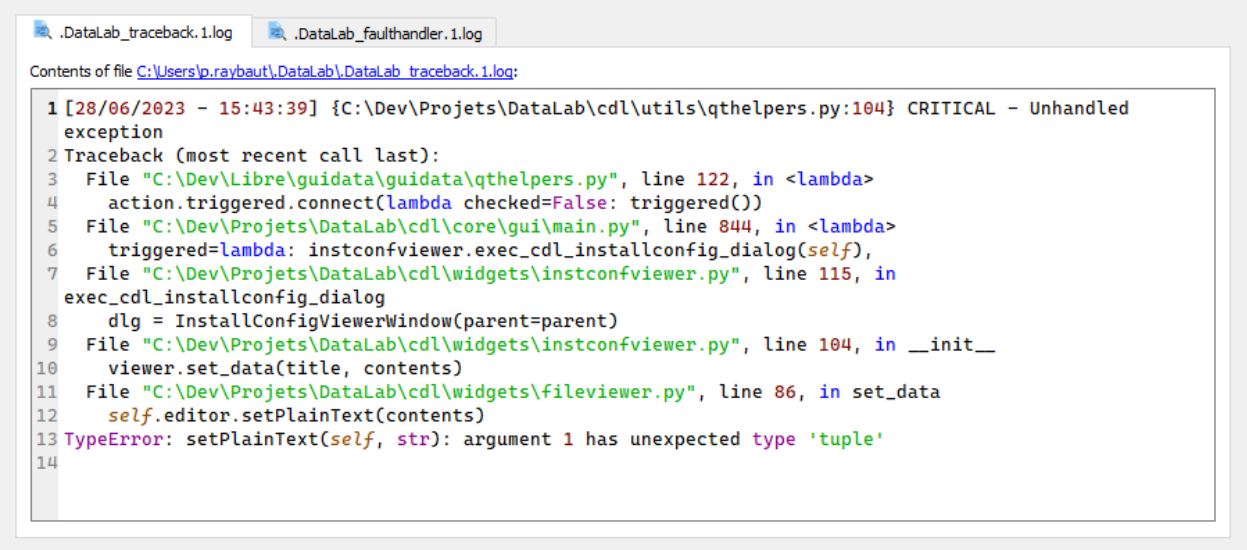


Fig. 76 – Journaux de bord DataLab (voir le menu « ? »)

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).

3.5.9 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

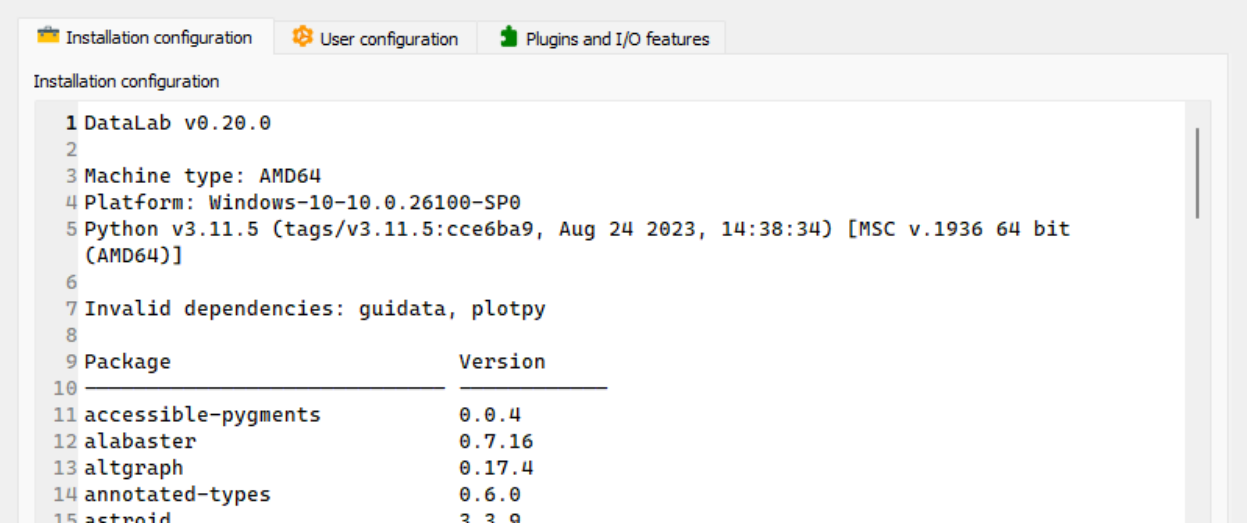


Fig. 77 – Installation et configuration (voir menu « ? »), onglet 1

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

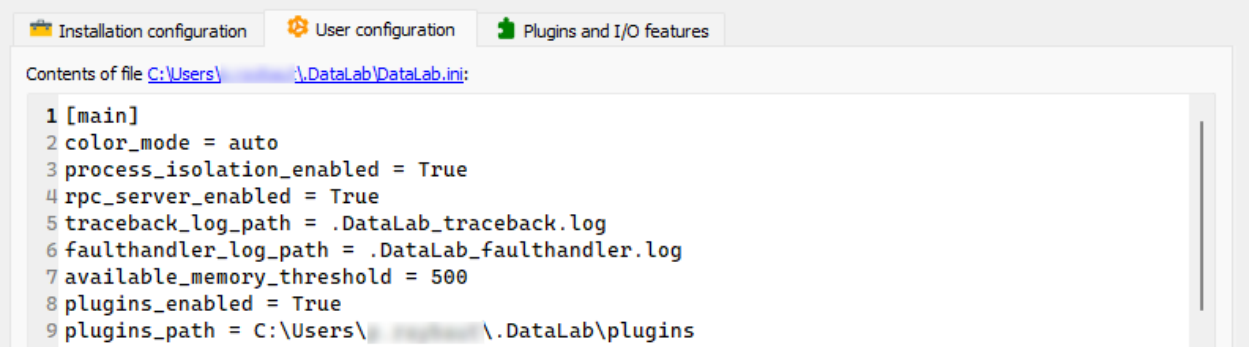


Fig. 78 – Installation et configuration (voir menu « ? »), onglet 2

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

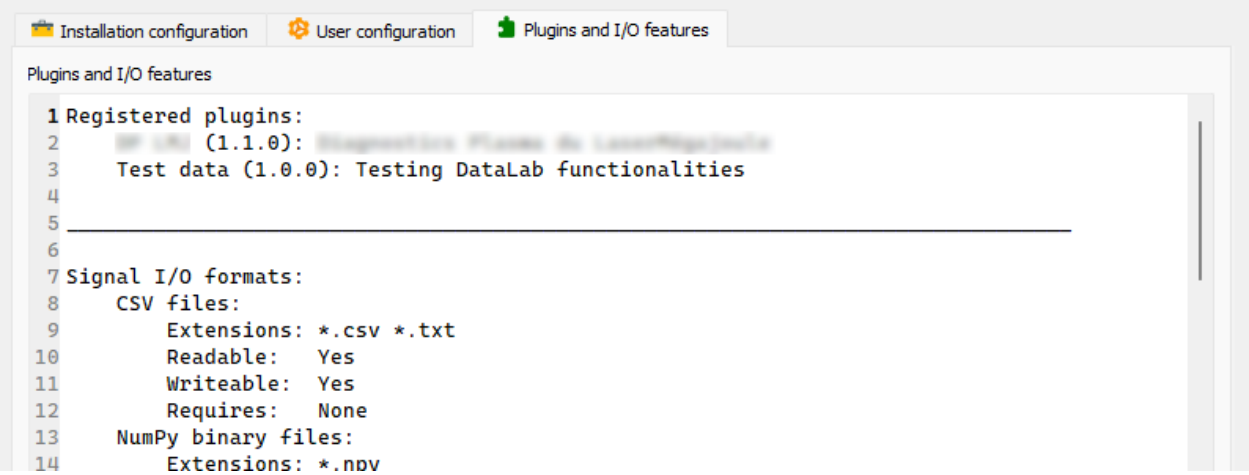


Fig. 79 – Installation et configuration (voir menu « ? »), onglet 3

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).

3.5.10 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] [--reset] [--unattended] [--accept_dialogs]
              [--screenshot] [--screenshot_path SCREENSHOT_PATH]
              [--delay DELAY] [--xmlrpcport XMLRPCPORT]
              [--verbose {quiet,normal,debug}] [--smoke-test [REPORT]]
              [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
  --screenshot_path SCREENSHOT_PATH
                        path to save 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
  --smoke-test [REPORT]
                        check the installation, write a JSON report (default:
                        datalab_smoke_test.json) and exit with a non-zero code
                        if a check failed
```

Ouvrir un fichier HDF5 au démarrage

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

```
$ 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
```

Vérifier une installation

Pour vérifier que DataLab est correctement installé, sans ouvrir sa fenêtre principale, utiliser l'une des commandes suivantes (`DataLab.exe` est l'exécutable de la version autonome pour Windows, dans son dossier d'installation):

```
$ datalab --smoke-test /path/to/report.json
$ DataLab.exe --smoke-test C:\path\to\report.json
```

DataLab vérifie les packages dont il dépend, les ressources fournies avec lui (modèles de macros, documentation PDF, traductions françaises et icônes) et le traitement dans un processus séparé, écrit les résultats dans le rapport JSON, puis s'arrête. Sans chemin, le rapport est écrit sous le nom `datalab_smoke_test.json` dans le dossier courant, qui doit être accessible en écriture. Le code de sortie vaut 0 si toutes les vérifications ont réussi, 1 sinon : le rapport donne la cause de chaque échec.

> **Note**
>
> La version autonome n'affiche rien dans une console : consulter le rapport. Si elle ne peut pas démarrer du tout, elle s'arrête avec le code 1 et écrit la cause dans `datalab_error.log`, dans le dossier courant.

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

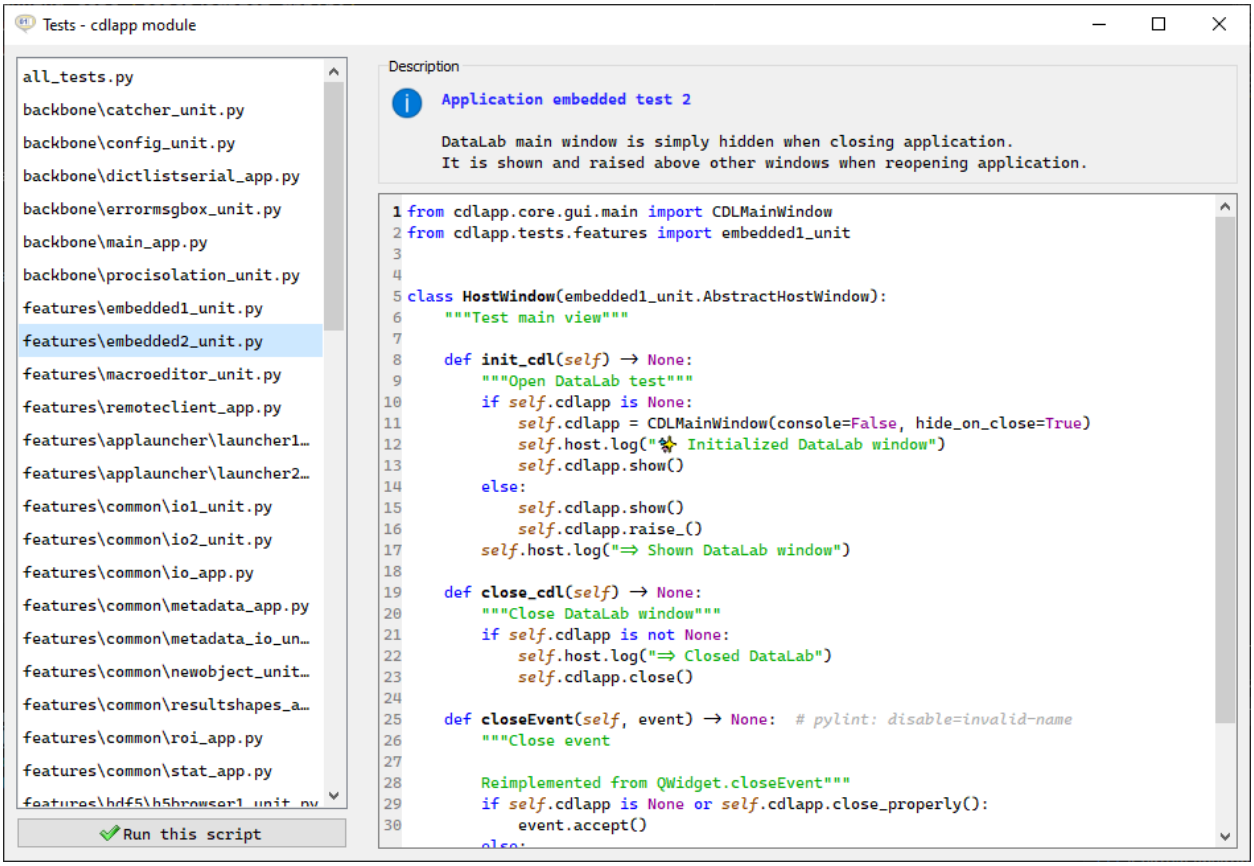
Pour exécuter l'ensemble des tests unitaires de DataLab, taper la commande suivante:

```
$ pytest
```

Exécuter les tests interactifs

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

```
$ datalab-tests
```



3.5.11 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).

> **Note**
>
> Une partie significative de la couche graphique de DataLab est fournie par le package sigimax : fenêtre principale générique, système de configuration, widgets de tracé amovibles, espace de travail et navigateur HDF5, visionneuse de journaux et dialogues scientifiques. Ces classes sont documentées dans la documentation de SigimaX, et sont celles à consulter lors de la construction d'une application scientifique Qt sur la même base (voir La plateforme DataLab).

Objets proxy (*datalab.control.proxy*)

Le module *datalab.control.proxy* permet d'accéder aux fonctionnalités de DataLab à partir d'une classe proxy.

Proxy distant

Le proxy distant est utilisé lorsque DataLab est lancé dans un processus différent de celui du proxy. Dans ce cas, le proxy se connecte au serveur XML-RPC de DataLab.

class datalab.control.proxy.**RemoteProxy**(autoconnect: bool = True)

Classe proxy distante de DataLab.

Cette classe donne accès aux fonctionnalités de DataLab à partir d'une classe proxy. Il s'agit de la version distante du proxy, utilisée lorsque DataLab est lancé dans un processus différent de celui du proxy.

Paramètres
- **autoconnect** – Se connecter automatiquement au serveur XML-RPC de DataLab.

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

Exemples

Voici un exemple simple d'utilisation de RemoteProxy dans un script Python ou un notebook Jupyter :

```
>>> 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']
```

(suite sur la page suivante)

(suite de la page précédente)

```
>>> 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

Ajouter des annotations à l'objet (éléments graphiques d'annotation).

Paramètres

- **items** – éléments graphiques d'annotation
- **refresh_plot** – rafraîchir le graphique. Par défaut True.
- **panel** – nom du panneau (valeurs valides : « signal », « image »). Si None, le panneau courant est utilisé.

add_group(title: str, panel: str | None = None, select: bool = False) → str

Ajouter un groupe à DataLab.

Paramètres

- **title** – Titre du groupe
- **panel** – Nom du panneau (valeurs valides : « signal », « image »). Par défaut None.
- **select** – Sélectionner le groupe après sa création. Par défaut False.

Renvoie

UUID du groupe créé

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

Ajouter des données d'image à DataLab.

Paramètres

- **title** – Titre de l'image
- **data** – Données de l'image
- **xunit** – Unité X. Par défaut « «
- **yunit** – Unité Y. Par défaut « «
- **zunit** – Unité Z. Par défaut « «
- **xlabel** – Libellé X. Par défaut « «
- **ylabel** – Libellé Y. Par défaut « «
- **zlabel** – Libellé Z. Par défaut « «
- **group_id** – Identifiant du groupe dans lequel ajouter l'image. Par défaut « «
- **set_current** – si True, définit l'image ajoutée comme courante

Renvoie

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

Lève

ValueError – Type de données invalide

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

Ajouter un libellé avec le titre de l'objet sur le graphique associé

Paramètres

- **title** – Titre du libellé. Par défaut None. Si None, le titre est le titre de l'objet.
- **panel** – nom du panneau (valeurs valides : « signal », « image »). Si None, le panneau courant est utilisé.

add_object(obj: SignalObj | ImageObj, group_id: str = '', set_current: bool = True) → None

Ajouter un objet à DataLab.

Paramètres
- **obj** – Objet signal ou image
- **group_id** – Identifiant du groupe dans lequel ajouter l'objet. Par défaut « «
- **set_current** – si True, définit l'objet ajouté comme courant

add_signal(title: str, xdata: ndarray, ydata: ndarray, xunit: str = '', yunit: str = '', xlabel: str = '', ylabel: str = '', group_id: str = '', set_current: bool = True) → bool

Ajouter des données de signal à DataLab.

Paramètres
- **title** – Titre du signal
- **xdata** – Données X
- **ydata** – Données Y
- **xunit** – Unité X. Par défaut « «
- **yunit** – Unité Y. Par défaut « «
- **xlabel** – Libellé X. Par défaut « «
- **ylabel** – Libellé Y. Par défaut « «
- **group_id** – Identifiant du groupe dans lequel ajouter le signal. Par défaut « «
- **set_current** – si True, définit le signal ajouté comme courant

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

Lève
- **ValueError** – Type de données xdata invalide
- **ValueError** – Type de données ydata invalide

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

Appeler la fonctionnalité de calcul `name`

> **Note**
>
> Ceci appelle soit la méthode `compute_<name>` du processeur (si elle existe), soit la fonctionnalité de calcul `<name>` du processeur (si elle est enregistrée, en utilisant la méthode `run_feature`). La fonction est recherchée dans tous les panneaux, en commençant par le panneau courant.

Paramètres
- **name** – Nom de la fonction de calcul
- **param** – Paramètre de la fonction de calcul. Par défaut None.

Lève
ValueError – fonction inconnue

call_method(method_name: str, *args, panel: str | None = None, **kwargs)

Appeler une méthode publique sur un panneau ou la fenêtre principale.

Ordre de résolution des méthodes quand panel est None : 1. Essayer la fenêtre principale (DLMainWindow) 2. Si non trouvé, essayer le panneau courant (BaseDataPanel)

Paramètres
- **method_name** – Nom de la méthode à appeler
- ***args** – Arguments positionnels à passer à la méthode
- **panel** – Nom du panneau (« signal », « image », ou None pour la détection automatique). Par défaut None.
- ****kwargs** – Arguments nommés à passer à la méthode

Renvoie
La valeur de retour de la méthode appelée

Lève

— **AttributeError** – Si la méthode n'existe pas ou n'est pas publique
— **ValueError** – Si le nom du panneau est invalide

close_application() → None

Fermer l'application DataLab

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

Essayer de se connecter au serveur XML-RPC de DataLab.

Paramètres
— **port** – Port XML-RPC auquel se connecter. Si non spécifié, le port est automatiquement récupéré depuis la configuration de DataLab.
— **timeout** – Temps maximum d'attente de connexion en secondes. Par défaut 5.0. Il s'agit du temps d'attente maximum total, et non par tentative.
— **retries** – Nombre de tentatives. Par défaut 10. Ce paramètre est déprécié et sera supprimé dans une version future (conservé pour la compatibilité ascendante).

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

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

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

Renvoie
Gestionnaire de contexte

Exemple

```
>>> with proxy.context_no_refresh():
...     proxy.add_image("image1", data1)
...     proxy.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

Supprimer les métadonnées des objets sélectionnés

Paramètres
— **refresh_plot** – Rafraîchir le graphique. Par défaut True.
— **keep_roi** – Conserver les ROI. Par défaut False.

disconnect() → None

Se déconnecter du serveur XML-RPC de DataLab.

export_capsule(filename: str, name: str | None = None, description: str | None = None, as_revision: bool = True, workspace_ref: str | None = None, workspace_filename: str | None = None, attachments: list[str] | None = None) → str

Exporter l'espace de travail sous forme de capsule (espace de travail et sa provenance).

Paramètres
— **filename** – Nom du fichier de la capsule (`.dlcapsule`).
— **name** – Nom de la capsule.
— **description** – Description de la capsule.
— **as_revision** – Lorsque l'espace de travail a été ouvert depuis une capsule, désigner cette capsule comme révision parente. Par défaut True.

— **workspace_ref** – IRI absolue où l'espace de travail sera publié : la capsule référence alors l'espace de travail au lieu de l'inclure.
— **workspace_filename** – Nom du fichier de l'espace de travail publié à workspace_ref (par défaut : dernier segment de l'IRI, à côté de la capsule).
— **attachments** – Fichiers légers déclarés dans la capsule (16 Mio au plus chacun), jamais ouverts ni exécutés.

Renvoie
Empreinte du manifeste de la capsule.

get_current_object_uuid() → str | None

Renvoie l'UUID de l'objet courant dans le panneau courant.

Renvoie
UUID de l'objet courant, ou None si aucun objet n'est courant.

get_current_panel() → str

Renvoie le nom du panneau courant.

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]]]

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

Renvoie
titres des groupes, listes des UUID et titres des objets internes
Type renvoyé
Tuple

get_method_list() → list[str]

Renvoie la liste des méthodes disponibles.

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

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

Paramètres
— **nb_id_title** – Numéro de l'objet, identifiant de l'objet, ou titre de l'objet. Par défaut None (objet courant).
— **panel** – Nom du panneau. Par défaut None (panneau courant).
Renvoie
Objet
Lève
KeyError – si l'objet n'est pas trouvé

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

Obtenir les formes des éléments graphiques associées à l'objet (signal/image).

Paramètres
— **nb_id_title** – Numéro de l'objet, identifiant de l'objet, ou titre de l'objet. Par défaut None (objet courant).
— **panel** – Nom du panneau. Par défaut None (panneau courant).
Renvoie
Liste des formes des éléments graphiques

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

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

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

Renvoie
Liste des titres des objets

Lève
ValueError – si le panneau n'est pas trouvé

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

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

Paramètres
— **panel** – nom du panneau (valeurs valides : « signal », « image »). Si None, le panneau courant est utilisé.
— **group** – Numéro du groupe, identifiant du groupe, ou titre du groupe. Par défaut None (tous les groupes).

Renvoie
Liste des UUID des objets

Lève
ValueError – si le panneau n'est pas trouvé

classmethod get_public_methods() → list[str]

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

Renvoie
Liste des méthodes publiques

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

Renvoie les UUID des objets sélectionnés.

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

Renvoie
Liste des UUID des objets sélectionnés.

get_version() → str

Renvoie la version publique de DataLab.

Renvoie
Version de DataLab

get_webapi_status() → dict[str, str | int | bool]

Obtenir l'état actuel du serveur WebAPI.

Renvoie
Dictionnaire avec les informations d'état incluant "running", "url" et "port".

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

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

Paramètres
— **filename** – Nom du fichier HDF5
— **reset_all** – Réinitialiser toutes les données de l'application. Par défaut None.

import_macro_from_file(filename: str) → None

Importer une macro depuis un fichier

Paramètres
filename – Nom du fichier.

is_connected() → bool

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

load_from_directory(path: str, param: DataSet | None = None) → None

Ouvrir des objets depuis un répertoire dans le panneau courant (signaux/images).

Paramètres
- **path** – chemin du répertoire
- **param** – Paramètres d'import optionnels propres au format, appliqués à chaque fichier qui en a besoin, sans afficher de boîte de dialogue (par exemple `sigima.io.RawImageImportParam` pour les images RAW). Par défaut None.

load_from_files(filenames: list[str], param: DataSet | None = None) → None

Ouvrir des objets depuis des fichiers dans le panneau courant (signaux/images).

Paramètres
- **filenames** – liste des noms de fichiers
- **param** – Paramètres d'import optionnels propres au format, appliqués à chaque fichier qui en a besoin, sans afficher de boîte de dialogue (par exemple `sigima.io.RawImageImportParam` pour les images RAW). Par défaut None.

load_h5_workspace(h5files: list[str], reset_all: bool = False) → None

Charger des fichiers d'espace de travail HDF5 natifs DataLab sans aucun élément d'interface graphique.

Cette méthode peut être appelée en toute sécurité depuis des scripts (par exemple, console interne, macros) car elle ne crée aucun widget Qt, boîte de dialogue ou barre de progression.

> **Avertissement**
>
> Cette méthode ne prend en charge que les fichiers HDF5 natifs DataLab. Pour importer des fichiers HDF5 arbitraires (non natifs), utilisez plutôt *open_h5_files()* ou *import_h5_file()*.

Paramètres
- **h5files** – Liste des noms de fichiers HDF5 natifs DataLab
- **reset_all** – Réinitialiser toutes les données de l'application avant l'importation. Par défaut False.

Lève
- **ValueError** – Si un fichier n'est pas un fichier HDF5 natif DataLab valide

open_capsule(filename: str, workspace: str | None = None) → None

Ouvrir une capsule comme espace de travail, en remplaçant l'espace de travail actuel.

Paramètres
- **filename** – Nom du fichier de la capsule (`.dlcapsule`).
- **workspace** – Fichier de l'espace de travail d'une capsule qui référence son espace de travail au lieu de l'inclure.

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

Ouvrir un fichier HDF5 DataLab ou importer depuis tout autre fichier HDF5.

Paramètres
- **h5files** – Liste des fichiers HDF5 à ouvrir. Par défaut None.
- **import_all** – Importer tous les objets depuis les fichiers HDF5. Par défaut None.
- **reset_all** – Réinitialiser toutes les données de l'application. Par défaut None.

raise_window() → None

Mettre la fenêtre DataLab au premier plan

remove_object(force: bool = False) → None

Supprimer l'objet courant du panneau courant.

Paramètres

force – si True, supprimer l'objet sans confirmation. Par défaut False.

reset_all() → None

Réinitialiser toutes les données de l'application

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

Exécuter une macro.

Paramètres

number_or_title – Numéro de la macro, ou titre de la macro. Par défaut None (macro courante).

Lève

ValueError – si la macro n'est pas trouvée

save_h5_workspace(filename: str) → None

Enregistrer l'espace de travail courant dans un fichier HDF5 natif DataLab sans éléments d'interface graphique.

Cette méthode peut être appelée en toute sécurité depuis des scripts (par exemple, console interne, macros) car elle ne crée aucun widget Qt, boîte de dialogue ou barre de progression.

Paramètres

filename – Nom du fichier HDF5 dans lequel enregistrer

Lève

IOError – Si le fichier ne peut pas être enregistré

save_to_h5_file(filename: str) → None

Enregistrer dans un fichier HDF5 DataLab.

Paramètres

filename – Nom du fichier HDF5

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

Sélectionner des groupes dans le panneau courant.

Paramètres

— **selection** – Liste des numéros de groupes (1 à N), ou liste des UUID de groupes, ou None pour sélectionner tous les groupes. Par défaut None.

— **panel** – nom du panneau (valeurs valides : « signal », « image »). Si None, le panneau courant est utilisé. Par défaut None.

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

Sélectionner des objets dans le panneau courant.

Paramètres

— **selection** – Liste des numéros d'objets (1 à N) ou UUID à sélectionner

— **panel** – nom du panneau (valeurs valides : « signal », « image »). Si None, le panneau courant est utilisé. Par défaut None.

set_current_panel(panel: str) → None

Basculer vers le panneau.

Paramètres

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

set_object(obj: SignalObj | ImageObj) → None

Définir les données d'un objet dans DataLab.

Mettre à jour un objet existant dans DataLab avec les nouvelles données de `obj`. L'objet est identifié par son UUID (qui est porté par `obj` depuis un appel précédent à *get_object()*).

Paramètres
obj – Objet signal ou image (doit avoir le même UUID qu'un objet existant dans DataLab)

Lève
KeyError – si aucun objet avec un UUID correspondant n'est trouvé

set_port(port: str | None = None) → None

Définir le port XML-RPC auquel se connecter.

Paramètres
port – Port XML-RPC auquel se connecter. Si None, le port est automatiquement récupéré depuis la configuration de DataLab.

start_webapi_server(host: str = '127.0.0.1', port: int = 8080) → dict[str, str | int]

Démarrer le serveur WebAPI.

Paramètres
— **host** – Adresse hôte à laquelle se lier. Par défaut « 127.0.0.1 ».
— **port** – Numéro de port. Par défaut 8080.

Renvoie
Dictionnaire avec les informations du serveur incluant "url" et "token".

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

Arrêter la macro.

Paramètres
number_or_title – Numéro de la macro, ou titre de la macro. Par défaut None (macro courante).

Lève
ValueError – si la macro n'est pas trouvée

stop_webapi_server() → bool

Arrêter le serveur WebAPI.

Renvoie
True si le serveur a été arrêté, False s'il n'était pas en cours d'exécution.

toggle_auto_refresh(state: bool) → None

Basculer l'état du rafraîchissement automatique.

Paramètres
state – État du rafraîchissement automatique

toggle_show_titles(state: bool) → None

Basculer l'état d'affichage des titres.

Paramètres
state – État d'affichage des titres

Proxy local

Le proxy local est utilisé lorsque DataLab est lancé dans le même processus que le proxy. Dans ce cas, le proxy est directement connecté à l'instance de la fenêtre principale de DataLab. L'utilisation typique est l'écriture de scripts de haut niveau.

class datalab.control.proxy.**LocalProxy**(datalab: DLMainWindow | None = None, input_source: Literal['local', 'plugin'] = 'local')

Classe proxy locale de DataLab.

Cette classe donne accès aux fonctionnalités de DataLab à partir d'une classe proxy. Il s'agit de la version locale du proxy, utilisée lorsque DataLab est lancé dans le même processus que le proxy.

Paramètres
- **datalab** (DLMainWindow) – Instance de DLMainWindow.
- **input_source** – Origine des objets ajoutés par l'intermédiaire de ce proxy.

add_signal(title: str, xdata: np.ndarray, ydata: np.ndarray, xunit: str = '', yunit: str = '', xlabel: str = '', ylabel: str = '', group_id: str = '', set_current: bool = True, new_session_behavior: SessionBehavior | None = None) → bool

Ajouter des données de signal à DataLab.

Paramètres
- **title** – Titre du signal
- **xdata** – Données X
- **ydata** – Données Y
- **xunit** – Unité X. Par défaut « «
- **yunit** – Unité Y. Par défaut « «
- **xlabel** – Libellé X. Par défaut « «
- **ylabel** – Libellé Y. Par défaut « «
- **group_id** – Identifiant du groupe dans lequel ajouter le signal. Par défaut « «
- **set_current** – si True, définit le signal ajouté comme courant
- **new_session_behavior** – Politique facultative de création des sessions d'historique

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

Lève
- **ValueError** – Type de données xdata invalide
- **ValueError** – Type de données ydata invalide

add_image(title: str, data: np.ndarray, xunit: str = '', yunit: str = '', zunit: str = '', xlabel: str = '', ylabel: str = '', zlabel: str = '', group_id: str = '', set_current: bool = True, new_session_behavior: SessionBehavior | None = None) → bool

Ajouter des données d'image à DataLab.

Paramètres
- **title** – Titre de l'image
- **data** – Données de l'image
- **xunit** – Unité X. Par défaut « «
- **yunit** – Unité Y. Par défaut « «
- **zunit** – Unité Z. Par défaut « «
- **xlabel** – Libellé X. Par défaut « «
- **ylabel** – Libellé Y. Par défaut « «
- **zlabel** – Libellé Z. Par défaut « «
- **group_id** – Identifiant du groupe dans lequel ajouter l'image. Par défaut « «
- **set_current** – si True, définit l'image ajoutée comme courante
- **new_session_behavior** – Politique facultative de création des sessions d'historique

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

Lève
ValueError – Type de données invalide

add_object(obj: SignalObj | ImageObj, group_id: str = '', set_current: bool = True, new_session_behavior: SessionBehavior | None = None) → bool

Ajouter un objet à DataLab.

Paramètres
- **obj** – Objet signal ou image
- **group_id** – Identifiant du groupe dans lequel ajouter l'objet. Par défaut « «
- **set_current** – si True, définit l'objet ajouté comme courant
- **new_session_behavior** – Politique facultative de création des sessions d'historique

Renvoie
True si l'objet a été ajouté avec succès, False sinon

multiload_session(panel: Literal['signal', 'image'], new_session_behavior: SessionBehavior | None = None) → Generator[None, None, None]

Appliquer une décision de session prise au moment du premier ajout à un chargement de plusieurs objets.

Paramètres
- **panel** – Panneau de données cible (« signal » ou « image »)
- **new_session_behavior** – Politique facultative de création des sessions d'historique

Lève
- **ValueError** – Si le panneau ou le comportement de session est invalide
- **RuntimeError** – Si une autre session de chargement multiple est déjà active

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

Appeler la fonctionnalité de calcul `name`

> **Note**
>
> Ceci appelle soit la méthode `compute_<name>` du processeur (si elle existe), soit la fonctionnalité de calcul `<name>` du processeur (si elle est enregistrée, en utilisant la méthode `run_feature`). La fonction est recherchée dans tous les panneaux, en commençant par le panneau courant.

Paramètres
- **name** – Nom de la fonction de calcul
- **param** – Paramètre de la fonction de calcul. Valeur par défaut : None

Lève
ValueError – fonction inconnue

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

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

Paramètres
- **nb_id_title** – Numéro, identifiant ou titre de l'objet. Valeur par défaut : None (objet courant)
- **panel** – Nom du panneau. Valeur par défaut : None (panneau courant)

Renvoie
Objet

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

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

Obtenir les formes des éléments graphiques associées à l'objet (signal/image).

Paramètres
- **nb_id_title** – Numéro, identifiant ou titre de l'objet. Valeur par défaut : None (objet courant)
- **panel** – Nom du panneau. Valeur par défaut : None (panneau courant)

Renvoie
Liste des formes des éléments graphiques

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

Ajouter des annotations à l'objet (éléments graphiques d'annotation).

Paramètres
— **itemsTrue** – éléments graphiques d'annotation
— **refresh_plotTrue** – Actualiser le graphique. Valeur par défaut : True
— **panel** – Nom du panneau (valeurs valides : « signal », « image »). Si None, le panneau courant est utilisé

load_h5_workspace(h5files: list[str] | str, reset_all: bool = False) → None

Charger des fichiers d'espace de travail HDF5 sans afficher de boîte de dialogue de fichiers.

Cette méthode charge directement un ou plusieurs fichiers HDF5 natifs de DataLab, sans passer par la boîte de dialogue de fichiers. Elle peut être appelée en toute sécurité depuis la console interne ou dans tout contexte où les boîtes de dialogue Qt poseraient des problèmes liés aux threads.

Paramètres
— **h5files** – Chemin(s) vers le(s) fichier(s) HDF5. Peut être une chaîne contenant un seul chemin ou une liste de chemins
— **reset_all** – Si True, réinitialiser l'espace de travail avant le chargement. Valeur par défaut : False.

Lève
ValueError – Si le fichier n'est pas un fichier HDF5 natif de DataLab

save_h5_workspace(filename: str) → None

Enregistrer l'espace de travail dans un fichier HDF5 sans afficher de boîte de dialogue de fichiers.

Cette méthode enregistre directement l'espace de travail courant dans un fichier HDF5 natif de DataLab, sans passer par la boîte de dialogue de fichiers. Elle peut être appelée en toute sécurité depuis la console interne ou dans tout contexte où les boîtes de dialogue Qt poseraient des problèmes liés aux threads.

Paramètres
filename – Chemin vers le fichier HDF5 de sortie

add_group(title: str, panel: str | None = None, select: bool = False) → str

Ajouter un groupe à DataLab.

Paramètres
— **title** – Titre du groupe
— **panel** – Nom du panneau (valeurs valides : « signal », « image »). Par défaut None.
— **select** – Sélectionner le groupe après sa création. Par défaut False.

Renvoie
UUID du groupe créé

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

Ajouter un libellé avec le titre de l'objet sur le graphique associé

Paramètres
— **title** – Titre du libellé. Par défaut None. Si None, le titre est le titre de l'objet.
— **panel** – nom du panneau (valeurs valides : « signal », « image »). Si None, le panneau courant est utilisé.

call_method(method_name: str, *args, panel: str | None = None, **kwargs)

Appeler une méthode publique sur un panneau ou la fenêtre principale.

Ordre de résolution des méthodes quand panel est None : 1. Essayer la fenêtre principale (DLMainWindow) 2. Si non trouvé, essayer le panneau courant (BaseDataPanel)

Paramètres
- **method_name** – Nom de la méthode à appeler
- ***args** – Arguments positionnels à passer à la méthode
- **panel** – Nom du panneau (« signal », « image », ou None pour la détection automatique). Par défaut None.
- ****kwargs** – Arguments nommés à passer à la méthode

Renvoie
La valeur de retour de la méthode appelée

Lève
- **AttributeError** – Si la méthode n'existe pas ou n'est pas publique
- **ValueError** – Si le nom du panneau est invalide

close_application() → None

Fermer l'application DataLab

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

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

Renvoie
Gestionnaire de contexte

Exemple

```
>>> with proxy.context_no_refresh():
...     proxy.add_image("image1", data1)
...     proxy.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

Supprimer les métadonnées des objets sélectionnés

Paramètres
- **refresh_plot** – Rafraîchir le graphique. Par défaut True.
- **keep_roi** – Conserver les ROI. Par défaut False.

export_capsule(filename: str, name: str | None = None, description: str | None = None, as_revision: bool = True, workspace_ref: str | None = None, workspace_filename: str | None = None, attachments: list[str] | None = None) → str

Exporter l'espace de travail sous forme de capsule (espace de travail et sa provenance).

Paramètres
- **filename** – Nom du fichier de la capsule (`.dlcapsule`).
- **name** – Nom de la capsule.
- **description** – Description de la capsule.
- **as_revision** – Lorsque l'espace de travail a été ouvert depuis une capsule, désigner cette capsule comme révision parente. Par défaut True.
- **workspace_ref** – IRI absolue où l'espace de travail sera publié : la capsule référence alors l'espace de travail au lieu de l'inclure.
- **workspace_filename** – Nom du fichier de l'espace de travail publié à workspace_ref (par défaut : dernier segment de l'IRI, à côté de la capsule).
- **attachments** – Fichiers légers déclarés dans la capsule (16 Mio au plus chacun), jamais ouverts ni exécutés.

Renvoie
Empreinte du manifeste de la capsule.

get_current_object_uuid() → str | None

Renvoie l'UUID de l'objet courant dans le panneau courant.

Renvoie
UUID de l'objet courant, ou None si aucun objet n'est courant.

get_current_panel() → str

Renvoie le nom du panneau courant.

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]]]

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

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

Type renvoyé
Tuple

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

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

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

Renvoie
Liste des titres des objets

Lève
ValueError – si le panneau n'est pas trouvé

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

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

Paramètres
— **panel** – nom du panneau (valeurs valides : « signal », « image »). Si None, le panneau courant est utilisé.
— **group** – Numéro du groupe, identifiant du groupe, ou titre du groupe. Par défaut None (tous les groupes).

Renvoie
Liste des UUID des objets

Lève
ValueError – si le panneau n'est pas trouvé

classmethod get_public_methods() → list[str]

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

Renvoie
Liste des méthodes publiques

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

Renvoie les UUID des objets sélectionnés.

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

Renvoie
Liste des UUID des objets sélectionnés.

get_version() → str

Renvoie la version publique de DataLab.

Renvoie
Version de DataLab

get_webapi_status() → dict

Obtenir l'état du serveur de l'API Web.

Renvoie
Dictionnaire contenant les clés « running », « url » et « token ».

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

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

Paramètres
— **filename** – Nom du fichier HDF5
— **reset_all** – Réinitialiser toutes les données de l'application. Par défaut None.

import_macro_from_file(filename: str) → None

Importer une macro depuis un fichier

Paramètres
filename – Nom du fichier.

load_from_directory(path: str, param: DataSet | None = None) → None

Ouvrir des objets depuis un répertoire dans le panneau courant (signaux/images).

Paramètres
— **path** – chemin du répertoire
— **param** – Paramètres d'import optionnels propres au format, appliqués à chaque fichier qui en a besoin, sans afficher de boîte de dialogue (par exemple `sigima.io.RawImageImportParam` pour les images RAW). Par défaut None.

load_from_files(filenames: list[str], param: DataSet | None = None) → None

Ouvrir des objets depuis des fichiers dans le panneau courant (signaux/images).

Paramètres
— **filenames** – liste des noms de fichiers
— **param** – Paramètres d'import optionnels propres au format, appliqués à chaque fichier qui en a besoin, sans afficher de boîte de dialogue (par exemple `sigima.io.RawImageImportParam` pour les images RAW). Par défaut None.

open_capsule(filename: str, workspace: str | None = None) → None

Ouvrir une capsule comme espace de travail, en remplaçant l'espace de travail actuel.

Paramètres
— **filename** – Nom du fichier de la capsule (`.dlcapsule`).
— **workspace** – Fichier de l'espace de travail d'une capsule qui référence son espace de travail au lieu de l'inclure.

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

Ouvrir un fichier HDF5 DataLab ou importer depuis tout autre fichier HDF5.

Paramètres
— **h5files** – Liste des fichiers HDF5 à ouvrir. Par défaut None.
— **import_all** – Importer tous les objets depuis les fichiers HDF5. Par défaut None.
— **reset_all** – Réinitialiser toutes les données de l'application. Par défaut None.

raise_window() → None

Mettre la fenêtre DataLab au premier plan

remove_object(force: bool = False) → None

Supprimer l'objet courant du panneau courant.

Paramètres
force – si True, supprimer l'objet sans confirmation. Par défaut False.

reset_all() → None

Réinitialiser toutes les données de l'application

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

Exécuter une macro.

Paramètres
number_or_title – Numéro de la macro, ou titre de la macro. Par défaut None (macro courante).

Lève
ValueError – si la macro n'est pas trouvée

save_to_h5_file(filename: str) → None

Enregistrer dans un fichier HDF5 DataLab.

Paramètres
filename – Nom du fichier HDF5

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

Sélectionner des groupes dans le panneau courant.

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

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

Sélectionner des objets dans le panneau courant.

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

set_current_panel(panel: str) → None

Basculer vers le panneau.

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

set_object(obj: SignalObj | ImageObj) → None

Définir les données d'un objet dans DataLab.

Mettre à jour un objet existant dans DataLab avec les nouvelles données de `obj`. L'objet est identifié par son UUID (qui est porté par `obj` depuis un appel précédent à *`get_object()`*).

Paramètres
obj – Objet signal ou image (doit avoir le même UUID qu'un objet existant dans DataLab)

Lève
KeyError – si aucun objet avec un UUID correspondant n'est trouvé

start_webapi_server(host: str | None = None, port: int | None = None) → dict

Démarrer le serveur de l'API Web.

Paramètres
- **host** – Adresse hôte à laquelle se lier. Par défaut « 127.0.0.1 ».
- **port** – Numéro de port. Par défaut, un port disponible est détecté automatiquement.

Renvoie
Dictionnaire contenant les clés « url » et « token ».

Lève
RuntimeError – Si les dépendances de l'API Web ne sont pas installées ou si le serveur est déjà en cours d'exécution.

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

Arrêter la macro.

Paramètres
number_or_title – Numéro de la macro, ou titre de la macro. Par défaut None (macro courante).

Lève
ValueError – si la macro n'est pas trouvée

stop_webapi_server() → None

Arrêter le serveur de l'API Web.

toggle_auto_refresh(state: bool) → None

Basculer l'état du rafraîchissement automatique.

Paramètres
state – État du rafraîchissement automatique

toggle_show_titles(state: bool) → None

Basculer l'état d'affichage des titres.

Paramètres
state – État d'affichage des titres

Gestionnaire de contexte de proxy

Le gestionnaire de contexte de proxy permet de gérer facilement la création et la destruction du proxy. Il s'utilise comme suit :

```
with proxy_context("local") as proxy:
    proxy.add_signal(...)
```

Le type de proxy peut être « local » ou « remote ». Pour un proxy distant, le port peut être précisé sous la forme « remote:port ».

> **Note**
>
> Le gestionnaire de contexte permet d'utiliser le proxy dans différents contextes (script Python, notebook Jupyter, etc.). Il permet également de passer facilement du proxy local au proxy distant, tout en conservant le même code à l'intérieur du contexte.

datalab.control.proxy.**proxy_context**(what: str) → Generator[LocalProxy | RemoteProxy, None, None]

Gestionnaire de contexte gérant la création et la destruction d'un proxy DataLab.

Paramètres
what – Type de proxy (« local » ou « remote »). Pour un proxy distant, le port peut être précisé sous la forme « remote:port »

Produit
proxy
LocalProxy si what == « local », RemoteProxy si what == « remote » ou « remote:port »

Exemple

with proxy_context(« local ») as proxy:
proxy.add_signal(…)

Appeler les méthodes des processeurs à l'aide d'objets proxy

Tous les objets proxy donnent accès aux méthodes de calcul de DataLab exposées par les classes de processeurs :
- `datalab.gui.processor.signal.SignalProcessor`
- `datalab.gui.processor.image.ImageProcessor`

Pour exécuter une fonctionnalité de calcul associée à un processeur, vous pouvez utiliser la méthode `calc()` de l'objet proxy :

```
# 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 servent à afficher les signaux et les images et fournissent 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.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 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
- **ValueError** – if multiple objects have the same title
- **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

resolve_title_reference(reference: str) → tuple[str, SignalObj | ImageObj | ObjectGroup] | None

Resolve a title reference to `(panel_str, object or group)`.

Paramètres
reference – `<UUID8>` (object) or `g<UUID8>` (group) reference

Renvoie
The unique match across signal and image panels, or None if the reference matches nothing or is ambiguous.

render_object_title(title: str) → str

Render an object title according to user settings.

refresh_object_title_displays() → None

Refresh object titles in trees and shown plot items.

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.

get_current_object_uuid() → str | None

Return current object uuid in current panel.

Renvoie
UUID of the current object, or None if no object is current.

add_group(title: str, panel: Literal['signal', 'image'] | None = None, select: bool = False) → str

Add group to DataLab.

Paramètres
— **title** – Group title

— **panel** – Panel name. Defaults to None.
— **select** – Select the group after creation. Defaults to False.

Renvoie
UUID of the created group

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.

call_method(method_name: str, *args, panel: Literal['signal', 'image'] | None = None, **kwargs)

Call a public method on a panel or main window.

This generic method allows calling any public method that is not explicitly exposed in the proxy API. The method resolution follows this order:

1. If panel is specified: call method on that specific panel
2. If panel is None: a. Try to call method on main window (DLMainWindow) b. If not found, try to call method on current panel (BaseDataPanel)

This makes it convenient to call panel methods without specifying the panel parameter when working on the current panel.

Paramètres
— **method_name** – Name of the method to call
— ***args** – Positional arguments to pass to the method
— **panel** – Panel name (« signal », « image », or None for auto-detection). Defaults to None.
— ****kwargs** – Keyword arguments to pass to the method

Renvoie
The return value of the called method

Lève
— **AttributeError** – If the method does not exist or is not public
— **ValueError** – If the panel name is invalid

Exemples

```
>>> # Call remove_object on current panel (auto-detected)
>>> win.call_method("remove_object", force=True)
>>> # Call a signal panel method specifically
>>> win.call_method("delete_all_objects", panel="signal")
>>> # Call main window method
>>> win.call_method("get_current_panel")
```

remote_request_context() → Iterator[None]

Context in which a request of a remote client runs (XML-RPC, Web API).

Errors are propagated to the caller, which reports them to the client, instead of being shown in a message box: no user waits for it, and showing a message box crashes DataLab when it runs without display. The error is also shown in the status bar.

call_method_slot(method_name: str, args: list, panel: Literal['signal', 'image'] | None, kwargs: dict) → None

Slot to call a method from RemoteServer thread in GUI thread.

This slot receives signals from RemoteServer and executes the method in the GUI thread, avoiding thread-safety issues with Qt widgets and dialogs.

Paramètres
- **method_name** – Name of the method to call
- **args** – Positional arguments as a list
- **panel** – Panel name or None for auto-detection
- **kwargs** – Keyword arguments as a dict

run_remote_command(command: Callable[..., object], *args, return_result: bool = False) → None

Slot to run a command requested by a remote client in GUI thread.

The exception raised by the command, if any, is stored in RemoteServer to be re-raised to the client. Being remote controlled, this slot signals that DataLab is ready only once the command is done and its exception stored.

Paramètres
- **command** – Main window method running the command
- **args** – Command arguments
- **return_result** – If True, store the command result in RemoteServer, to be returned to the client (it must be XML-RPC serializable)

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.

start_webapi_server(host: str | None = None, port: int | None = None) → dict

Start the Web API server.

Paramètres
— **host** – Host address to bind to. Defaults to « 127.0.0.1 ».
— **port** – Port number. Defaults to auto-detect available port.

Renvoie
Dictionary with « url » and « token » keys.

Lève
RuntimeError – If Web API deps not installed or server already running.

stop_webapi_server() → None

Stop the Web API server.

get_webapi_status() → dict

Get Web API server status.

Renvoie
Dictionary with « running », « url », and « token » keys.

property panels: tuple[*AbstractPanel*, ...]

Return the tuple of implemented panels (signal, image, macro, history)

Renvoie
Tuple of panels

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_menu_screenshots() → None

Take menu screenshots

open_plugin_instrument(plugin: PluginBase, tool: PluginTool) → InstrumentWindow

Show the instrument window of a plugin tool, creating it if needed.

The successive windows of an instrument share its worker: a window opened again calls the instrument once the closed one has closed it.

close_plugin_instruments() → None

Close all plugin instrument windows, and wait for their instruments, including those of windows closed before, within a common time limit.

show_applications(plugin_id: str | None = None) → None

Show the Applications panel

The first time, the panel floats beside the main window; afterwards, it is shown where the user left it (docked, tabified or floating). When it is already shown with the requested application, it flashes to draw attention to it; switching to another application needs no flashing.

Paramètres
plugin_id – ID of the application plugin to show (default: the one already shown)

set_plugins_enabled(enabled: bool) → None

Apply the global third-party plugin enabled state.

reload_plugins() → None

Reload third-party plugins at runtime.

This unregisters active plugins, clears plugin actions from both panels, re-discovers plugin modules (reloading code changes from disk), re-registers enabled plugins, then recreates plugin actions and refreshes menus.

show_command_palette() → None

Show the command palette (searchable list of menu commands).

Lists every command available for the current panel by its menu path and triggers the one chosen by the user.

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: DataSet | None = None, edit: bool = True) → 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.

— **edit** – Whether to show parameter edit dialog. Defaults to True. Set to False when calling from remote/API to avoid blocking dialogs.

Lève

FeatureNotFoundError – unknown function (a `ValueError`)

has_objects() → bool

Return True if sig/ima panels have any object

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

verify_provenance_activity(activity_id: str) → dict

Recompute a recorded activity as a separate candidate and compare it.

Nothing is inserted in the workspace and the ledger is not modified.

Paramètres

activity_id – Identifier of the recorded activity.

Renvoie

Verification report (JSON-compatible).

show_capsule_dialog() → None

Show the workspace capsule dialog (export, inspect, replay, compare).

export_capsule(filename: str, name: str | None = None, description: str | None = None, as_revision: bool = True, workspace_ref: str | None = None, workspace_filename: str | None = None, attachments: list[str] | None = None) → str

Export the workspace as a capsule (workspace and its provenance).

Paramètres

— **filename** – Capsule file name (`.dlcapsule`).

— **name** – Capsule name.

— **description** – Capsule description.

— **as_revision** – When the workspace was opened from a capsule, name that capsule as the parent revision. Defaults to True.
— **workspace_ref** – Absolute IRI where the workspace will be published: the capsule then references the workspace instead of including it, and the workspace is saved apart, to workspace_filename.
— **workspace_filename** – File name of the workspace published at workspace_ref (default: last segment of the IRI, next to the capsule).
— **attachments** – Light files (protocol, notes, small tables, 16 MiB at most each) declared in the capsule, never opened or executed.

Renvoie
Manifest fingerprint of the capsule.

open_capsule(filename: str, workspace: str | None = None) → None

Open a capsule as the workspace, replacing the current one.

The capsule is checked first (archive, manifest, workspace SHA-256 and size): the workspace is left unchanged if it is invalid. A revision exported later names this capsule as its parent.

Paramètres
— **filename** – Capsule file name (`.dlcapsule`).
— **workspace** – Workspace file of a capsule that references its workspace instead of including it.

remove_object(force: bool = False) → None

Remove current object from current panel.

Paramètres
force – if True, remove object without confirmation. Defaults to False.

save_to_h5_file(filename: str | None = 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.

When the question about clearing the workspace will be asked, the files are chosen first: if a workspace file holds provenance, the question says that appending it ignores that provenance (no further warning).

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

import_dataset_from_file(filename: str, dsetname: str | None, import_all: bool | None, reset_all: bool) → None

Open a DataLab workspace file, or import a single dataset from it

Paramètres
— **filename** – Path to the HDF5 file (already validated)
— **dsetname** – Dataset name to import, or `None` to import all
— **import_all** – If `True`, import all datasets without browsing
— **reset_all** – If `True`, clear workspace before importing

import_all_from_h5_file(filename: str, reset_all: bool | None = None) → None

Import an HDF5 file through the DataLab HDF5 browser

Paramètres
- **filename** – HDF5 filename
- **reset_all** – Delete all DataLab signals/images before importing data

browse_h5_files(filenames: list[str], reset_all: bool | None = None) → None

Browse HDF5 files

Paramètres
- **filenames** – HDF5 filenames
- **reset_all** – Reset all application data before importing

load_h5_workspace(h5files: list[str], reset_all: bool = False) → None

Load native DataLab HDF5 workspace files programmatically.

This method does not create file-selection widgets or progress bars. When history recording is active and the new-session policy is `"ask"`, a history-session question may still be shown before loading.

When a file holding provenance is appended (`reset_all=False`), its provenance is not imported: no question is asked, and the notice `datalab_capsule.recorder.APPEND_NOTICE` is added to `provenance.notices` (and logged).

> **Avertissement**
>
> This method only supports native DataLab HDF5 files. For importing arbitrary HDF5 files (non-native), use the GUI menu or macros with *datalab.control.proxy.RemoteProxy*.

Paramètres
- **h5files** – List of native DataLab HDF5 filenames
- **reset_all** – Reset all application data before importing. Defaults to False.

Lève
- **ValueError** – If a file is not a valid native DataLab HDF5 file

save_h5_workspace(filename: str) → None

Save current workspace to a native DataLab HDF5 file without GUI elements.

This method can be safely called from the internal console as it does not create any Qt widgets, dialogs, or progress bars. It is designed for programmatic use when saving DataLab workspace files.

Paramètres
- **filename** – HDF5 filename to save to

Lève
- **IOError** – If file cannot be saved

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, new_session_behavior: SessionBehavior | None = None) → bool

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
- **new_session_behavior** – Optional history session creation policy

Renvoie
True if the object was added successfully, False otherwise

set_object(obj: SignalObj | ImageObj) → None

Set object data - update an existing signal or image in-place.

The existing object is identified by UUID carried by `obj` (from a previous *get_object()* call).

Paramètres
obj – object with updated data (signal or image)

Lève
- **KeyError** – if no object with matching UUID is found
- **TypeError** – if object type is unsupported

load_from_files(filenames: list[str], param: DataSet | None = None) → None

Open objects from files in current panel (signals/images)

Paramètres
- **filenames** – list of filenames
- **param** – Optional format-specific import parameters applied to every file requiring them, without showing any dialog. Defaults to None.

load_from_directory(path: str, param: DataSet | None = None) → None

Open objects from directory in current panel (signals/images).

Paramètres
- **path** – directory path
- **param** – Optional format-specific import parameters applied to every file requiring them, without showing any dialog. Defaults to None.

get_version() → str

Return DataLab public version.

Renvoie
DataLab version

add_signal(title: str, xdata: np.ndarray, ydata: np.ndarray, xunit: str = '', yunit: str = '', xlabel: str = '', ylabel: str = '', group_id: str = '', set_current: bool = True, new_session_behavior: SessionBehavior | None = None) → 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
- **new_session_behavior** – Optional history session creation policy

Renvoie
True if signal was added successfully, False otherwise

Lève
— **ValueError** – Invalid xdata dtype
— **ValueError** – Invalid ydata dtype

add_image(title: str, data: np.ndarray, xunit: str = '', yunit: str = '', zunit: str = '', xlabel: str = '', ylabel: str = '', zlabel: str = '', group_id: str = '', set_current: bool = True, *, new_session_behavior: SessionBehavior | None = None) → 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
— **new_session_behavior** – Optional history session creation policy

Renvoie
True if image was added successfully, False otherwise

Lève
ValueError – Invalid data dtype

play_demo() → None

Play demo

show_tour() → None

Show tour

show_welcome_page() → None

Show the welcome page and bring it to the front

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.**collect_metadata_keys**(objs: Sequence[TypeObj]) → list[tuple[str, str]]

Return the user-visible scalar metadata keys of objects.

Paramètres
objs – objects to inspect

Renvoie
Sorted `(key, description)` pairs, the description showing an example value; internal options, ROIs and analysis results are excluded.

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.**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 the provenance of an object in the Provenance tab.

When the workspace ledger records a production, an old record or an analysis of the object, the provenance view is shown (see `datalab.widgets.provenance.ProvenanceWidget`); otherwise, the history rebuilt from the processing metadata of the object is shown.

Paramètres
obj – Signal or Image object

Renvoie
True if a history was found and displayed, False otherwise.

refresh_history() → None

Show again the History of the current object (ledger read again).

Called when activities were recorded or the workspace changed: results of data checks and verifications shown before are dropped.

update_properties_from(obj: SignalObj | ImageObj | None = None, force_tab: Literal['creation', 'processing', 'analysis', None] | None = None) → None

Update properties panel (properties, creation, processing) from object.

Paramètres
— **obj** – Signal or Image object
— **force_tab** – Force a specific tab to be current

keeping_tab(kind: str) → Generator[None, None, None]

Keep the tab of kind current when another object is shown meanwhile.

For navigation from a tab to another object (e.g. from the provenance of a result to one of its inputs), which otherwise shows Properties.

mark_as_newly_created(obj: SignalObj | ImageObj) → None

Mark object to show Creation tab on next selection.

Paramètres
obj – Object to mark

mark_as_freshly_processed(obj: SignalObj | ImageObj) → None

Mark object to show Processing tab on next selection.

Paramètres
obj – Object to mark

mark_as_fresh_analysis(obj: SignalObj | ImageObj) → None

Mark object to show Analysis tab on next selection.

Paramètres
obj – Object to mark

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

setup_legacy_creation_tab(obj: SignalObj | ImageObj, error: str, set_current: bool = False) → bool

Setup actions for explicitly converting historical peak parameters.

setup_invalid_creation_tab(obj: SignalObj | ImageObj, error: str, set_current: bool = False) → bool

Show a read-only state for invalid creation metadata.

convert_current_creation_parameters() → None

Convert the current object's historical peak creation parameters.

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 to show and apply again a processing.

The call that produced the object is restored (see `datalab.processingcall.resolve_processing_call()`): 1-to-1, 2-to-1 and n-to-1 processing, with every input. Parameters are edited in a parameter editor; a processing without editable parameters gets an explicit Reapply button. When the processing cannot be applied again (deleted input, unavailable operation…), the reason is shown.

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

reapply_processing(obj: SignalObj | ImageObj | None = None) → None

Apply again the processing of an object without editable parameters.

Paramètres
obj – Processed object (default: the object of the Processing tab).

setup_analysis_tab(obj: SignalObj | ImageObj, set_current: bool = False) → bool

Setup the Analysis tab with parameter editor for re-running analysis.

This tab lets the user edit the parameters of a 1-to-0 analysis operation (peak detection, FWHM, segments, etc.) and re-run it in place with the modified parameters.

Paramètres
- **obj** – Signal or Image object
- **set_current** – If True, set the Analysis tab as current after creation

Renvoie
True if Analysis tab was set up, False otherwise

apply_analysis_parameters(obj: SignalObj | ImageObj | None = None, interactive: bool = True, param: DataSet | None = None) → bool

Apply analysis parameters: re-run the 1-to-0 analysis in place.

Paramètres
- **obj** – Signal or Image object to re-analyze. If None, uses the current analysis object.
- **interactive** – If True, show error messages in the UI.
- **param** – Explicit analysis parameters to apply. When None (default), fall back to the editor dataset or the stored analysis parameters.

apply_processing_parameters(obj: SignalObj | ImageObj | None = None, interactive: bool = True, param: DataSet | None = None) → ProcessingReport

Apply processing parameters by recomputing the existing object in place.

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.
- **param** – Explicit processing parameters to apply. When provided, this takes precedence and makes the call independent of the Processing tab editor state (used e.g. by programmatic recompute paths). When None (default), fall back to the editor dataset or the stored processing parameters.

Renvoie
ProcessingReport with success status, object UUID, and optional message.

apply_recomputed_object_in_place(obj: SignalObj | ImageObj, new_obj: SignalObj | ImageObj, proc_params: ProcessingParameters) → None

Apply a freshly recomputed object onto `obj` in place.

Copies scientific data, coordinates, labels and units while preserving metadata, annotations and display settings. Only processing metadata changes.

Paramètres
- **obj** – Existing object to update in place (identity preserved).
- **new_obj** – Freshly recomputed object providing title + data.
- **proc_params** – Updated processing parameters to store on `obj`.

class datalab.gui.panel.base.**H5ImportBatch**(panel: BaseDataPanel, reference_remap: dict[str, str], objects: list[TypeObj], groups: list[ObjectGroup])

Objects and UUID remapping created by one native HDF5 import.

apply_reference_remap(reference_remap: dict[str, str], title_remap: dict[str, str]) → None

Remap title and processing references in this imported batch.

Paramètres
- **reference_remap** – serialized UUID to imported UUID mapping
- **title_remap** – serialized title reference to imported title reference

classmethod finalize_imports(batches: list[H5ImportBatch], refresh_panels: bool = False) → None

Apply the combined signal/image UUID remapping to imported batches.

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 only UUIDs that conflict with existing objects (object import).

abstractmethod serialize_to_hdf5(writer: NativeH5Writer) → None

Serialize whole panel to a HDF5 file

abstractmethod deserialize_from_hdf5(reader: NativeH5Reader, reset_all: bool = False) → H5ImportBatch | None

Deserialize whole panel from a HDF5 file

Paramètres
- **reader** – HDF5 reader
- **reset_all** – If True, preserve original UUIDs (workspace reload). If False, regenerate only UUIDs that conflict with existing objects (object import).

abstractmethod create_object() → ObjItf

Create and return object

abstractmethod add_object(obj: ObjItf) → None

Add object to panel

abstractmethod remove_all_objects()

Remove all objects

class datalab.gui.panel.base.**PasteMetadataParam**(title: str | None = None, comment: str | None = None, icon: str = '', readonly: bool = False, skip_defaults: bool = False)

Paste metadata parameters

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**(objs: list[TypeObj] | None = None, extensions: list[str] | None = None, labels: list[str] | None = None)

Save to directory parameters

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**(objs: list[TypeObj] | None = None, known_keys: Sequence[tuple[str, str]] | None = None)

Add metadata parameters

Paramètres

- **objs** – objects receiving the metadata item
- **known_keys** – suggested `(key, description)` pairs

on_help_button_click(_item: ButtonItem, _value: None, parent: QWidget) → None

Help button callback.

get_known_key_choices(_item=None, _value=None)

Return the suggested metadata keys.

on_known_key_changed(_item=None, value=None) → None

Copy the selected suggestion into the metadata key.

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 | None]

Build values according to current parameters.

Renvoie

One value per object; `None` for objects left unchanged because the extraction pattern does not match.

Lève

ValueError – If the extraction pattern is invalid or does not match (when asked to report it), or if a value cannot be converted.

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

abstractmethod static get_roi_class() → Type[TypeROI]

Return ROI class

abstractmethod 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) → H5ImportBatch

Deserialize whole panel from a HDF5 file

Paramètres

- **reader** – HDF5 reader
- **reset_all** – If True, preserve original UUIDs (workspace reload). If False, regenerate only UUIDs that conflict with existing objects (object import).

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

add_objects(objects: Sequence[TypeObj], group_id: str | None = None, set_current: bool = True) → None

Add multiple objects atomically.

Paramètres

- **objects** – Objects to add, in insertion order
- **group_id** – Group ID to which the objects belong. If None or empty, the objects are added to the current group.
- **set_current** – If True, set the last added object as current

set_object(obj: TypeObj) → None

Update an existing object in-place with data from `obj`.

The existing object is identified by UUID (carried by `obj` from a previous `get_object()` call). All data attributes are copied from `obj` to the existing object while preserving internal metadata (number, group membership).

Paramètres

obj – SignalObj or ImageObj with the same UUID as an existing object.

Lève

KeyError – if no object with matching UUID is found

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, group_uuid: str | None = None) → ObjectGroup

Add group

Paramètres

— **title** – group title

— **select** – if True, select the group in the tree view. Defaults to False.

— **group_uuid** – optional group UUID. If None, a new UUID is generated.

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)

get_known_metadata_keys(objs: Sequence[TypeObj]) → list[tuple[str, str]]

Return the metadata keys suggested by the Add metadata dialog.

Paramètres

objs – selected objects

Renvoie

the keys found on the objects, then the keys expected by the methods of application plugins

Type renvoyé

`(key, description)` pairs

add_metadata(param: AddMetadataParam | None = None) → None

Add metadata item to selected object(s)

Paramètres

param – Add metadata parameters

copy_roi(roi_data=None) → None

Copy regions of interest

Paramètres

roi_data – ROI snapshot kept for legacy session replay compatibility. When `None` (interactive use), the ROI is read from the currently selected object.

paste_roi(roi_data=None) → None

Paste regions of interest

Paramètres

roi_data – ROI snapshot kept for legacy session replay compatibility. When `None` (interactive use), the clipboard populated by *copy_roi()* is used.

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)

abstractmethod 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

abstractmethod 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

edit_import_parameters(filenames: list[str], param: DataSet | None = None) → list[tuple[str, DataSet | None]] | None

Return files to import with their optional format-specific parameters.

Paramètres
- **filenames** – File names to import.
- **param** – Optional import parameters applied to every file requiring parameters, without showing any dialog.

Renvoie
File names with aligned import parameters, without the files whose parameter dialog was cancelled, or None if the whole import was cancelled.

Lève
TypeError – If `param` is given, as this panel has no import parameters.

apply_import_parameters(objs: list[SignalObj] | list[ImageObj], param: DataSet | None) → None

Apply optional import parameters to newly read objects.

Paramètres
- **objs** – Imported objects.
- **param** – Optional format-specific import parameters.

edit_export_parameters(obj: TypeObj, filename: str) → tuple[bool, ImageExportParam | None]

Edit optional object export parameters.

Paramètres
- **obj** – Object to export
- **filename** – File name

Renvoie
A tuple containing the confirmation state and optional export parameters

write_object_to_file(obj: TypeObj, filename: str, param: ImageExportParam | None = None) → None

Save object to file (signal/image).

Paramètres
- **obj** – object
- **filename** – file name
- **param** – optional image export parameters

load_from_directory(directory: str | None = None, import_filenames: list[str] | None = None, import_params: list[DataSet] | None = None, param: DataSet | None = None, skipped_filenames: list[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
- **import_filenames** – Optional file names, relative to `directory`, of the files imported with format-specific parameters. Recorded by history so that replay does not reopen import dialogs.
- **import_params** – Optional format-specific parameters aligned with `import_filenames`.
- **param** – Optional format-specific parameters applied to every file requiring import parameters (e.g. RAW images), without showing any dialog.
- **skipped_filenames** – Optional file names, relative to `directory`, of the files whose import dialog was cancelled. Recorded by history so that replay skips them without reopening their dialog.

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, import_params: list[DataSet | None] | None = None, param: DataSet | None = None) → list[TypeObj]

Open and return objects from files, optionally adding them to DataLab.

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.
- **import_params** – Optional format-specific parameters aligned with file names. When provided by history replay, import dialogs are not reopened.
- **param** – Optional format-specific parameters applied to every file requiring import parameters (e.g. RAW images), without showing any dialog. Cannot be combined with `import_params`.

Renvoie
list of new objects

save_to_files(filenames: list[str] | str | None = None, export_params: list[ImageExportParam | None] | None = None, object_uuids: list[str] | None = None) → None

Save selected objects to files (signal/image).

Paramètres
- **filenames** – File names
- **export_params** – Optional image export parameters aligned with file names
- **object_uuids** – Optional object UUIDs aligned with 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

refresh_after_reference_remap() → None

Refresh cached presentation after imported references are remapped.

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_selected() → None

Recompute/rerun selected objects or group with stored parameters.

This method handles both single objects and groups. For each object, it recomputes, on demand:
— the processing that produced it (1-to-1, 2-to-1 or n-to-1, as restored by `datalab.processingcall.resolve_processing_call()` : from the provenance ledger, else from the processing metadata), in place, and
— 1-to-0 analysis operations (statistics, measurements, detections, etc.).
Analysis results are not recomputed automatically when data, ROIs or object properties change: this manual action is the single, explicit entry point to refresh them. Objects without recomputable parameters are skipped.

recompute_1_to_1_objects(objects: list[SignalObj | ImageObj]) → tuple[set[str], bool]

Recompute in-place 1-to-1 processing operations for the given objects.

Paramètres

objects – Objects with stored 1-to-1 processing parameters

Renvoie

— Set of object UUIDs successfully recomputed
— True if operation was interrupted (progress canceled or user chose to stop after a failure), False otherwise

Type renvoyé
Tuple containing

recompute_1_to_0_objects(objects: list[SignalObj | ImageObj]) → tuple[set[str], bool]

Recompute 1-to-0 analysis operations for the given objects.

Paramètres
objects – Objects with stored 1-to-0 analysis parameters

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: 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

datalab.gui.panel.signal.**LEGACY_AUTOMATIC_LINE_WIDTH = 1.0**

Line width stored by DataLab 1.3 or earlier for curves drawn with the default line width, or with a line width clamped for curves with many points

class datalab.gui.panel.signal.**SignalAnnotatedSegmentTool**(manager, setup_shape_cb: Callable | None = None, handle_final_shape_cb: Callable | None = None, shape_style: tuple[str, str] | None=None, toolbar_id: str = <class 'plotpy.tools.base.DefaultToolbarID'>, title: str | None = None, icon: str | None = None, tip: str | None = None, switch_to_default_tool: bool | None = None)

Annotated segment tool for signals

The length and the angle of a segment combine its X and Y extents, whose units generally differ on a signal: unlike on images, they are not displayed.

create_shape() → tuple[AnnotatedSegment, int, int]

Create an annotated segment without computations

Renvoie

Annotated segment and the indices of its handles

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

alias de 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

deserialize_object_from_hdf5(reader: NativeH5Reader, name: str, reset_all: bool = False) → SignalObj

Deserialize object from a HDF5 file

DataLab 1.3 and earlier stored automatic curve styles in signal metadata: when loading a file saved by these versions, the colors and line styles of their automatic style cycle are removed, so that curves are given the current automatic styles. Other stored styles are kept. These versions also stored the line width of every curve: a line width of 1.0 (the default width, also used for curves with many points) is removed, so that curves are given the configured line width; other widths were chosen by the user and are kept.

Paramètres

- **reader** – HDF5 reader
- **name** – Object name in HDF5 file
- **reset_all** – If True, preserve original UUIDs (workspace reload). If False, regenerate only UUIDs that conflict with existing objects (object import).

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 the curve styles of the selected signals

Colors and line styles chosen by the user are removed from the metadata of the selected signals, whose curves are given automatic styles again.

Image panel

datalab.gui.panel.image.**get_image_export_dtype_choices**(param: ImageExportGUIParam, _item: ChoiceItem, _value: str) → list[tuple[str, str, None]]

Return target dtype choices supported by the selected image format.

class datalab.gui.panel.image.**ImageExportGUIParam**(title: str | None = None, comment: str | None = None, icon: str = '', readonly: bool = False, skip_defaults: bool = False)

Image export parameters restricted to the selected file format.

is_item_active(name: str) → bool

Return True if the item named `name` is taken into account.

get_format_options() → dict[str, object]

Return writer options exposed by this format parameter class.

Inactive options are replaced by their default value, as they are not taken into account for the current values of the other options.

to_export_param() → ImageExportParam

Return backend export parameters from the current GUI values.

Inactive parameters (e.g. gamma without normalization) are reset to their default value, as they are not taken into account.

Renvoie
Backend image export parameters

Lève
ValueError – If a format option value is invalid

datalab.gui.panel.image.**get_image_export_choice_label**(value: object) → str

Return a translated human-readable label for an option choice.

datalab.gui.panel.image.**serialize_image_export_sequence**(value: object, spec: ImageExportOptionSpec) → tuple[int, int] | tuple[float, float] | list[float] | None

Serialize a pair or list option from its editable GUI representation.

datalab.gui.panel.image.**create_image_export_option_item**(spec: ImageExportOptionSpec, extension: str = '') → DataItem

Create a guidata item for an image export capability option.

Paramètres
— **spec** – Option specification
— **extension** – Canonical format extension, used for format-specific help
Renvoie
Data item editing the option

datalab.gui.panel.image.**get_image_export_param_class**(filename: str) → type[ImageExportGUIParam]

Return the cached GUI parameter class for an export format.

datalab.gui.panel.image.**is_raw_image_file**(filename: str) → bool

Return True if the file is a headerless RAW image requiring parameters.

Paramètres
filename – File name.
Renvoie
True if the file has a `.raw` extension.

datalab.gui.panel.image.**create_image_export_gui_param**(obj: ImageObj, filename: str) → ImageExportGUIParam

Create format-aware image export parameters with practical defaults.

Paramètres
— **obj** – Source image
— **filename** – Destination file name
Renvoie
Image export GUI parameters

datalab.gui.panel.image.**compute_image_export_preview**(source: ImageObj, param: ImageExportParam, filename: str) → ImageObj

Return the image as it will be read back from the exported file.

This function runs in the preview process, so its arguments must be picklable.

Paramètres
— **source** – Image to export
— **param** – Image export parameters
— **filename** – Destination file name
Renvoie
Copy of the source image holding the exported data, displayed in gray levels

datalab.gui.panel.image.**format_image_export_error**(message: str) → str

Return only the error message of an expected export validation error.

Validation errors raised by Sigima (`ValueError`/`TypeError`) explain the invalid configuration: the traceback is only kept for unexpected errors.

Paramètres
message – Error message or formatted traceback

Renvoie
Message to display

class datalab.gui.panel.image.**ImageExportDialog**(parent: QWidget | None, obj: ImageObj, filename: str, param: ImageExportGUIParam, controller_factory: Callable = <class 'datalab.gui.processor.preview.PreviewController'>)

Format-aware image export dialog with a live preview of the exported data.

static to_export_param(guiparam: ImageExportGUIParam) → ImageExportParam

Return backend export parameters from GUI parameters.

Paramètres
guiparam – Format-specific GUI parameters

Renvoie
Backend image export parameters

Lève
ValueError – If a format option value is invalid

get_export_param() → ImageExportParam

Return backend parameters from the current GUI values.

accept() → None

Validate the export configuration before accepting.

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

alias de `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

edit_import_parameters(filenames: list[str], param: DataSet | None = None) → list[tuple[str, DataSet | None]] | None

Return files to import with their RAW image import parameters.

When several RAW files are imported, the user first chooses whether the same parameters apply to all of them (a single dialog) or not (one dialog per file).

Paramètres
— **filenames** – File names to import.

— **param** – Optional RAW image import parameters applied to every RAW file without showing any dialog (e.g. remote control).

Renvoie
File names with aligned import parameters, without the RAW files whose dialog was cancelled, or None if the whole import was cancelled.

Lève
TypeError – If `param` is not a RAW image import parameter set.

edit_raw_import_parameters(raw_files: list[str]) → dict[str, RawImageImportParam] | None

Edit RAW image import parameters, once for all files or per file.

Paramètres
raw_files – RAW file names.

Renvoie
Parameters of the confirmed files, or None if the import was cancelled.

apply_import_parameters(objs: list[ImageObj], param: DataSet | None) → None

Apply RAW display semantics without changing imported values.

Paramètres
— **objs** – Imported image objects.
— **param** – Optional format-specific import parameters.

edit_export_parameters(obj: ImageObj, filename: str) → tuple[bool, ImageExportParam | None]

Edit image export parameters for the selected file format.

Paramètres
— **obj** – Image to export
— **filename** – Destination file name

Renvoie
A tuple containing the confirmation state and export parameters

write_object_to_file(obj: ImageObj, filename: str, param: ImageExportParam | None = None) → None

Write an image with optional format-aware export parameters.

Paramètres
— **obj** – Image to export
— **filename** – Destination file name
— **param** – Optional image export parameters

register_contrast_editor(obj: ImageObj, editor: ImageROIEditor) → None

Register an image ROI editor for contrast synchronization.

apply_shared_contrast(obj: ImageObj, zmin: float, zmax: float, source: ImageROIEditor | None = None) → None

Apply a contrast change coming from another view.

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

closeEvent(event) → None

Persist splitter state on close.

showEvent(event) → None

Trigger the macro-recovery prompt on first show.

Deferred to the first showEvent (and re-scheduled via singleShot) so the splash screen has been finished before the QMessageBox appears — otherwise the dialog can pop up behind the splash.

add_macro() → Macro

Add macro, optionally with name

Renvoie
Macro object

add_macro_with_code(title: str, code: str, source: str | None = None) → Macro

Add a macro with a predefined title and code, without prompting.

This helper is used by the AI assistant to programmatically inject a generated macro into the panel.

Paramètres
— **title** – Macro title (displayed in the tab).
— **code** – Python source code of the macro.
— **source** – Optional origin marker (e.g. `"import"`, `"ai"`, `"template:simple_macro"`) recorded in the recent cache.

Renvoie
The newly created macro.

add_blank_macro() → Macro

Add a new empty macro and return it.

add_macro_from_template(template: MacroTemplate) → Macro

Add a new macro initialised from template.

Paramètres
template – Macro template description.

Renvoie
The newly created macro.

duplicate_macro(number_or_title: int | str | None = None) → Macro | None

Duplicate the given (or current) macro.

Paramètres
number_or_title – Number of the macro (starting at 1) or title. Defaults to None (current macro).

Renvoie
The duplicated macro, or None if no source macro was found.

show_recent_dialog() → None

Open a dialog listing recent macros and allow reopening one.

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)

update_untitled_counter() → None
Update the untitled counter based on existing macro titles

This scans all macro titles to find the highest « macro_XX » number and updates the global counter to prevent duplicate names.

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) → 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: 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: 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.

clear_plugin_actions() → None

Clear plugin actions and submenus

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: 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) → 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) → 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: 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: 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.

clear_plugin_actions() → None

Clear plugin actions and submenus

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: 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) → 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 model

The *datalab.objectmodel* module defines the object data model used by the GUI to store signals, images and groups.

The model is based on a hierarchical tree of objects, with two levels:
- The top level is a list of groups (ObjectGroup instances)
- The second level is a list of objects (SignalObj or ImageObj instances)

The model is implemented by the ObjectModel class.

Object group

The ObjectGroup class represents a group of objects. It is a container for SignalObj and ImageObj instances.

class datalab.objectmodel.**ObjectGroup**(title: str, model: ObjectModel, prefix: str, group_uuid: str | None = None)

Represents a DataLab object group

Paramètres
- **title** – group title
- **model** – object model
- **prefix** – prefix for short ID (« gs » for signal groups, « gi » for image groups)
- **group_uuid** – optional group UUID. If None, a new UUID is generated.

property title: str

Return group title

append(obj: SignalObj | ImageObj) → None

Append object to group

insert(index: int, obj: SignalObj | ImageObj) → None

Insert object at index

remove(obj: SignalObj | ImageObj) → None

Remove object from group

clear() → None

Clear group

get_objects() → list[SignalObj | ImageObj]

Return objects in group

get_object_ids() → list[str]

Return object ids in group

Object model

The ObjectModel class is a container for ObjectGroup instances, as well as a container for SignalObj and ImageObj instances.

class datalab.objectmodel.**ObjectModel**(group_prefix: str)

Represents a DataLab object model (groups of signals/images)

reset_short_ids() → None

Reset short IDs (used for object numbering)

This method is called when an object was removed from a group.

has_uuid(uuid: str) → bool

Check if an object or group with the given UUID exists in the model.

Paramètres
uuid – UUID string to check

Renvoie
True if an object or group with this UUID exists, False otherwise

clear() → None

Clear model

get_all_objects(flatten: bool = True) → list[SignalObj | ImageObj] | list[list[SignalObj | ImageObj]]

Return all objects, in order of appearance in groups

Paramètres
flatten – if True, return a flat list of objects, otherwise return a list of lists (one list per group)

Renvoie
List of objects in order of appearance in groups.

get_object_or_group(uuid: str) → SignalObj | ImageObj | ObjectGroup

Return object or group with uuid

find_by_title_reference(reference: str) → list[SignalObj | ImageObj | ObjectGroup]

Return the objects or groups matching a title reference.

Paramètres
reference – <UUID8> (object) or g<UUID8> (group) reference

Renvoie
Matching objects or groups (more than one only on prefix collision).

get_group(uuid: str) → ObjectGroup

Return group with uuid

get_number(obj_or_group: SignalObj | ImageObj | ObjectGroup) → int

Return number of object or group

get_group_from_number(number: int) → ObjectGroup

Return group from its number.

Paramètres
number – group number (starts with 1)

Renvoie
Group

Lève
IndexError – if group with number not found

get_group_from_title(title: str) → ObjectGroup

Return group from its title.

Paramètres
title – group title

Renvoie
Group

Lève
— **KeyError** – if group with title not found
— **ValueError** – if multiple groups have the same title

get_group_from_object(obj: SignalObj | ImageObj) → ObjectGroup

Return group containing object

Paramètres
obj – object to find group for

Renvoie
Group

Lève
KeyError – if object not found in any group

get_groups(uuids: list[str] | None = None) → list[ObjectGroup]

Return groups

add_group(title: str, group_uuid: str | None = None) → ObjectGroup

Add group to model

Paramètres
— **title** – group title
— **group_uuid** – optional group UUID. If None, a new UUID is generated.

Renvoie
Created group object

get_object_group_id(obj: SignalObj | ImageObj) → str | None

Return group id of object

Paramètres
obj – object to get group id from

Renvoie
group id or None if object is not in any group

get_group_object_ids(group_id: str) → list[str]

Return object ids in group

remove_group(group: ObjectGroup) → None

Remove group from model

add_object(obj: SignalObj | ImageObj, group_id: str) → None

Add object to model

add_objects(objects: Sequence[SignalObj | ImageObj], group_id: str) → bool

Add objects to the model atomically.

Paramètres
— **objects** – Objects to add, in insertion order
— **group_id** – UUID of the target group

Renvoie
True if the short IDs of objects or groups already in the model changed; False when the objects were added after every existing object (the usual case), which leaves them unchanged.

Lève
— **KeyError** – If the target group does not exist
— **ValueError** – If an object UUID is duplicated or already exists

remove_object(obj: SignalObj | ImageObj) → None

Remove object from model

get_object_from_number(number: int) → SignalObj | ImageObj

Return object from its number.

Paramètres
number – object number (unique in model)
Renvoie
Object
Lève
IndexError – if object with number not found

get_objects(uuids: list[str]) → list[SignalObj | ImageObj]

Return objects with uuids

get_object_ids(flatten: bool = True) → list[str] | list[list[str]]

Return object ids, in order of appearance in groups

Paramètres
flatten – if True, return a flat list of object ids, otherwise return a list of lists (one list per group)
Renvoie
List of object ids in order of appearance in groups.

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(flatten: bool = True) → list[str] | list[list[str]]

Return object titles, in order of appearance in groups

Paramètres
flatten – if True, return a flat list of object titles, otherwise return a list of lists (one list per group)
Renvoie
List of object titles in order of appearance in groups.

get_object_from_title(title: str) → SignalObj | ImageObj

Return object with title.

Paramètres
title – object title
Renvoie
object with title
Lève
— **KeyError** – if object with title not found
— **ValueError** – if multiple objects have the same title

get_title_reference_index() → dict[str, list[SignalObj | ImageObj | ObjectGroup]]

Return a dictionary mapping title references to groups and objects.

Each value is the list *find_by_title_reference()* returns for the reference. The index must be built again after objects or groups are added or removed.

reorder_groups(group_ids: list[str]) → None

Reorder groups.

Paramètres
group_ids – list of group uuids

reorder_objects(obj_ids: dict[str, list[str]]) → None

Reorder objects in groups.

Paramètres
obj_ids – dict of group uuids and list of object uuids

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: QTreeWidgetItem | None = None) → Iterator[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

resolve_title_reference(reference: str) → tuple[str, SignalObj | ImageObj | ObjectGroup] | None

Resolve a title reference to `(panel_str, obj)`.

Default implementation only looks up the tree's own model and returns an empty `panel_str`. Subclasses with access to several panels should override this method.

get_title_reference_display(reference: str) → str | None

Return the configured display text for a resolvable title reference.

get_source_title(reference: str) → str | None

Return the title of the object or group a reference resolves to.

build_title_reference_tooltip(text: str) → str

Return an HTML tooltip fragment listing the source objects referenced in `text`, or an empty string when no such reference is found.

handle_title_reference_click(reference: str) → None

Handle a click on a title-reference hyperlink. Default implementation selects the matching object within the tree's own model.

title_reference_at(pos: QPoint) → str | None

Return the title reference under viewport position `pos`, or `None` when the cursor is not over an anchor.

mousePressEvent(event: QMouseEvent) → None

Reimplement Qt method to handle short-ID hyperlink clicks.

mouseMoveEvent(event: QMouseEvent) → None

Reimplement Qt method to update the cursor over short-ID anchors.

hovered_index() → QModelIndex

Return the index of the item under the mouse cursor (invalid if none)

viewportEvent(event: QEvent) → bool

Reimplement Qt method to track the hovered item

changeEvent(event: QEvent) → None

Reimplement Qt method to keep hover events after a style change (some styles disable them when unpolishing item views)

populate_tree() → None

Populate tree with objects

update_tree() → None

Update tree

Note: If an item is not found in the tree, the tree is repopulated to ensure consistency between the tree and the model. This can happen in rare cases when objects are added to the model but the tree was not properly updated.

resize_title_column(items: Sequence[QTreeWidgetItem] | None = None) → None

Fit the title column to the titles.

Paramètres

items – Items just added to a tree whose column was already fitted to the other items: the column is only widened if one of them needs it, without measuring every title again. All items when None.

add_group_item(group: ObjectGroup, resize: bool = True) → None

Add group item

Paramètres

— **group** – Group to add, with its objects
— **resize** – If True, fit the title column to the new items

add_object_item(obj: SignalObj | ImageObj, group_id: str, set_current: bool = True) → None

Add item

add_object_items(objects: Sequence[SignalObj | ImageObj], group_id: str, set_current: bool = True) → None

Add multiple object items with a single tree update cycle.

Paramètres

— **objects** – Objects to add, in insertion order
— **group_id** – UUID of the parent group
— **set_current** – If True, set the last object as current

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

A delete button is shown at the right of the hovered item (signal, image or group), unless disabled (see *set_delete_button_enabled()*): clicking it emits SIG_DELETE_REQUESTED with the item UUID.

set_delete_button_enabled(enabled: bool) → None

Show or hide the delete button of the hovered item

Paramètres
enabled – True to show the button

is_delete_button_enabled() → bool

Return True if the delete button of the hovered item is shown

delete_button_rect(index: QModelIndex) → QRect | None

Return the rectangle of the delete button of an item

The button is shown on the hovered item only, at the right edge of the visible part of the viewport (the title column may be wider than the viewport).

Paramètres
index – model index of the item

Renvoie
Button rectangle in viewport coordinates, or None if the item shows no delete button.

get_delete_targets(uuid: str) → list[str]

Return the UUIDs of the items deleted by the delete button of an item

Paramètres
uuid – UUID of the item whose delete button is clicked

Renvoie
UUIDs of the selected objects or groups if the item is part of the selection, else the item UUID only.

delete_button_tooltip(index: QModelIndex) → str

Return the tooltip of the delete button of an item

Paramètres
index – model index of the item

Renvoie
Tooltip text

drawRow(painter: QPainter, option: QStyleOptionViewItem, index: QModelIndex) → None

Reimplement Qt method to draw the delete button of the hovered item

viewportEvent(event: QEvent) → bool

Reimplement Qt method to show the tooltip of the delete button and follow the mouse cursor over it

mousePressEvent(event: QMouseEvent) → None

Reimplement Qt method to press the delete button, without selecting or dragging the item

mouseMoveEvent(event: QMouseEvent) → None

Reimplement Qt method to follow the mouse cursor over the delete button

mouseReleaseEvent(event: QMouseEvent) → None

Reimplement Qt method to click the delete button

mouseDoubleClickEvent(event: QMouseEvent) → None

Reimplement Qt method to ignore double-clicks on the delete button (which would open the item in a separate view)

resolve_title_reference(reference: str) → tuple[str, SignalObj | ImageObj | ObjectGroup] | None

Resolve a title reference across the signal and image panels of the main window, so titles can reference objects living in either panel.

handle_title_reference_click(reference: str) → None

Select the referenced object, switching active panel if needed.

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

get_current_object_uuid() → str | None

Return current object uuid, or None if current item is a group or if no item is current

set_current_object(obj: SignalObj | ImageObj) → None

Set current object

select_uuids(uuids: Sequence[str], notify: bool = True) → None

Select objects or groups atomically, the last one becoming current

Paramètres

- **uuids** – UUIDs of the objects or groups to select
- **notify** – if True, publish the complete final state once (see *item_selection_changed()*)

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

update_automatic_styles() → None

Give the visible curves the automatic style of their position

Visible curves are numbered in the order of their signals in the panel. Each curve whose color and line style were not chosen by the user is given the automatic style of its position (see `CurveStyles.style_at()`), so that the first curves shown are always given the first styles.

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_existing_items() → None

Update all existing plot items from their objects and current settings

Unlike *refresh_plot()*, the visibility of the plot items and the shape items are left unchanged. This applies a change of display settings (e.g. signal auto downsampling or line width) to the plot items of all objects, and not only to those of the selected objects.

update_item_on_plot(oid: str, just_show: bool = False) → None

Update plot item.

Paramètres
— **oid** – object uuid
— **just_show** – if True, only show the item (do not update it). Defaults to False.

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_automatic_styles() → None

Update the automatic styles of the visible plot items

This base implementation does nothing.

update_existing_items() → None

Update all existing plot items from their objects and current settings

Unlike *refresh_plot()*, the visibility of the plot items and the shape items are left unchanged. This applies a change of display settings (e.g. signal auto downsampling or line width) to the plot items of all objects, and not only to those of the selected objects.

update_item_on_plot(oid: str, just_show: bool = False) → None

Update plot item.

Paramètres
— **oid** – object uuid
— **just_show** – if True, only show the item (do not update it). Defaults to False.

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, source_panel: BaseDataPanel | 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, source_panel: BaseDataPanel | 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

apply_shared_contrast(zmin: float, zmax: float) → None

Apply a contrast change coming from the image panel.

plot_lut_changed(plot: BasePlot) → None

Synchronize ROI editor contrast changes with the source image panel.

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

Extends the SigimaX plot widget with the « Process signal » action, which sends a profile to the DataLab signal panel.

Paramètres
`plot_type` – 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

PLOTWIDGET_CLASS

alias de *DataLabPlotWidget*

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

append_informed

True when the user was already told, for the current opening, that the provenance of appended workspaces is not imported (no repeat).

cancelled_files: list[str]

Files whose appending the user cancelled (see `__confirm_append()`)

save_file(filename: str) → None

Save all signals and images from DataLab model into a HDF5 file

The file is replaced only once completely written: if saving fails, the previous file (if any) is left untouched.

open_file_headless(filename: str, reset_all: bool) → bool

Open native DataLab HDF5 file without any GUI elements.

This method can be safely called from any thread (e.g., the console thread) as it does not create any Qt widgets or dialogs. When a file holding provenance is appended (reset_all False), its provenance is not imported and the notice `datalab_capsule.recorder.APPEND_NOTICE` is added to `mainwindow.provenance.notices` (and logged).

Paramètres
— **filename** – HDF5 filename
— **reset_all** – Reset all application data before importing

Renvoie
True if file was successfully opened as a native DataLab file, False if the file format is not compatible (KeyError was raised)

Lève
ProvenanceFormatError – If the provenance block is invalid (raised before the workspace is replaced).

open_file(filename: str, import_all: bool, reset_all: bool) → None

Open HDF5 file

When the file holds provenance and is appended to the current workspace, the user is warned before anything changes (see `__confirm_append()`) and may cancel. A file that is not a DataLab workspace is imported as a generic HDF5 file.

open_workspace_file(filename: str) → None

Open a DataLab workspace file, replacing the workspace.

Unlike *open_file()*, a file that is not a DataLab workspace is never imported as a generic HDF5 file.

Lève
— **ValueError** – If the file is not a DataLab workspace.
— **ProvenanceFormatError** – If the provenance block is invalid.

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 4

Contribuer

DataLab est votre plateforme. Si vous souhaitez qu'elle s'améliore, vous pouvez contribuer au projet, que vous soyez développeur ou non.

Il existe de nombreuses façons de contribuer au projet DataLab, en fonction du temps dont vous disposez, de votre expérience avec les projets open source et de vos compétences.

4.1 Partagez vos idées et vos expériences

Outre les rapports d'anomalies classiques et les demandes de fonctionnalités, vous pouvez partager vos idées et vos expériences pour améliorer DataLab. En particulier, nous sommes très intéressés par vos commentaires sur la documentation et les tutoriels. De plus, si vous avez un cas d'utilisation que vous souhaitez partager avec la communauté, faites-le nous savoir.

Sans coder une seule ligne, vous pouvez contribuer au projet DataLab en :

- Signaler une anomalie
- Suggérer une amélioration
- Suggérer un sujet de documentation
- Suggérer un sujet de tutoriel

4.2 Partagez vos connaissances scientifiques/techniques

Vos connaissances techniques ou scientifiques sont également très précieuses pour nous, car vous pouvez contribuer directement à la documentation ou aux tutoriels. Ou, mieux encore, si vous souhaitez rédiger un tutoriel, nous serons heureux de vous aider.

Encore une fois sans coder, vous pouvez contribuer au projet DataLab en :

- Ecrire de la documentation
- Ecrire un tutoriel

4.3 Contribuer aux nouvelles fonctionnalités

Même si vous n'êtes pas développeur, vous pouvez contribuer au projet en :
- Testant de nouvelles fonctionnalités
- Ecrivant et soumettant de nouveaux plugins
- Ecivant et soumettant des macro-commandes

4.4 Développer de nouvelles fonctionnalités

Si vous êtes développeur, vous pouvez contribuer au cœur du projet en corrigeant des anomalies ou en implémentant de nouvelles fonctionnalités.

Voir la section Contribuer au code pour plus d'informations.

4.4.1 Contribuer au code

Cette page explique comment vous pouvez contribuer au code du projet.

> **Voir aussi**
>
> Les règles de codage sont présentées dans la page Règles de codage.

Forker le projet

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

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

Soumettre une pull request

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

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

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

Tâches de mainteneur

Quelques tâches de maintenance ne sont volontairement pas exécutées par la CI de release et doivent être effectuées localement sur un poste Windows par un mainteneur du projet. Elles produisent des fichiers versionnés que la CI consomme ensuite tels quels.

Rafraîchir les captures d'écran de la documentation

Les fichiers PNG sous `doc/images/` sont versionnés. Ils sont régénérés en lançant DataLab et en capturant ses dialogues / panneaux — une étape qui nécessite une véritable session de bureau (le runner CI sous Xvfb se fige au démarrage de l'interface DataLab et le rendu dépend des polices et du thème du système, ce qui le rend non-déterministe d'un runner à l'autre).

Le pipeline de capture produit des fichiers par langue (`foo.fr.png` / `foo.en.png`) grâce à la configuration `figure_language_filename` de Sphinx. Les sources RST référencent le nom sans suffixe (`foo.png`) ; chaque build de langue sélectionne automatiquement le suffixe correspondant lorsqu'il est disponible et retombe sur le fichier non suffixé sinon.

À relancer dès qu'une modification affecte le rendu de l'interface (menus, dialogues, panneaux, thèmes) :

```
scripts\update_screenshots.bat
```

ou via la tâche VS Code 🖼 Refresh doc screenshots. Validez les PNGs nouveaux ou mis à jour sous `doc/images/shots/` dans un commit dédié (par exemple `docs: refresh screenshots`). Dès qu'une paire `foo.fr.png` + `foo.en.png` existe pour une capture donnée, le fichier non suffixé `foo.png` n'est plus utilisé que comme repli et pourra être supprimé dans un commit de nettoyage ultérieur.

Régénérer les ressources graphiques

L'icône de l'application (`resources/DataLab.ico`) et les bitmaps de l'installeur WiX (`wix/dialog.bmp`, `wix/banner.bmp`) sont générés à partir de sources SVG. À relancer uniquement quand les sources SVG changent :

```
scripts\build_resources.bat
```

ou via la tâche VS Code 🎨 Generate resources.

Les images affichées par l'application (écran de démarrage, logos, filigrane des graphiques) sont générées dans `datalab/data/logo/` avec leurs variantes haute résolution `@2x`, utilisées sur les écrans haute résolution : pour ne régénérer que ces images, exécutez `scripts\build_resources.bat logo`.

4.4.2 Règles de codage

Règles de codage génériques

Nous suivons le style de codage PEP 8.

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

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

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

- 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.

4.4.3 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.

Branche de maintenance

— `release` : Représente la ligne de maintenance actuelle pour les versions correctives.
  — Créé à partir de `main` après une version stable lorsque le premier correctif est nécessaire.
  — Accepte les fusions des branches `fix/xxx` et `hotfix/xxx`.
  — Fusionné de nouveau dans `main` pour chaque tag de version corrective (par exemple, 1.0.1, 1.0.2).
  — Réinitialisé ou recréé lors du démarrage d'un nouveau cycle de version mineure/majeure.

> **Note**
>
> La branche `release` n'est pas versionnée (pas de `release/1.0.x`). Elle représente toujours « la ligne de maintenance actuelle ». Lorsqu'une nouvelle version mineure est publiée (par exemple, 1.2.0), l'ancienne branche `release` est supprimée et une nouvelle est créée à partir du nouveau tag lorsque le premier correctif pour 1.2.1 est nécessaire.

Branches de correction d'anomalies

— `fix/xxx` : Utilisé pour les corrections générales d'anomalies qui ne sont pas urgentes.
  — Créé à partir de `develop` (pour la prochaine version de fonctionnalité) ou `release` (pour la version corrective).
  — Fusionné de nouveau dans la branche d'origine une fois terminé.
  — Supprimé après la fusion.
— `hotfix/xxx` : Utilisé pour les corrections urgentes critiques pour la production.
  — Créé à partir de `release` (si elle existe) ou `main` (si aucune branche `release` n'existe encore).
  — Fusionné de nouveau dans `release` ou `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

Pour la prochaine version de fonctionnalité (cible : `develop`) :

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
```

Pour la version de maintenance actuelle (cible : `release`) :

1. Créer une branche de correction d'anomalie à partir de `release` :

```
git checkout release
git checkout -b fix/bug_description
```

2. Appliquer la correction et valider les modifications.
3. Fusionner la branche de correction de nouveau dans `release` :

```
git checkout release
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` ou `release` 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 `release` (ou `main` si aucune branche `release` n'existe) :

```
git checkout release  # or: git checkout main
git checkout -b hotfix/critical_bug
```

2. Appliquer la correction et valider les modifications.
3. Fusionner la correction de nouveau dans `release` (ou `main`) :

```
git checkout release  # or: 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.

Workflow pour les versions correctives

Dès qu'une version corrective est prête à être publiée (par exemple, 1.0.1, 1.0.2) :

1. S'assurer que la branche `release` contient toutes les corrections souhaitées.
2. Fusionner `release` dans `main` :

```
git checkout main
git merge --no-ff release
```

3. Taguer la version sur `main` :

```
git tag -a v1.0.1 -m "Release version 1.0.1"
git push origin main --tags
```

4. Conserver la branche `release` pour des correctifs supplémentaires dans la même série de versions mineures.

Workflow pour les versions mineures/majeures

Dès qu'une nouvelle version mineure ou majeure est prête à être publiée (par exemple, 1.1.0, 2.0.0) :

1. Fusionner `develop` dans `main` :

```
git checkout main
git merge --no-ff develop
```

2. Taguer la version sur `main` :

```
git tag -a v1.1.0 -m "Release version 1.1.0"
git push origin main --tags
```

3. Supprimer l'ancienne branche `release` (si elle existe) :

```
git branch -d release
git push origin --delete release
```

4. Créer une nouvelle branche `release` à partir de `main` lorsque le premier correctif pour 1.1.1 est nécessaire :

```
git checkout main
git checkout -b release
git push -u origin release
```

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 de bogues dans `release` ou `main` sont toujours sélectionnées dans `develop`.
— Différencier clairement entre `fix/xxx` (correctifs non urgents) et `hotfix/xxx` (correctifs critiques pour la production).
— Lors de la création de la branche `release`, mettre à jour les notes de version pour indiquer la version qu'elle cible (par exemple, ajouter un commentaire dans le commit de fusion : « Créer une branche de maintenance pour v1.0.x »).
— La branche `release` représente toujours la ligne de maintenance actuelle. Pour savoir quelle version elle cible, vérifiez le tag le plus récent sur `main` ou les notes de version.

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.

La branche `release` fournit une ligne de maintenance dédiée pour les versions correctives, permettant à `develop` de poursuivre avec de nouvelles fonctionnalités sans interférence. Cela résout le problème de la coordination des modifications non publiées à travers l'écosystème PlotPyStack (DataLab, SigimaX, Sigima, PlotPy, guidata, PythonQwt) en fournissant une branche stable pour les tests CI pendant les cycles de maintenance.

4.4.4 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.

4.4.5 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.

⚠ Si vous contribuez aux fonctionnalités de traitement de DataLab, vous devrez travailler à la fois sur le dépôt principal de DataLab et sur celui de Sigima. Pour installer `sigima` en mode éditable, utilisez la commande suivante :

```
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
3.10	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 DATALAB_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).

Installeur Windows

L'exécutable Windows officiel et l'installeur MSI sont générés par le pipeline de publication de l'intégration continue GitHub. La version de Python embarquée dans l'application autonome distribuée aux utilisateurs finaux est définie par la valeur par défaut de l'entrée `python-version` du workflow de build réutilisable (`.github/workflows/_build.yml`), utilisée par son job `build-msi` ; les workflows de publication ne la redéfinissent pas. Ce workflow est la source de vérité unique — reportez-vous-y pour connaître la version exacte actuellement utilisée.

L'installateur Windows est construit avec WiX Toolset V4.0.5. L'utilisation de WiX Toolset nécessite le SDK .NET V6.0 au minimum.

Vous pouvez installer le SDK .NET avec `winget` :

```
winget install Microsoft.DotNet.SDK.8
```

Une fois le SDK .NET installé, WiX Toolset peut être installé et configuré avec les commandes suivantes :

```
dotnet tool install --global wix --version 4.0.5
wix extension add WixToolset.UI.wixext/4.0.5
```

Vous devez d'abord générer le script de l'installateur à partir d'un modèle générique :

```
python wix/makewxs.py
```

La construction de l'installateur s'effectue avec la commande suivante :

```
wix build .\wix\DataLab.wxs -ext WixToolset.UI.wixext
```

Test de fumée avant publication

Avant de téléverser l'installeur MSI, le job `build-msi` du workflow de build réutilisable le soumet à un test de fumée avec `scripts/frozen_smoke_test.py` : il installe le MSI dans un dossier jetable, compare les fichiers installés avec le dossier PyInstaller, démarre le `DataLab.exe` installé en mode non interactif avec captures d'écran, exécute `DataLab.exe --smoke-test` (vérification de l'installation implémentée dans `datalab/utils/smoketest.py`), désinstalle le MSI et vérifie que les artefacts de publication n'ont pas été modifiés. Tout échec fait échouer le job : rien n'est alors publié. Les captures d'écran, rapports et journaux sont téléversés dans l'artefact `msi-smoke-test`.

Pour l'exécuter localement, après avoir construit l'exécutable et l'installeur (le script refuse de s'exécuter si DataLab de même version majeure est installé sur l'ordinateur, car il désinstalle le MSI à la fin) :

```
python scripts/frozen_smoke_test.py all --msi wix/DataLab-1.4.0.msi --dist dist/
↪DataLab --workdir build/msi-smoke-test
```

Les commandes `gui` et `probe` de ce script se contentent de démarrer un exécutable (par exemple `dist/DataLab/DataLab.exe`) avec captures d'écran, ou d'exécuter sa vérification d'installation.

Lorsque DataLab nécessite des versions de Sigima, SigimaX ou DataLab-Capsule qui ne sont pas encore publiées, exécuter le workflow « Release (dry-run) » avec l'entrée `dependencies` à `development` : ces packages sont alors construits depuis leur branche de même nom que la branche choisie, ou depuis `develop`.

Logiciels tiers

Les logiciels suivants peuvent être nécessaires pour maintenir le projet :

Logiciel	Description
gettext	Traductions
Git	Système de contrôle de version
MikTeX	Distribution LaTeX sur Windows

4.4.6 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 intégrée à la version 1.0 (novembre 2025) :• L'objectif était de découpler le modèle de données, le calcul et l'E/S de l'interface utilisateur• Cela a conduit à la création de la bibliothèque Sigima, qui gère désormais le traitement des données indépendamment de l'interface graphique de DataLab• Cette refonte améliore la maintenabilité, la testabilité et l'extensibilité de la base de code• Cela permet également à DataLab d'être utilisé comme une bibliothèque (via Sigima) dans d'autres projets logiciels, favorisant l'intégration et la réutilisation

🏗 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• Prise en charge des séries temporelles• Connecteurs de base de données• Plugin Spyder pour l'analyse de données interactive
🛠 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 ✓ Livrée

Une interface web a été livrée sous le nom de DataLab-Web, l'édition native pour navigateur de la plateforme (voir la présentation de l'écosystème). Plutôt que de s'exécuter sur un serveur, elle exécute le moteur de calcul Sigima entièrement dans le navigateur — CPython compilé en WebAssembly via Pyodide — associé à une interface utilisateur React. DataLab reste ainsi une application locale : aucun serveur, aucun compte, et les données ne quittent jamais la machine de l'utilisateur, conformément à la philosophie de confidentialité du projet. Essayez-la sur datalab-platform.com/web.

⏱ 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 (XML-RPC)) - 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 (XML-RPC)), 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 ✓ Livrée

> ⓘ Un noyau Jupyter a été livré sous le nom de DataLab-Kernel (voir sa documentation et la présentation de l'écosystème), offrant un accès piloté par notebook aux espaces de travail DataLab, avec synchronisation en direct optionnelle avec l'interface graphique de bureau. Les notes d'exploration originales sont conservées ci-dessous à titre de contexte.

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 (XML-RPC)), 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
0.10212	• 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
0.9201 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 5

Notes de version

Cette section contient les notes de version pour toutes les versions de DataLab, documentant les nouvelles fonctionnalités, les améliorations, les correctifs et les changements majeurs.

5.1 Version 1.4

5.1.1 DataLab Version 1.4.0

Prérequis

- DataLab-Platform :
  - Sigima 1.4.0 ou ultérieur est requis : il fournit les contrats d'opération et la qualification de rejeu, l'import d'images RAW, l'export d'images adapté au format, les annotations portables, les métadonnées d'origine expérimentale, la sérialisation d'objets au format NPZ et les règles de sélection par centre de pixel des ROI.
  - SigimaX 1.1.0 ou ultérieur est requis : il fournit les adaptateurs d'annotations PlotPy portables, les styles de courbes à contraste élevé, le rendu haute résolution de l'écran de démarrage et du filigrane, ainsi que l'exploration HDF5 NeXus et multidimensionnelle.
  - DataLab-Capsule 0.2.x (`>= 0.2.0, < 0.3`) est requis pour l'enregistrement et la vérification de la provenance des espaces de travail ainsi que l'échange de fichiers `.dlcapsule`.
- PlotPyStack :
  - guidata 3.17.2 ou ultérieur est requis : il fournit les éléments de plage à histogramme, la priorité des icônes qualifiées par module, les fonctions d'images haute résolution, et donne à chaque fenêtre une barre de titre suivant le thème de couleurs sous Windows, y compris aux fenêtres des plugins (corrige Issue #492).
  - PlotPy 2.12.1 ou ultérieur est requis : il fournit des icônes SVG nettes sur les écrans haute résolution, le sous-échantillonnage automatique min/max des signaux et des interactions plus fluides sur les superpositions d'images.
- Bibliothèques tierces :
  - Le package Python et les nouveaux projets de plugins générés exigent désormais Python 3.10 ou ultérieur. Les utilisateurs de Python 3.9 doivent conserver une version antérieure compatible. L'application autonome continue d'inclure son propre interpréteur Python.
  - Le lanceur de tests Windows exclut Python 3.9 des matrices courantes, tout en conservant les tests explicites de versions historiques précisément sélectionnées.

— pip 22.2 ou ultérieur est requis pour résoudre et installer les dépendances additionnelles des plugins sous forme de wheels.
— NumPy jusqu'à la version 2.5 (`>= 1.22, < 2.6`) et SciPy jusqu'à la version 1.18 (`>= 1.10.1, < 1.19`) sont désormais pris en charge.

✧ Nouvelles fonctionnalités

Page d'accueil :

— Une nouvelle page d"Accueil, semblable à celle de DataLab-Web, est affichée au démarrage et chaque fois que le panneau signal ou image courant est vide. Elle rassemble les principales actions pour bien démarrer : créer un signal ou une image, ouvrir des fichiers, parcourir un fichier HDF5, ouvrir un espace de travail HDF5, importer des données texte, interroger l'assistant IA, suivre la visite guidée, lancer la démo, lire la documentation ou consulter les notes de version
— La page est un panneau ancrable, en onglet avec les vues signal et image : elle reste disponible après la création ou l'ouverture de données, et peut être rouverte à tout moment depuis le menu **?** (Page d'accueil) ou depuis le menu Affichage
— Une option Afficher la page d'accueil lorsque le panneau courant est vide, disponible sur la page elle-même et dans les paramètres généraux, détermine si la page est affichée automatiquement, y compris au démarrage

Luminosité et contraste :

— Une nouvelle opération Traitement > Exposition > Luminosité et contraste fournit un histogramme avec des contrôles synchronisés du minimum, du maximum, de la luminosité et du contraste, des actions Auto et Réinitialiser déterministes, un remappage tenant compte des ROI et un aperçu interactif. Une plage d'entrée est initialisée à partir de la première image sélectionnée et appliquée à toute la sélection, chaque résultat conservant son image source et son type de données.
— La fenêtre d'intensité peut s'étendre au-delà de la plage de données observée. Les limites en virgule flottante exactes survivent à l'édition et à la réapplication, tandis que les contraintes de paramètres déclarées restent appliquées et Annuler préserve les paramètres d'origine.

Import d'images RAW :

— Les fichiers d'images RAW sans en-tête (`*.raw`) peuvent désormais être ouverts depuis Fichier > Ouvrir…, Fichier > Ouvrir depuis un répertoire…, par glisser-déposer, avec l'Assistant IA (outil `load_file`) et par contrôle à distance (`load_from_files` et `load_from_directory` avec un argument optionnel `RawImageImportParam`) (implémente l'issue #237)
— La boîte de dialogue Import d'image RAW définit le type de données des pixels, la largeur et la hauteur de l'image, le décalage en octets avant la première image, le nombre d'images, l'espacement en octets entre images consécutives, l'ordre des octets, ainsi que l'option Blanc à zéro (qui affiche l'image importée avec une palette de niveaux de gris inversée sans modifier les valeurs des pixels). Les fichiers RAW multi-images créent une image par trame
— Lorsque plusieurs fichiers RAW sont importés simultanément, DataLab demande s'il faut appliquer les mêmes paramètres d'import à tous les fichiers ou définir les paramètres de chaque fichier (chaque boîte de dialogue reprenant les valeurs validées pour le fichier précédent) ; les actions enregistrées dans l'historique conservent les paramètres d'import afin que leur rejeu ne rouvre pas la boîte de dialogue

Export d'images adapté au format :

— L'enregistrement d'une image avec Fichier > Enregistrer… ouvre désormais une boîte de dialogue Exporter l'image adaptée au format de fichier sélectionné, avec un aperçu en direct affichant l'image en niveaux de gris telle qu'elle sera relue depuis le fichier exporté (y compris la quantification et la compression avec perte pour les fichiers BMP, PNG, JPEG, JPEG 2000 et TIFF) (implémente l'issue #256)
— La préparation des données couvre la normalisation (aucune, min-max, percentiles ou bornes manuelles), le comportement hors plage (écrêtage ou rééchelonnage), le traitement des valeurs non finies (erreur, écrêtage ou remplacement), la correction gamma, l'inversion et la conversion vers un type de données pris en charge par le format de destination ; lors de l'export d'une image flottante ou d'un type non pris en charge vers un format uniquement entier (comme PNG, JPEG ou BMP), le rééchelonnage min-max est sélectionné par défaut, et les images complexes sont refusées lorsque le format ne peut pas stocker de valeurs complexes
— Les options propres au format sont proposées et validées pour chaque format (niveau de compression PNG, qualité/sous-échantillonnage/progressif/optimisation/lissage JPEG, mode et couches de qualité/irréversible/ordre

de progression/résolutions/tuilage/marqueurs PLT JPEG 2000, compression/niveau/prédicteur/lignes par bande/tuilage/résolution/interprétation photométrique TIFF, séparateur et précision des fichiers texte et CSV, compression MAT-File), et les actions enregistrées dans l'historique conservent les paramètres d'export ainsi que les UUID des objets cibles

Aperçu interactif des traitements :

- Les boîtes de dialogue des traitements 1-vers-1 compatibles proposent désormais un aperçu interactif facultatif pour les signaux et images, y compris les opérations image-vers-signal. L'aperçu est désactivé par défaut ; Annuler laisse l'espace de travail inchangé, tandis que OK réutilise un aperçu courant terminé pour une source unique et exécute sinon le traitement normal sur la sélection initiale.
- Les aperçus terminés conservent leur processus dédié prêt pour les boîtes de dialogue suivantes, ce qui évite des démarrages de processus répétés. L'annulation d'un aperçu encore en cours arrête son processus afin qu'aucun calcul spéculatif ne se poursuive en arrière-plan.
- Les paramètres numériques bornés proposent désormais des curseurs en complément d'une saisie textuelle précise. L'onglet Traitement conserve Appliquer et le retraitement automatique, avec des mises à jour différées jusqu'au relâchement du curseur et à une saisie valide.

Statut des plugins :

- Cliquez sur le libellé Plugins : de la barre de statut pour ouvrir Configurer les plugins…, comme avec les libellés de statut de l'API Web et de la console interne. Le raccourci est disponible lorsque les plugins sont globalement activés, y compris lorsqu'aucun plugin n'est chargé ou qu'un plugin n'a pas pu être chargé.

Plugins d'application :

- Ajout d'une entrée de premier niveau Applications qui présente les plugins d'application scientifique actifs dans un nouveau panneau Applications
- Chaque application affiche sa description, sa version, ses méthodes, ses outils et ses exemples ; les plugins de traitement seul restent en dehors de cette vue ciblée
- Chaque méthode présente les entrées qu'elle attend (type et nombre d'objets, métadonnées requises) et un statut mis à jour indiquant si la sélection courante peut être analysée, et sinon pourquoi
- Lorsque des métadonnées requises manquent, le statut renvoie vers Édition > Métadonnées > Ajouter des métadonnées…, dont la liste Clés connues propose aussi les clés attendues par les méthodes des applications ; le statut est mis à jour dès que des objets sont modifiés
- Exécuter… lance directement une méthode sur la sélection courante lorsque celle-ci convient, et ne demande l'affectation des entrées que si elle est ambiguë ou invalide ; la boîte de dialogue d'affectation vérifie les objets choisis avant de permettre l'exécution
- Les exemples sont listés sous les méthodes pour lesquelles ils sont conçus : Essayer avec cet exemple ouvre l'exemple, préremplit les paramètres de la méthode et exécute celle-ci une fois qu'ils sont acceptés. Un exemple peut servir plusieurs méthodes, et les exemples conçus pour aucune méthode sont listés comme jeux de données
- Les plugins d'application peuvent aussi lister des outils, comme des assistants ou des éditeurs, dans le panneau Applications
- Les outils de plugin apparaissent aussi dans le sous-menu du plugin du menu Plugins, dans le panneau Signal ou Image sur lequel ils travaillent. Un outil qui a besoin d'une sélection est désactivé jusqu'à ce que les bons objets soient sélectionnés, et son bouton dans le panneau Applications indique quoi sélectionner
- Les plugins d'application peuvent fournir des instruments, comme des caméras ou des oscilloscopes simulés, sans écrire d'interface utilisateur : DataLab affiche une fenêtre avec une vue en direct et les réglages de l'instrument, rafraîchie à chaque modification des réglages ou en continu en mode En direct, et Acquérir ajoute les objets acquis dans un nouveau groupe, prêts pour les méthodes de l'application
- L'option Nouveau groupe de la fenêtre d'instrument peut être désactivée pour ajouter les acquisitions suivantes au groupe courant du panneau, au lieu de créer un groupe pour chacune d'elles. L'option et le groupe courant sont pris en compte au clic sur Acquérir, de sorte que sélectionner un autre groupe pendant une acquisition ne change pas sa destination. Chaque instrument définit l'état par défaut de l'option, et le choix de l'utilisateur est mémorisé pour chaque instrument
- Les instruments sont appelés en dehors du fil d'exécution de l'interface utilisateur, un appel à la fois : un instrument lent, comme un appareil réel attendant ses données, ne fige plus DataLab, et les demandes de rafraîchissement répétées entre-temps sont regroupées. Les instruments peuvent aussi déclarer des méthodes `open` et `close`, appelées à l'ouverture et à la fermeture de leur fenêtre (y compris lors du rechargement des plugins ou de la fermeture de DataLab), par exemple pour se connecter à un appareil puis le libérer. Les appels restent successifs lorsqu'une

fenêtre est fermée puis rouverte pendant un appel. Lorsque `open` échoue, son message reste affiché dans la fenêtre et l'instrument est ouvert à nouveau avant le rafraîchissement ou l'acquisition suivant, par exemple après une modification des réglages

- Les développeurs de plugins décrivent les entrées des recettes de manière déclarative (titres, effectifs minimaux, métadonnées requises) et peuvent ajouter des suggestions d'affectation et des vérifications rapides des entrées ; chaque recette s'exécute alors via le lanceur générique de DataLab, sans boîte de dialogue d'entrées propre au plugin
- Le panneau Applications est un panneau comme les autres, qui ne masque pas l'espace de travail de DataLab : la première fois, il s'affiche en fenêtre flottante à côté de la fenêtre principale, sur toute la hauteur de celle-ci, afin que les données et leurs vues restent visibles pendant son utilisation. Il peut ensuite être ancré (par exemple à côté de la liste des signaux ou des images dont il analyse la sélection), regroupé en onglets avec un autre panneau ou laissé flottant, et DataLab l'affiche à nouveau là où l'utilisateur l'a laissé, avec la même taille ; lorsqu'il est demandé alors qu'il est déjà affiché, sans changer d'application, il clignote brièvement pour indiquer où il se trouve. Il reste ouvert après le lancement d'une méthode ou l'ouverture d'un exemple
- Le panneau reste compact et lisible lorsqu'il est ancré : chaque méthode occupe une ligne, avec une pastille de couleur et un court texte indiquant si la sélection courante peut être analysée, et un bouton Exécuter… ; sous la liste, une zone Détails donne tous les détails de la méthode sélectionnée (description, raisons de son statut, entrées attendues avec la description et la clé de leurs métadonnées, exemples). Le résultat de la dernière exécution d'une méthode s'affiche sous sa ligne. Les sections de la liste peuvent être repliées, et la hauteur des détails réglée à l'aide d'un séparateur : DataLab mémorise ces choix
- Lorsque plusieurs plugins d'application sont installés, leur nombre est indiqué en haut du panneau, au-dessus de la liste permettant de choisir l'application ; la documentation et les options de page d'accueil de l'application se trouvent dans le menu situé à côté
- Les méthodes d'application terminées sélectionnent le dernier objet qu'elles ont généré, de manière cohérente avec les traitements standard de DataLab
- Les plugins d'application ajoutent des tuiles à une nouvelle section Applications de la page d'accueil : par défaut, une tuile par plugin affiche sa page dans le panneau Applications, et les plugins peuvent déclarer des tuiles supplémentaires, par exemple pour ouvrir directement un exemple. Les tuiles supplémentaires sont affichées lorsqu'il y a la place nécessaire, et passent sinon dans le menu de la tuile principale
- La page d'accueil reste lisible lorsque de nombreuses applications sont installées : les tuiles sont limitées à un nombre de lignes configurable (deux par défaut), une dernière tuile affichant le panneau Applications pour les autres. Les applications peuvent être épinglées en tête ou masquées, et celles utilisées récemment viennent en premier ; le panneau Applications propose les mêmes options de page d'accueil
- Les plugins peuvent déclarer une icône, affichée dans le panneau Applications, sur leur tuile de la page d'accueil et dans la boîte de dialogue Configurer les plugins…
- DataLab vérifie les déclarations des plugins d'application (méthodes, exemples, outils, tuiles de la page d'accueil) lors de leur chargement : un plugin dont une déclaration est invalide, par exemple une méthode dont la version du plugin n'a pas été mise à jour avec celui-ci, ou un exemple faisant référence à une méthode renommée, n'est pas chargé et est listé avec son erreur dans Configurer les plugins…, tandis que DataLab et les autres plugins démarrent normalement (corrige l'issue #484)

Installation de plugins :

- Les catalogues Desktop en ligne et hors ligne acceptent désormais les plugins en fichier unique `datalab_*.py` à côté des wheels. Les fichiers historiques n'ont besoin ni de projet de packaging ni de modification du code source pour être référencés ; l'inspection du catalogue ne les importe ni ne les exécute jamais (implémente l'issue #419)
- Les versions connues des scripts peuvent être mises à jour ; les fichiers sans version sont identifiés par SHA-256 et proposent un remplacement explicite lorsque leur contenu change. Les scripts précédemment installés depuis un fichier sont reconnus et les copies de développement existantes ne sont pas écrasées
- Les commandes de catalogue local gèrent les dossiers mêlant scripts et wheels. Les plugins en fichier unique utilisent les bibliothèques déjà disponibles dans DataLab et ne déclenchent pas de téléchargement de dépendances. La documentation distingue désormais explicitement l'organisation du code source, les formats de distribution et les capacités fonctionnelles, y compris dans le guide hors ligne français
- Les plugins peuvent désormais être installés depuis un fichier, sans aucun outil Python, y compris dans la version autonome de DataLab : le nouvel onglet Depuis un fichier de Plugins > Configurer les plugins… accepte un wheel en Python pur (`.whl`) ou un module unique `datalab_<nom>.py`

- Avant l'installation, DataLab vérifie le fichier sans l'exécuter et affiche son nom, sa version, ses classes de plugins, ses dépendances et son empreinte SHA-256
- Les plugins peuvent désormais fournir des dépendances supplémentaires sous forme de paquets wheel séparés, y compris des bibliothèques natives compatibles, sans modifier les paquets déjà fournis par DataLab. Un catalogue local complet peut être installé sans accès à Internet ; les fichiers manquants et les conflits de versions sont signalés avant toute activation d'une installation partielle
- La boîte de dialogue de confirmation liste les paquets supplémentaires et leurs empreintes. Ils sont partagés entre les plugins d'un même environnement d'exécution Python, activés après redémarrage de DataLab et conservés lorsqu'un plugin est désinstallé
- Les plugins installés sont listés dans le même onglet, où ils peuvent être désinstallés ; un plugin nouvellement installé est activé après rechargement si aucun paquet supplémentaire n'est nécessaire, ou après redémarrage si des dépendances ont été ajoutées. Une nouvelle version d'un plugin déjà chargé est utilisée au prochain démarrage
- Le nouvel onglet Catalogue liste les plugins du catalogue de plugins DataLab, officiels ou issus de la communauté, avec un champ de recherche : Installer et Mettre à jour téléchargent le wheel, le vérifient par rapport au catalogue et l'installent après confirmation. L'adresse du catalogue peut être modifiée pour utiliser une copie privée
- Les organisations peuvent diffuser leurs propres plugins, ou les versions de plugins publics qu'elles ont validées, depuis des catalogues locaux de plugins : des dossiers, généralement partagés sur le réseau, déclarés dans l'onglet Avancé ou par la variable d'environnement `DATALAB_PLUGINS_CATALOG`. Leurs plugins sont listés dans l'onglet Catalogue avant ceux du catalogue de plugins DataLab, et les masquent lorsque les deux proposent le même plugin
- Un dossier de catalogue local peut contenir des wheels et des modules en fichier unique, listés à mesure qu'ils y sont copiés, ou un index `catalog.json` indiquant les noms d'affichage, les dépréciations et les versions retirées ; `datalab-plugin catalog build` écrit cet index en conservant les métadonnées administrées, et `datalab-plugin catalog check` signale les fichiers manquants, modifiés, refusés ou non listés
- Les auteurs peuvent préparer un catalogue hors ligne avec `datalab-plugin catalog prepare` : la commande télécharge les paquets de dépendances supplémentaires nécessaires à l'environnement hôte de référence sans les installer, puis écrit le catalogue. Le dossier complet peut être distribué sur un partage réseau ou copié sur des ordinateurs sans accès à Internet
- Un catalogue illisible, comme un partage réseau non connecté ou le catalogue en ligne sur un ordinateur sans accès à Internet, est signalé dans l'onglet Catalogue sans masquer les plugins des autres catalogues
- La boîte de dialogue Configurer les plugins… s'ouvre plus grande et conserve la taille qu'elle avait à sa dernière fermeture. Ses onglets suivent l'ordre d'utilisation et ont des titres explicites, décrits par leurs info-bulles : Installés, Catalogue, Depuis un fichier et Avancé

Générateur de projets de plugins :

- `datalab-plugin create` crée désormais par défaut un plugin en fichier unique, la façon la plus simple d'étendre DataLab : la commande demande seulement le nom du plugin et sa description, puis écrit un fichier `datalab_<nom>.py` dans le répertoire de plugins de l'utilisateur. Une fois les plugins rechargés, le plugin ajoute son sous-menu au menu Plugins, avec un traitement d'exemple des signaux sélectionnés (ou des images, avec `--object-kind image`), dont la courte fonction NumPy est la seule partie à modifier. Le fichier n'écrase jamais un fichier existant, et peut être partagé et installé avec Plugins > Configurer les plugins… > Depuis un fichier. Le projet installable structuré en couches, décrit ci-dessous, est créé avec `--project` (implémente l'issue #413)
- Durcissement de `datalab-plugin create` à partir du retour du pilote Camera : les projets générés séparent désormais dès le premier commit le code `core` indépendant de l'hôte, l'orchestration `workflow` sans interface et les `adapters` Desktop/Web
- La racine des packages générés expose une identité stable sans importer Qt ; le point d'entrée installé cible l'adaptateur Desktop, tandis que la prise en charge Web est explicitement marquée comme non prise en charge
- Chaque projet généré comprend un test de non-régression exécutable de l'architecture, une documentation de contribution et d'architecture, ainsi qu'un changelog
- Les noms et descriptions de plugins longs sont désormais mis en forme afin que les projets nouvellement générés réussissent leurs contrôles Ruff intégrés sans modification manuelle du code source
- Les projets générés sont prêts à publier : un workflow de release joint le wheel à une release GitHub lorsqu'une étiquette de version est poussée et affiche la fiche à soumettre au catalogue de plugins DataLab ; `pyproject.toml` exige la version majeure courante de DataLab et déclare les URL du projet
- `datalab-plugin create --project` demande le compte GitHub qui héberge le projet et en déduit l'ID de plugin par défaut (`io.github.<compte>.<nom>`), les ID `org.datalab.` étant réservés aux plugins de

DataLab-Platform

Ajout de métadonnées :

- Édition > Métadonnées > Ajouter une métadonnée… peut désormais extraire la valeur du texte formaté avec une expression régulière, par exemple un temps d'exposition ou un numéro de tir lu dans les titres des objets. Les objets sans correspondance restent inchangés, sauf si vous demandez une erreur, et un facteur d'échelle convertit les valeurs numériques (par exemple des millisecondes en secondes)
- Les clés de métadonnées peuvent désormais contenir des points et des tirets, comme les clés des plugins, et une liste Clés connues copie une clé déjà présente sur les objets sélectionnés

Provenance de l'espace de travail :

- DataLab enregistre désormais la manière dont chaque résultat a été obtenu : le traitement appliqué, ses paramètres, les objets utilisés et leur contenu exact, ainsi que les versions des logiciels. Cet enregistrement est sauvegardé avec l'espace de travail HDF5 et relu à sa réouverture, par DataLab ou par DataLab-Web : l'historique d'un résultat survit ainsi à l'enregistrement, à la fermeture et au partage
- Les traitements de signaux et d'images, les opérations combinant plusieurs objets et les analyses (résultats géométriques et tableaux) sont enregistrés, chacun avec l'identité de son opération Sigima ; les régions d'intérêt, les incertitudes, l'alignement en X appliqué aux signaux sur des grilles différentes et les options globales lues par le calcul (comme la normalisation des noyaux) font partie de l'enregistrement, et les signaux et images de tous les types numériques (float32, complexes…) sont couverts. Une analyse qui ne trouve rien est enregistrée comme telle. Un résultat dont les données, l'étalonnage ou les régions d'intérêt ont été modifiés après son calcul est signalé comme altéré plutôt que considéré comme fiable
- Tout traitement ou toute analyse enregistrés d'une opération Sigima qualifiée (toutes sauf le spectre de phase) peuvent être recalculés et chaque résultat comparé au résultat stocké : exactement, ou dans la tolérance déclarée pour l'opération lorsque les bibliothèques numériques diffèrent (exact, dans la tolérance, différent, ou non vérifié avec la raison). Le recalcul est disponible depuis l'onglet Provenance du panneau des propriétés (Vérifier le calcul) et depuis les scripts ; il ne modifie jamais l'espace de travail. Lorsque seules les régions d'intérêt d'une entrée ont changé depuis le calcul (par exemple des régions créées sur l'image analysée par une détection), il est effectué sur une copie portant les régions d'intérêt enregistrées, si bien qu'une telle analyse reste vérifiable
- L'onglet Historique du panneau des propriétés est renommé Provenance (pour éviter toute confusion avec le panneau Historique) et affiche désormais la provenance de l'objet sélectionné : l'exécution enregistrée qui a produit son contenu actuel, ses paramètres et ses entrées, et, étape par étape, la manière dont chaque entrée a elle-même été produite, telle qu'elle était à ce moment-là (par exemple le sigma réellement utilisé, même si l'entrée a été filtrée à nouveau depuis), avec le logiciel qui l'a enregistrée (DataLab Desktop ou DataLab Web). Chaque étape est une fiche avec ses paramètres, ses entrées et ses résultats, et des liens placés à côté de ce sur quoi ils agissent : afficher l'étape enregistrée d'une entrée, Aller à l'objet, Vérifier cette étape, Action de l'historique et détails. Contrôler les données compare l'objet, ses entrées directes et les étapes antérieures affichées à leurs états enregistrés sans calculer ; les entrées supprimées sont signalées immédiatement, les entrées modifiées ou indisponibles après contrôle. Aller à l'action de l'historique d'une étape laisse l'onglet Provenance ouvert
- L'onglet Traitement restaure l'appel enregistré du résultat sélectionné, y compris pour les opérations combinant plusieurs objets (différence de deux signaux, moyenne de plusieurs images…) : chaque entrée est conservée dans l'ordre avec son rôle, les paramètres peuvent être modifiés, et les traitements sans paramètres proposent un bouton Réappliquer. Lorsque le traitement ne peut pas être appliqué à nouveau (par exemple si une entrée a été supprimée), l'onglet en indique la raison. Cela fonctionne aussi pour les espaces de travail rouverts depuis un fichier, enregistrés par DataLab ou par DataLab-Web
- Les actions de l'historique sont liées à la provenance de leur dernière exécution réussie : Afficher la provenance de l'exécution dans le panneau Historique l'affiche, et la provenance d'un résultat mène à l'action de l'historique qui l'a produit, analyses comprises. En mode d'édition du panneau Historique, modifier l'un des résultats d'une commande en produisant plusieurs ne change que ses paramètres dans l'action
- Lorsqu'un fichier d'espace de travail contenant une provenance est ajouté à l'espace de travail actuel (sans le vider), DataLab indique désormais, avant toute modification, que sa provenance antérieure ne sera pas importée : dans la question proposant de vider l'espace de travail (les fichiers sont choisis d'abord), ou dans un avertissement unique lorsque cette question n'est pas posée. Les objets ajoutés n'héritent jamais de la provenance d'un autre objet (ils ne sont ni affichés ni utilisés comme entrée d'origine d'une étape enregistrée, y compris lors de sa vérification), et leurs traitements ultérieurs sont enregistrés
- DataLab Desktop et DataLab-Web enregistrent les mêmes traitements de manière identique et rejouent les résul-

tats l'un de l'autre
- Avant d'interpoler des signaux dont les grilles en X diffèrent, DataLab demande toujours confirmation ; l'alignement appliqué est celui qu'il a toujours appliqué, désormais nommé et enregistré afin qu'un recalcul l'applique à l'identique. Lorsque plusieurs paires sont sélectionnées pour une opération sur deux signaux, chacune aligne désormais l'opérande sur la grille de son propre signal source
- Les espaces de travail enregistrés par des versions antérieures conservent leur historique de traitement tel qu'il a été stocké : DataLab montre ce qui est connu et ce qui manque, n'invente rien, et ne recalcule jamais ces entrées
- Un espace de travail dont l'enregistrement de provenance est endommagé est refusé avec un message explicite, et l'espace de travail actuel reste inchangé
- Les espaces de travail peuvent être échangés sous forme de capsules (un espace de travail et une description de sa provenance) depuis Fichier > Capsule d'espace de travail…, la même boîte de dialogue que dans DataLab-Web, afin qu'une autre personne puisse les vérifier et les rejouer avec l'une ou l'autre édition :
  - Exporter l'espace de travail, avec un nom, une description, des pièces jointes légères (protocole, notes, petits tableaux) et, pour un espace de travail ouvert depuis une capsule (y compris après l'avoir enregistré puis rouvert), la capsule dont il est une révision ; l'espace de travail peut aussi être référencé par son adresse de publication au lieu d'être inclus
  - Inspecter une capsule sans l'ouvrir : ses activités, ses pièces jointes (enregistrées seulement à la demande, jamais ouvertes), son intégrité (espace de travail et états enregistrés vérifiés, rien n'est recalculé) et les rapports de vérification et revues fournis avec elle, chacun vérifié par rapport à la capsule ; en rédiger une revue, ou Ouvrir comme espace de travail (action enregistrée dans le panneau Historique)
  - Rejeu local recalcule une activité enregistrée de l'espace de travail ouvert et la compare à son résultat enregistré, sans modifier l'espace de travail ; le rapport de vérification peut être enregistré pour être partagé avec la capsule
  - Révisions énumère les capsules exportées, inspectées ou récupérées sur cet ordinateur et compare deux d'entre elles (changements scientifiques, de présentation, de stockage, d'environnement et de pièces jointes)
  - Registre se connecte à un registre de capsules facultatif pour publier des capsules (jamais leurs espaces de travail) et des revues, lister les révisions publiées et les récupérer, tout ce qui est renvoyé étant vérifié
  - Les scripts et les macros peuvent aussi exporter et ouvrir des capsules (`export_capsule` et `open_capsule` du proxy), et la ligne de commande `workspace-capsule` de DataLab-Capsule permet toujours de les valider, de les comparer et de les vérifier
  - Une capsule, ou l'espace de travail qu'elle référence, qui dépasse la limite de taille est refusée avant d'être lue, et DataLab n'écrit jamais une capsule qu'il refuserait d'ouvrir, ni la capsule ou son espace de travail référencé l'un sur l'autre ou sur une pièce jointe ; les limites et les coûts mesurés sont indiqués dans la page Capsules d'espace de travail
  - La version autonome (installeur) embarque ce dont les capsules ont besoin (schémas et contexte RO-Crate de DataLab-Capsule)

Références aux objets dans les titres :

DataLab utilise désormais des références aux objets stables, fondées sur les UUID, dans les arborescences d'objets, les titres de résultats, les légendes des graphiques et les espaces de travail HDF5 (implémente Issue #367 et Issue #149).

- Les signaux, images et groupes des arborescences affichent désormais leur titre courant suivi de leur propre référence stable `#UUID8` (`#gUUID8` pour les groupes), à la place des anciens identifiants `s001`, `i001` et `gs001`
- Les références aux sources intégrées dans les titres de résultats ne changent plus lorsque des objets sont réordonnés, supprimés ou renumérotés, et restent cliquables pour sélectionner l'objet source
- Le nouveau paramètre Références dans les titres de résultats permet d'afficher ces références soit sous forme d'UUID à 8 caractères (par défaut), soit avec les titres actuels des sources, qui suivent les renommages ultérieurs et sont résolus sur l'ensemble des chaînes de traitement (par exemple `derivative(derivative(Mon signal))`)
- Lorsqu'un objet source est supprimé, sa référence est affichée sous la forme de son UUID à 8 caractères au lieu de devenir un espace réservé anonyme
- Les titres des espaces de travail enregistrés par les versions précédentes conservent leurs anciennes références de type `s001` sous forme de texte simple

Listes de signaux et d'images :

Les signaux, images et groupes peuvent désormais être supprimés directement depuis les listes, comme dans DataLab-Web (implémente l'issue #449).

- Un bouton de suppression apparaît à droite du signal, de l'image ou du groupe survolé : il devient rouge lorsque

le curseur de la souris le survole, et son info-bulle indique ce qu'il supprime
- Un clic sur ce bouton supprime l'élément, ou toute la sélection lorsque l'élément en fait partie, après confirmation. Supprimer un élément non sélectionné laisse inchangés la sélection courante et les objets affichés
- Comme avec l'action Supprimer, la suppression est enregistrée dans le panneau Historique
- Le bouton peut être masqué avec le nouveau paramètre Afficher un bouton de suppression au survol des éléments des listes (Visualisation > Commun)

Formules des modèles d'ajustement :
- Les fenêtres d'ajustement interactif affichent désormais la formule du modèle ajusté sous les paramètres d'ajustement, et le libellé de chaque paramètre indique son symbole dans la formule (par exemple « Facteur d'échelle (x_0) ») : les paramètres enregistrés dans les métadonnées du signal ajusté peuvent ainsi être reliés au modèle sans lire le code source. La documentation Ajustement indique également, pour chaque modèle, le nom sous lequel chaque paramètre est enregistré (corrige l'issue #400)
- Les coefficients des ajustements polynomial et linéaire sont nommés `a`, `b`, `c`… comme dans la formule et dans les métadonnées, au lieu de `c2`, `c1`, `c0`
- La documentation Ajustement explique comment déduire la longueur d'onde du pic, la température et l'émissivité des paramètres d'un ajustement planckien

Contrôle à distance et plugins :
- Le plugin Tracé de contours d'isoniveau (`org.datalab.isolevel-map`, version 1.1.0), auparavant fourni comme exemple de plugin, est désormais un plugin intégré disponible directement depuis Plugins > Tracé de contours d'isoniveau > Afficher le tracé de contours…
- `add_group` renvoie désormais l'UUID du groupe créé, de sorte qu'un client distant, une macro ou un plugin peut y ajouter des objets avec `add_object(obj, group_id=...)` (ou `add_signal`/`add_image`) au lieu de rechercher le groupe par son titre
- `load_from_files` et `load_from_directory` acceptent désormais un argument optionnel `param` contenant les paramètres d'import propres au format (par exemple `RawImageImportParam` pour les images RAW), appliqués sans afficher de boîte de dialogue

Fichiers NeXus :
- Les groupes NeXus `NXdata` sont présentés comme des éléments uniques dans l'explorateur HDF5 et importés comme signaux ou images avec leurs axes, unités, libellés et, pour les signaux, incertitudes. Les bords de classes sont convertis en centres de classes, les axes d'image non uniformes sont conservés, et les axes d'image décroissants sont inversés avec l'image. Les axes invalides sont signalés dans l'explorateur au lieu d'être devinés
- Les fichiers écrits selon les conventions NeXus anciennes, comme les scans du synchrotron SOLEIL, sont importés avec leurs axes (attributs `axis` et `primary` des jeux de données d'axe), et chaque compteur et détecteur d'un scan apparaît comme un élément à part entière : les compteurs comme des signaux en fonction du moteur scanné, les détecteurs comme des images (un spectre par point) ou des piles d'images (une image par point) avec le moteur selon leur premier axe
- Le navigateur HDF5 sélectionne et affiche la mesure par défaut d'un fichier NeXus (attributs `default`) à son ouverture
- Les données de détecteur compressées, comme les données bitshuffle/LZ4 des détecteurs Eiger, sont désormais lues (avec hdf5plugin, y compris dans l'application autonome) ; un jeu de données compressé avec un filtre indisponible est signalé dans le navigateur au lieu de provoquer une erreur
- Les signaux et images importés enregistrent leur origine (fichier, groupe, signal et axes, conversions appliquées, titre et dates de l'entrée NeXus) dans leurs métadonnées `experimental_source`, enregistrées avec l'espace de travail : un espace de travail enregistré se rouvre sans le fichier NeXus
- Les fichiers `.nxs` et `.nx5` sont proposés à l'ouverture de fichiers HDF5
- Les plugins peuvent décrire les signaux et images des fichiers dont les groupes NeXus `NXdata` sont incomplets (pas d'axes, plusieurs compteurs…) avec des profils de correspondance déclaratifs (JSON) enregistrés par `sigimax.h5.mapping.register_mapping` : le navigateur HDF5 affiche alors chaque objet décrit comme un élément supplémentaire, importé avec ses axes, libellés et unités ; les axes d'un objet décrit peuvent ne décrire que ses premières dimensions, de sorte qu'un même profil s'applique aux compteurs et aux détecteurs d'un scan

Vérification de l'installation :
- La nouvelle option de ligne de commande `--smoke-test` vérifie que DataLab est correctement installé sans ouvrir sa fenêtre principale, par exemple `DataLab.exe --smoke-test report.json` pour la version autonome ou `datalab --smoke-test report.json` : les packages dont il dépend, les ressources fournies avec

lui (modèles de macros, documentation PDF, traductions françaises et icônes) et le traitement dans un processus séparé. Les résultats sont écrits dans un rapport JSON, et le code de sortie est non nul lorsqu'une vérification a échoué (voir Fonctionnalités de la ligne de commande)
— Avant sa publication, chaque installeur Windows est désormais installé sur un ordinateur de test, où l'application installée est démarrée puis vérifiée de cette façon (implémente l'issue #366)

Changements

Lisibilité des courbes :
— Les courbes des signaux sont plus lisibles, en particulier lorsque de nombreuses courbes sont affichées (corrige l'issue #453) :
  — Les couleurs des courbes ressortent désormais sur fond clair comme sur fond sombre : plusieurs couleurs de l'ancienne palette (orange, rose, vert-jaune, cyan) étaient pâles sur fond blanc, et les courbes grises se confondaient avec la grille
  — Au-delà des 10 premières courbes, les courbes sont désormais tracées uniquement en tirets : les traits mixtes (tiret-point et tiret-point-point), qui ne traçaient que la moitié de la courbe, ne sont plus utilisés
  — La grille des graphiques utilise désormais des traits pleins discrets au lieu de pointillés gris, qui se confondaient avec les courbes en tirets ; elle suit également les changements de thème de couleurs
— Les couleurs et les styles de trait des courbes dépendent désormais de la position des courbes parmi celles affichées dans le graphique, dans l'ordre des signaux du panneau : la première courbe affichée est toujours tracée avec la première couleur. L'ouverture de l'éditeur de ROI, de Voir dans une nouvelle fenêtre ou d'autres boîtes de dialogue ne décale plus les styles des courbes suivantes, ce qui pouvait faire apparaître des pointillés avec seulement quelques courbes affichées
— Les styles de courbes automatiques ne sont plus enregistrés dans les espaces de travail : seuls une couleur ou un style de trait choisis par l'utilisateur sont enregistrés et toujours conservés, quelle que soit leur valeur. À l'ouverture des espaces de travail et des fichiers d'historique enregistrés avec DataLab 1.3 ou une version antérieure, qui enregistraient le style automatique de chaque courbe, ces styles sont supprimés afin que les courbes soient affichées avec les nouveaux styles automatiques ; un style choisi avec ces versions est également supprimé lorsqu'il est identique à l'un de leurs styles automatiques. Les métadonnées importées depuis un fichier sont appliquées telles quelles
— Affichage > Réinitialiser les styles de courbes rend désormais aux signaux sélectionnés des styles de courbes automatiques, en supprimant les couleurs et les styles de trait choisis pour eux
— Sur les écrans haute résolution (par exemple avec une mise à l'échelle de l'affichage de 150 %), les courbes d'épaisseur 1 sont désormais tracées avec une largeur d'un pixel de l'écran, ce qui rend leur tracé beaucoup plus rapide : les signaux bruités étaient tracés jusqu'à plusieurs centaines de fois plus lentement qu'à 100 % (nécessite PlotPy ≥ 2.12.0)

Images synthétiques en coordonnées physiques :
— Les images gaussienne 2D, rampe 2D, réseau sinusoïdal, anneaux, mire de Siemens et sinus cardinal 2D étaient calculées sur le domaine [xmin, xmax] × [ymin, ymax] de leurs paramètres de création, mais étaient affichées en coordonnées pixel : leurs centres, largeurs et fréquences spatiales ne correspondaient pas aux axes (un réseau sinusoïdal de fréquence 0,1 sur [0, 100] présentait 10 périodes le long d'un axe gradué de 0 à 1023). Ces images sont désormais affichées sur leur domaine. La nouvelle option Coordonnées physiques de leurs paramètres de création rétablit les coordonnées pixel lorsqu'elle est désactivée (corrige l'issue #400)
— Les images créées avec des versions antérieures conservent leurs coordonnées pixel lorsque leurs paramètres de création sont modifiés ou recalculés

Visite guidée :
— La visite guidée ne démarre plus automatiquement au premier lancement : elle est désormais proposée depuis la page d'accueil, et reste disponible depuis le menu **?**

Réactivité avec les grandes campagnes :
— L'ouverture, la sélection et l'affichage de grandes campagnes générées ne répètent plus un rafraîchissement complet de l'espace de travail et des graphiques pour chaque objet
— Les espaces de travail HDF5 natifs et les résultats de méthodes multi-objets se chargent désormais par lots, ce qui rend les grandes collections de signaux et d'images disponibles nettement plus vite tout en préservant chaque objet sélectionné et visible

Réactivité de l'affichage des images :

— Le déplacement ou le redimensionnement de régions d'intérêt, d'annotations et de marqueurs sur les images est désormais beaucoup plus fluide, car l'image n'est plus recalculée à chaque mouvement de la souris : avec une image de 2048 × 2048 pixels, chaque mouvement de la souris prend environ 5 ms au lieu de 13 ms (13 ms au lieu de 33 ms avec une mise à l'échelle de l'affichage de 200 %). Le déplacement et le zoom des images sans pixels masqués sont également jusqu'à 25 % plus rapides (nécessite PlotPy ≥ 2.12.1)

Icônes nettes avec la mise à l'échelle de l'affichage :

— Les icônes des graphiques et des éditeurs fournies par PlotPy et guidata (barres d'outils des graphiques, panneau de contraste, coupes, liste des éléments, éditeur de tableaux, …) sont désormais des images SVG : elles restent nettes avec une mise à l'échelle de l'affichage de 125 %, 150 % ou 200 %, au lieu d'apparaître floues (nécessite PlotPy ≥ 2.12.1, corrige l'issue #470)

— L'action Traitement > Redimensionner des images affichait l'icône de redimensionnement des colonnes des éditeurs de guidata au lieu de sa propre icône : les icônes de DataLab ont désormais priorité sur les icônes de guidata et de PlotPy portant le même nom (nécessite guidata ≥ 3.17.1, corrige l'issue #472)

— Les icônes des plugins (`PluginInfo.icon`, outils et tuiles de la page d'accueil) peuvent également désigner une icône enregistrée par un module, par exemple `"datalab:delete.svg"`, en plus des ressources `package:path` et des noms de fichiers d'icônes de DataLab

Annotations de tracé portables :

— Les annotations de tracé sont désormais stockées dans un format indépendant du moteur de rendu, partagé avec DataLab-Web, de sorte que les annotations dans les espaces de travail et les fichiers `.dlabann` ne sont plus liées à PlotPy

— Les annotations PlotPy existantes restent lisibles et ne sont converties qu'après qu'une modification d'annotation a été acceptée ; ouvrir simplement un espace de travail ou annuler l'éditeur laisse ses données inchangées

— Les identifiants d'annotation, l'état de verrouillage, les métadonnées personnalisées et les données d'extension sont conservés lors des modifications, tandis que les charges utiles tierces inconnues sont conservées sans modification

— Les rectangles et segments tracés sur les signaux sont désormais les mêmes formes avec étiquette que sur les images ; contrairement aux images, les segments tracés sur les signaux n'affichent pas leur longueur et leur angle, qui mélangeraient les unités de X et de Y

Régions d'intérêt :

— Une région d'intérêt rectangulaire sélectionne désormais les pixels dont le centre est situé dans le rectangle, bornes comprises, comme les ROI circulaires et polygonales et comme l'affichage : ses bornes étaient arrondies au pixel le plus proche, ce qui pouvait décaler la sélection jusqu'à un demi-pixel. Un rectangle dont les bornes sont situées exactement sur des centres de pixels inclut désormais les pixels des deux bornes, soit une ligne et une colonne de plus qu'auparavant. Cela s'applique aux traitements et analyses dans les ROI, à l'extraction de ROI et à la zone de fond de la soustraction d'offset

— Les régions d'intérêt circulaires et polygonales sélectionnent elles aussi désormais les pixels dont le centre est situé dans leur forme, bord compris : les sommets des polygones étaient arrondis au pixel le plus proche, de sorte que les pixels sélectionnés pouvaient différer du polygone tracé sur l'image, et les cercles étaient tracés en unités de pixels sur les images à coordonnées non uniformes

— Une ROI rectangulaire définie en indices de pixels couvre désormais ses pixels jusqu'à leurs bords extérieurs, de sorte qu'elle conserve les mêmes pixels lorsqu'elle est convertie en coordonnées physiques et inversement

— Les cellules des grilles de ROI (ROI > Créer une grille de ROI…) sont désormais formées de pixels entiers, de sorte que des cellules adjacentes n'ont jamais de pixel en commun et qu'une grille pleine couvre chaque pixel exactement une fois, quelle que soit la taille de l'image ; sur les images à coordonnées non uniformes, la grille couvre désormais toute l'image, dont les dernières lignes et colonnes étaient omises

— Les régions d'intérêt enregistrées avec des versions antérieures ne sont pas converties : elles sélectionnent les pixels selon ces règles lorsqu'elles sont réutilisées, de sorte que les traitements et analyses dans ces régions peuvent donner des résultats légèrement différents lorsqu'ils sont recalculés

Profils d'image en coordonnées physiques :

— Les profils rectilignes, de segment, moyens et radiaux (Analyse > Profils d'intensité) sont désormais définis dans les coordonnées physiques de l'image, comme les régions d'intérêt et les résultats d'analyse, au lieu d'indices de pixels : les paramètres des profils indiquent des coordonnées dans les unités des axes de l'image, et l'axe X des signaux obtenus contient des coordonnées ou des distances physiques avec l'unité correspondante. Sur les images sans calibration (origine 0, taille de pixel 1), les résultats sont inchangés

- Le profil radial est désormais circulaire sur les images à pixels rectangulaires ou à coordonnées non uniformes, et son centre est correct sur les images calibrées, où le centre de l'image était mal placé
- Les profils de segment et radiaux sont refusés sur les images dont les unités X et Y diffèrent, les distances n'étant alors pas définies ; les coordonnées hors de l'image sont refusées au lieu d'être ramenées silencieusement au bord. Les lignes et segments tracés au-delà de l'image dans la boîte de dialogue de profil sont ramenés à ses pixels extrêmes
- Les scripts, macros et plugins doivent utiliser les nouveaux noms de paramètres (`y`/`x` pour les profils rectilignes, `x1`, `y1`, `x2`, `y2` pour les profils de segment et moyens, `xc`/`yc` pour les profils radiaux) et des coordonnées physiques. Les calculs de profils enregistrés dans la provenance d'espaces de travail sauvegardés avec des versions antérieures sont signalés comme non vérifiables, au lieu d'être rejoués avec une signification différente

Explorateur HDF5 :

- Parcourir un fichier HDF5 ne lit plus ses jeux de données en entier : ils sont listés et décrits à partir de leur forme et de leur type de données, et les aperçus de valeurs ne lisent que leurs premiers éléments. Les gros fichiers s'ouvrent plus vite et utilisent moins de mémoire ; les données sont lues lorsqu'un signal ou une image est prévisualisé ou importé
- L'explorateur HDF5 reste réactif pendant la lecture d'un aperçu, et les éléments sélectionnés entre-temps ne sont pas lus ; la sélection d'un élément qui ne peut pas être prévisualisé (un groupe, par exemple) efface l'aperçu. Les tableaux 1D ordinaires et les images 2D, les objets NeXus et décrits par un profil au-delà de 2^{24} valeurs (environ 16,8 millions) sont prévisualisés avec un pas le long de chaque axe (les axes des images NeXus sont encore lus entièrement, pour les vérifier comme à l'import) ; les jeux de données composites et les signaux stockés dans des tableaux 2D utilisent encore des objets complets pour leurs aperçus graphiques
- Les jeux de données à trois dimensions ou plus (piles d'images, cubes spectraux…) sont désormais listés dans le navigateur HDF5 et importés sous la forme d'une image ou d'un signal par trame : un sélecteur de vue choisit les axes affichés, les trames (première, dernière et pas selon un autre axe) et l'indice des autres axes, affiche la première trame en aperçu, et importe les trames une fois Importer ces trames coché (avec une confirmation au-delà de 100 trames). Les trames des jeux de données NeXus conservent leurs axes, et leur titre donne leur position (par exemple la position du moteur pour chaque image de détecteur d'un scan)
- Les attributs de jeux de données de plus de 10 000 valeurs ne sont plus copiés dans les métadonnées des objets importés

Le guide de l'explorateur HDF5 décrit l'import d'objets scientifiques, l'origine expérimentale et les limites des aperçus.

Scripts, macros et plugins :

- Un objet ou un groupe recherché par son titre, depuis une macro, un plugin, un client distant (par exemple `get_object("My signal")`) ou l'API Web, n'est plus silencieusement remplacé par le premier trouvé lorsque plusieurs objets ou groupes portent ce titre (par exemple des fichiers importés deux fois, des objets « Untitled ») : une erreur indique désormais les objets correspondants avec leur UUID, qui les identifie sans ambiguïté (HTTP 409 pour l'API Web)
- Pour les développeurs de plugins : `patch_title_with_ids` ne prend plus de troisième argument (`id_func`), et les fonctions des anciens identifiants de type `s001` (`SHORT_ID_REGEX`, `find_by_short_id`, `find_short_ids_in_title`, `replace_short_ids_by_uuids_in_titles`, `replace_uuids_by_short_ids_in_titles`) sont retirées de `datalab.objectmodel`, les titres désignant désormais les objets par leur UUID
- Pour les développeurs de contrôleurs DataLab personnalisés dérivés de `BaseProxy` : les nouvelles méthodes `export_capsule` et `open_capsule` doivent être implémentées

🛠 Corrections de bugs

Performances avec de nombreux objets :

- Correction du ralentissement progressif des traitements à mesure que les signaux ou les images s'accumulent dans un panneau : chaque nouveau résultat réécrivait les titres de tous les objets existants, reconstruisait toute l'arborescence des objets et mesurait de nouveau chaque titre, si bien que créer 1000 signaux l'un après l'autre prenait environ 39 minutes ; cela prend désormais environ 3 minutes (une étape de traitement avec 1000 objets dans le panneau est passée de 2,3 s à environ 0,3 s)
- Les panneaux d'images ne ralentissent plus à cause des courbes de coupe conservées pour les images masquées (nécessite PlotPy ≥ 2.11.2)

— Correction de l'altération des titres faisant référence à des objets numérotés 1000 ou plus (par exemple `s1000`) lors de l'ajout ou du réordonnancement d'objets

Disposition de la fenêtre :

— Correction de la largeur des panneaux Signal/Image non restaurée au démarrage : après avoir réduit les panneaux puis redémarré DataLab, ils réapparaissaient plus larges (jusqu'à leur largeur maximale) au lieu de conserver la largeur choisie par l'utilisateur

— Correction de l'échec du démarrage de DataLab lorsque son fichier de configuration contenait une disposition de fenêtre ou un nom de fichier journal invalide (par exemple après une modification manuelle du fichier) : ces paramètres sont désormais ignorés, et les paramètres par défaut sont utilisés

Console :

— Correction de la console qui restait affichée en cas d'erreur après la désactivation de Afficher la console en cas d'erreur dans la boîte de dialogue des préférences (avec Qt 5, lorsque ce paramètre était activé au démarrage), jusqu'au redémarrage de DataLab

Démo :

— Correction de l'échec de la démo, avec l'erreur « No module named "pytest" », dans la version autonome de DataLab (installée avec l'installeur MSI) et dans les installations sans les dépendances de test (par exemple `pip install datalab-platform`), qu'elle soit lancée depuis le menu **?**, la page d'accueil, la dernière étape de la visite guidée ou la commande `datalab-demo` (corrige l'issue #365)

— La commande `datalab-demo` ne supprime plus la configuration DataLab de l'utilisateur : la démo s'exécute désormais avec les paramètres par défaut, sans lire ni modifier les paramètres de l'utilisateur

Version autonome :

— Correction de la version autonome de DataLab qui s'arrêtait avec un code de succès lorsqu'elle ne parvenait pas à démarrer : elle s'arrête désormais avec un code d'erreur, afin que les scripts et outils de déploiement qui la lancent puissent détecter l'échec (la cause est toujours écrite dans `datalab_error.log`, dans le dossier courant)

Mise à l'échelle de l'affichage :

— Correction du flou de l'écran de démarrage, du logo DataLab de la page d'accueil et de la visite guidée, et du filigrane des graphiques vides avec une mise à l'échelle de l'affichage Windows supérieure à 100 % (par exemple 125 %) : ils sont désormais affichés à la résolution de l'écran

— Correction de la majeure partie du texte de l'interface (page d'accueil, propriétés des objets, onglets des panneaux, boîtes de dialogue, …) qui conservait sa taille à 100 % avec une mise à l'échelle de l'affichage Windows supérieure à 100 % (par exemple 125 %), alors que les menus et la barre d'état étaient agrandis. Tout le texte suit désormais la mise à l'échelle de l'affichage et le réglage de la taille du texte de Windows, et utilise la police standard des applications Windows (« Segoe UI »). Cela concernait DataLab exécuté avec Qt 5, comme l'application autonome (nécessite guidata ≥ 3.16.1)

— Correction de la ligne de l'axe des X des graphiques de signaux et d'images, qui n'était pas tracée avec une mise à l'échelle de l'affichage de 125 % ou 150 % par exemple, selon la taille et la position du graphique dans la fenêtre (nécessite PythonQwt ≥ 0.17.1)

— Correction des images affichées avec une résolution inférieure à celle de l'écran avec une mise à l'échelle de l'affichage supérieure à 100 % : avec une mise à l'échelle de 200 %, les images étaient affichées avec la moitié de la résolution de l'écran, et avec 125 %, une ligne et une colonne de pixels de l'écran sur cinq étaient dupliquées, ce qui rendait irréguliers les détails fins (par exemple le bruit). Les images sont désormais affichées à la résolution de l'écran (nécessite PlotPy ≥ 2.12.1)

— Correction de l'icône rouge des plugins dans la barre d'état (affichée lorsque des plugins n'ont pas pu être chargés), qui était trop petite et décalée vers le coin supérieur gauche avec une mise à l'échelle de l'affichage supérieure à 100 % (par exemple réduite de moitié avec 200 %)

Thème sombre :

— Correction de la plupart des fenêtres qui conservaient une barre de titre blanche avec le thème sombre sous Windows, à côté de leur contenu sombre : l'historique des conversations de l'Assistant IA, la capsule d'espace de travail, la palette de commandes et d'autres boîtes de dialogue, l'assistant d'importation, les boîtes de confirmation et de saisie (par exemple lors du renommage d'un groupe ou d'une conversation) et les panneaux détachés de la fenêtre principale (vues signal et image, panneaux macro, historique, Assistant IA et accueil, console). Toutes les fenêtres ont désormais une barre de titre sombre avec le thème sombre, y compris après un changement du mode de couleur dans la boîte de dialogue des préférences (nécessite guidata ≥ 3.17.2, corrige Issue #490)

Filigrane des graphiques :

— Correction du filigrane des graphiques vides, qui était lu dans une autre installation de DataLab partageant la même configuration utilisateur (par exemple DataLab installé dans un autre environnement Python) : le filigrane disparaissait une fois cette autre installation supprimée, et restait flou sur les écrans haute résolution lorsqu'il s'agissait d'une version plus ancienne. Le filigrane du DataLab en cours d'exécution est désormais toujours utilisé

Affichage des signaux :

— Correction du sous-échantillonnage automatique des signaux (Préférences > Visualisation > Signaux), qui ne commençait qu'au-delà de 5 fois le nombre maximal de points configuré : avec la valeur par défaut de 100 000 points, les signaux de quelques centaines de milliers de points étaient tracés en entier, ce qui ralentissait l'affichage. Les signaux comportant plus de points que ce maximum sont désormais sous-échantillonnés de sorte qu'au plus ce nombre de points soit tracé, et la boîte de dialogue des préférences ainsi que la documentation indiquent désormais la valeur par défaut réelle (100 000 au lieu de 10 000) (corrige Issue #455)

— Correction des modifications des préférences de sous-échantillonnage automatique et d'épaisseur de trait qui ne s'appliquaient qu'aux courbes des signaux sélectionnés : les autres courbes conservaient les anciennes préférences, même une fois sélectionnées à nouveau. Ces préférences s'appliquent désormais à toutes les courbes dès la fermeture de la boîte de dialogue des préférences

— Correction des graduations de l'axe X qui étaient décalées par rapport aux lignes verticales de la grille et aux courbes après l'ouverture d'un signal ou un changement de sélection, pour les signaux dont l'axe X ne commence et ne se termine pas sur une graduation (par exemple, `step.npy` des données de test de Sigima) : le décalage persistait jusqu'à un déplacement ou un zoom de la vue. L'axe X reste désormais aligné avec la grille et les courbes (corrige Issue #457)

— Correction des courbes des espaces de travail rechargés qui conservaient l'épaisseur de trait qu'elles avaient lors de l'enregistrement de l'espace de travail : les modifications de la préférence d'épaisseur de trait (Préférences > Visualisation > Signaux) ne s'y appliquaient plus, et les traits épais n'étaient plus tracés avec une largeur de 1 pixel pour les signaux comportant de nombreux points, ce qui ralentissait leur affichage. Seules les épaisseurs de trait choisies pour une courbe (par exemple dans sa boîte de dialogue de paramètres) sont désormais enregistrées, et elles sont elles aussi tracées avec une largeur de 1 pixel pour les signaux comportant de nombreux points. À l'ouverture des espaces de travail et des fichiers d'historique enregistrés avec DataLab 1.3 ou antérieur, qui enregistraient l'épaisseur de trait de chaque courbe, les épaisseurs de trait de 1 sont supprimées afin que les courbes suivent la préférence d'épaisseur de trait ; les autres épaisseurs de trait sont conservées

Palettes de couleurs personnalisées :

— Correction des palettes de couleurs personnalisées (créées avec le gestionnaire de palettes de couleurs) qui étaient enregistrées en dehors du répertoire de configuration de DataLab, dans un répertoire `.plotpy` ou `.none` du répertoire de configuration de l'utilisateur partagé avec les autres applications basées sur PlotPy, selon la manière dont DataLab avait été lancé. Elles sont désormais enregistrées dans le sous-répertoire `plotpy` du répertoire de configuration de DataLab ; la première fois, les palettes de couleurs personnalisées enregistrées dans les anciens répertoires y sont copiées (nécessite PlotPy ≥ 2.12.0)

Import HDF5 :

— Correction de l'écrêtage des images contenant des valeurs entières non signées sur 32 bits supérieures à 2 147 483 647 lors de leur import depuis un fichier HDF5 (avertissement « Clipping uint32 data to int32 ») : ces images sont désormais importées en flottants sur 64 bits en conservant chaque valeur exactement. Les autres images entières qu'aucun type entier pris en charge ne peut contenir utilisent aussi, lorsque c'est possible, un type flottant suffisamment précis, au lieu d'être arrondies en flottants sur 32 bits au-delà de 16 777 216 ; un avertissement est émis lorsqu'aucun type pris en charge ne garantit une conversion exacte (notamment au-delà de $\pm 2^{53}$ pour float64)

— Correction de la perte de la partie imaginaire des jeux de données 1D complexes en simple précision (`complex64`) lors de leur import depuis un fichier HDF5 : ils sont désormais importés comme des signaux complexes. Les signaux entiers au-delà de $\pm 2^{53}$ émettent désormais aussi l'avertissement de précision au lieu d'être arrondis silencieusement

— Correction de l'import de plusieurs jeux de données depuis des fichiers HDF5, qui s'arrêtait au premier jeu de données impossible à importer : les autres jeux de données sont désormais importés, et ceux qui n'ont pas pu l'être sont listés dans un message unique à la fin

— Correction de l'échec d'ouverture dans l'explorateur HDF5 des fichiers HDF5 contenant des liens rompus, par exemple les fichiers maîtres de synchrotron dont les fichiers de détecteur ne sont pas disponibles (liens externes vers un fichier absent) : les liens rompus sont désormais ignorés et le reste du fichier peut être exploré et importé

— Correction des fichiers HDF5 contenant un groupe lié dans lui-même ou dans l'un de ses sous-groupes, qui ne

pouvaient pas être ouverts dans l'explorateur HDF5 (le fichier était parcouru sans fin jusqu'à une erreur) : ces liens vers un groupe parent sont désormais ignorés, tandis que les autres alias d'un groupe ou d'un jeu de données restent affichés à chacun de leurs chemins
- Correction des jeux de données stockés dans d'autres fichiers, comme les données de détecteur des fichiers maîtres de synchrotron, qui n'apparaissaient pas dans l'explorateur HDF5 même lorsque ces fichiers sont disponibles : les jeux de données atteints par un lien externe étaient recherchés à leur chemin dans leur propre fichier, si bien qu'ils étaient ignorés, ou que les données d'un autre jeu de données du fichier ouvert, situé à ce chemin, étaient importées à leur place. Ils sont désormais listés et importés à leur chemin dans le fichier ouvert
- Correction de l'échec d'ouverture des fichiers HDF5 dont le nom contient une virgule : la virgule était prise pour le séparateur entre un nom de fichier et le nom d'un jeu de données à importer. Lors de l'ajout d'un tel espace de travail, la question proposant de vider l'espace de travail n'indiquait pas non plus que sa provenance n'est pas importée, et un ajout annulé par l'utilisateur était tout de même enregistré dans le panneau Historique
- Correction des jeux de données contenant une seule ligne de valeurs (par exemple, un scan unique stocké sous forme de tableau 1 × N), qui étaient ignorés sans message lors de leur import depuis un fichier HDF5 : ils sont désormais importés et prévisualisés comme des signaux
- Correction d'une erreur lors d'un clic sur Afficher le tableau dans l'explorateur HDF5 après la sélection d'un groupe au clavier, ou après la fermeture du fichier du jeu de données sélectionné : le bouton n'affiche désormais que des tableaux, et il est désactivé une fois le fichier fermé

Enregistrement de l'espace de travail :
- Correction d'un enregistrement en échec qui détruisait la version précédente du fichier d'espace de travail : le fichier était vidé avant l'écriture des données, si bien qu'une erreur pendant l'enregistrement, un disque plein ou une interruption ne laissaient sur le disque ni l'ancien ni le nouvel espace de travail. Les espaces de travail, les capsules et les fichiers d'historique (`.dlhist`) sont désormais écrits dans un fichier temporaire, qui ne remplace le fichier précédent qu'une fois entièrement écrit ; en cas d'échec de l'enregistrement, le fichier précédent reste intact

Import de fichiers :
- Correction de l'ouverture des fichiers TIFF compressés (compression LZW, JPEG ou WebP), qui échouait avec l'erreur « requires the "imagecodecs" package » ; LZW est la compression par défaut de nombreux outils d'acquisition et d'édition d'images. Ces fichiers s'ouvrent désormais sans installation supplémentaire, y compris avec l'installeur autonome. Les images binaires compressées en CCITT sont également prises en charge avec l'installeur autonome et avec Python 3.11 ou ultérieur (corrige l'issue #381)

Métadonnées :
- L'ajout, le collage ou la suppression de métadonnées marque désormais l'espace de travail comme modifié : DataLab propose donc d'enregistrer ces changements avant de se fermer
- L'import de métadonnées depuis un fichier dans un objet (Édition > Métadonnées > Importer les métadonnées) ne lui donne plus l'identité de l'objet dont les métadonnées avaient été exportées : deux objets partageant une identité perturbaient les références aux sources dans les titres, Sélectionner les objets source et l'enregistrement de l'espace de travail

Régions d'intérêt :
- Correction de la perte de l'option Inverser la logique de la ROI des ROI d'image lors de la validation de l'éditeur de ROI : l'option restait sans effet lorsqu'elle était définie dans la boîte de dialogue de coordonnées de l'éditeur (par exemple ROI rectangulaire), et les ROI d'image créées avec cette option (par exemple par une macro) la perdaient une fois modifiées
- Correction des ROI définies en indices de pixels (par exemple par une macro ou par le contrôle à distance), qui étaient converties en coordonnées physiques lors de la validation de l'éditeur de ROI. Elles restent désormais en indices de pixels, y compris lorsqu'elles sont déplacées dans l'éditeur

Traitements :
- Les résultats de l'extraction de plusieurs régions d'intérêt dans un seul objet peuvent de nouveau être retraités : l'onglet Traitement affichait « Unknown computing feature: extract_rois » avec Réappliquer désactivé, et le rejeu de l'historique ignorait l'action
- Édition > Retraiter recalcule désormais aussi les résultats des opérations combinant plusieurs objets (par exemple la différence de deux signaux ou la moyenne de plusieurs images), qui étaient ignorés
- Correction du Traitement > Rééchantillonnage des images, qui décalait toutes les valeurs d'un demi-pixel : rééchantillonner une image avec les paramètres par défaut, c'est-à-dire sur sa propre grille, remplissait sa dernière

ligne et sa dernière colonne de NaN. Les paramètres X/Y min/max sont désormais les coordonnées des centres du premier et du dernier pixel de sortie (par défaut, ceux de l'image), et le mode "Taille de pixel" conserve exactement la taille de pixel demandée
- — Correction de la somme, de la moyenne, du produit et de l'écart-type de plusieurs signaux, ainsi que de leur combinaison en image, qui dépendaient de l'ordre des signaux : lorsqu'un signal réel venait en premier, les parties imaginaires des signaux complexes étaient perdues
- — La somme, la moyenne et le produit de plusieurs images acceptent désormais les images complexes, qui provoquaient une erreur
- — Correction du Traitement > Rééchantillonnage des images complexes avec les paramètres par défaut, et de l'"Étalonnage polynomial de l'axe Z des images complexes, qui supprimaient les parties imaginaires : les images complexes restent désormais complexes
- — Correction de la modification en place du résultat d'un retournement diagonal, horizontal ou vertical ou d'une rotation de 90° d'une image (par exemple depuis une macro ou un plugin), qui modifiait aussi l'image source
- — Correction de l'extraction de plusieurs régions d'intérêt dans une seule image, qui donnait des coordonnées erronées aux images à coordonnées non uniformes (les coordonnées X étaient prises sur les lignes extraites et les coordonnées Y sur les colonnes extraites)
- — La rotation arbitraire des images (Traitement > Géométrie > Rotation de…) refuse désormais les images à coordonnées non uniformes, comme le redimensionnement et le rééchantillonnage, au lieu de produire une image dont les coordonnées ne correspondaient pas à ses pixels
- — Correction de l'incertitude du produit de signaux, qui était perdue là où l'un des signaux est nul : elle y est désormais également propagée
- — Correction du décalage d'un pixel des ROI, des annotations et des résultats d'analyse (centroïdes, pics détectés, contours…) après une symétrie ou une rotation de 90° d'une image ; après une rotation de 90° d'une image non carrée, ou d'une image dont les coordonnées ne commencent pas à zéro, les résultats d'analyse se retrouvaient même loin des éléments qu'ils décrivent. Ils se déplacent désormais exactement avec les pixels ; pour les images à coordonnées non uniformes, les cercles, ellipses et polygones sont déplacés de façon approchée lorsque les pixels sont inversés le long d'un axe non uniforme (un cercle ne reste pas un cercle). Les ROI définies en indices de pixels sont aussi correctement déplacées, et les ROI inverses (qui masquent l'intérieur de leur forme) ne deviennent plus des ROI normales après ces opérations, une translation ou une distribution d'images sur une grille
- — Correction de ROI > Extraire…, qui donnait à l'image extraite une origine erronée lorsque les coordonnées de l'image source ne commencent pas à zéro, et des coordonnées ne correspondant pas aux pixels extraits pour les images à coordonnées non uniformes
- — Correction des coordonnées des résultats d'analyse calculés dans une ROI, qui pouvaient être fausses jusqu'à un demi-pixel
- — Correction des résultats d'analyse des images à coordonnées non uniformes (Analyse > Barycentre…, Centre du cercle minimum…, Détection de pics 2D, Détection de contours…, Transformée de Hough circulaire… et Détection de taches), qui étaient donnés en indices de pixels, ou selon l'ancienne taille de pixel et l'ancienne origine d'une image passée en coordonnées non uniformes, de sorte qu'ils étaient tracés loin des éléments qu'ils décrivent : ils sont désormais placés aux positions des pixels sur les axes de l'image, les rayons étant mis à l'échelle de la taille locale des pixels. Les résultats des images à coordonnées uniformes sont inchangés
- — ROI > Extraire… signale désormais une erreur explicite pour une région d'intérêt ne contenant aucun pixel de l'image (par exemple une cellule d'une grille de ROI plus étroite qu'un pixel, ou une région hors de l'image) au lieu de créer silencieusement une image vide ; les autres régions d'intérêt sont tout de même extraites
- — Correction de Traitement > Ajustement de niveau > Soustraction d'offset, qui utilisait une zone de fond tronquée à des coordonnées entières : la zone était erronée pour les images dont les coordonnées ne sont pas en pixels (par exemple en millimètres), et vide lorsqu'elle mesurait moins d'une unité

Plugins :

- — Les plugins organisés en dossier, comme ceux créés par `datalab-plugin create --project`, sont désormais chargés lorsque leur dossier est ajouté aux répertoires de plugins, sans devoir être installés au préalable : ils étaient ignorés sans message, sauf si une installation antérieure avait laissé des informations de paquet dans le projet. Le dossier du projet lui-même peut désormais être ajouté, au lieu de son sous-dossier `src` (corrige l'issue #411)
- — Correction de petites fenêtres vides qui apparaissaient brièvement à l'écran à l'ouverture de Plugins > Configurer les plugins…, juste avant l'affichage de la boîte de dialogue (corrige l'issue #486)

Plugins et historique :

— Le rejeu d'une action de l'historique ou le recalcul d'un résultat créé par une fonctionnalité de plugin portant le même nom qu'une fonctionnalité intégrée exécute désormais la fonctionnalité du plugin au lieu de la fonctionnalité intégrée

Panneau d'historique :

— Correction d'un avertissement démesuré qui s'affichait lors de la suppression d'un groupe de signaux ou d'images dont les résultats de traitement étaient conservés dans un autre groupe, même lorsque le panneau d'historique n'enregistrait pas : il listait chaque objet supprimé, souvent sous la forme d'un identifiant illisible, et indiquait que des « opérations n'avaient pas pu être reconnectées » alors qu'il n'y avait rien à reconnecter. La suppression de tels objets n'affiche plus aucun message, et les résultats qui en dépendent restent inchangés, comme auparavant. L'avertissement n'est désormais affiché que lorsqu'une étape de traitement enregistrée perd sa source, et il liste chaque étape une seule fois, dans un message de taille limitée (corrige l'issue #451)

— Correction de la colonne Description du panneau d'historique qui affichait le contenu brut des paramètres volumineux, par exemple l'histogramme complet d'une action Luminosité et contraste : la ligne prenait une hauteur démesurée et le défilement du panneau devenait erratique. La description affiche désormais les paramètres réels (par exemple le minimum et le maximum de la plage de luminosité et de contraste) sur une seule ligne, raccourcie à la largeur de la colonne, le résumé complet figurant dans son info-bulle ; les valeurs longues sont raccourcies, et les détails dépliés défilent dans une ligne de hauteur limitée (corrige l'issue #369)

Titres des objets :

— Correction des légendes des graphiques qui affichaient toujours le titre généré d'origine après le renommage d'un objet, par exemple un profil extrait d'une image (corrige Issue #323)

— Correction des références aux sources illisibles dans les titres lorsque l'objet est sélectionné dans l'arborescence signal ou image, notamment sous Linux (corrige Issue #358)

— Correction des titres de résultats qui désignaient de mauvais objets sources après l'ajout d'un espace de travail HDF5 à une session non vide ; ajouter deux fois le même espace de travail conserve désormais aussi des chaînes de traitement importées indépendantes, de sorte que Retraiter et Sélectionner les objets source utilisent les sources issues du même import (corrige Issue #357)

Ajustement de courbes :

— Correction de l'ajustement planckien automatique (Traitement > Ajustement > Ajustement planckien), qui renvoyait une courbe environ 150 fois plus basse que le signal sur des données en unités physiques, comme un signal de corps noir avec des longueurs d'onde en mètres ; la fenêtre d'ajustement planckien interactif part également d'une courbe correspondant au pic du signal (corrige l'issue #400)

— Correction de l'ajustement exponentiel, qui échouait sur des constantes de temps courtes selon l'unité de X, par exemple une constante de temps de 1 ms avec X en secondes (ajustements automatique et interactif)

Création de signaux :

— Le signal de corps noir (Création > Corps noir (loi de Planck)) a désormais des titres et des unités d'axes (longueur d'onde en m, luminance énergétique spectrale en $W \cdot sr^{-1} \cdot m^{-3}$), qui étaient absents (corrige l'issue #400)

Contrôle à distance :

— Correction des fichiers chargés à distance avec `load_from_files` ou `load_from_directory` qui étaient parfois ignorés silencieusement, ou encore en cours de chargement après le retour de l'appel, lorsqu'ils étaient demandés juste après le démarrage de DataLab : les requêtes reçues pendant le démarrage de DataLab sont désormais traitées dès que sa fenêtre principale est prête (corrige l'issue #398)

— Les erreurs levées par les commandes à distance, comme `calc` avec une fonction de calcul inconnue ou `set_object` avec un objet inconnu, sont désormais signalées au client au lieu d'être ignorées silencieusement

— Correction des commandes à distance qui renvoyaient parfois le résultat de la commande précédente au lieu du leur : par exemple, `get_current_object_uuid` pouvait renvoyer un nom de panneau au lieu de `None` en l'absence d'objet courant

— Correction du blocage de DataLab lorsqu'il était fermé alors qu'une commande à distance était encore en cours d'exécution (par exemple en fermant la fenêtre principale avec `call_method("close")`)

— Correction du plantage de DataLab, ou de l'ouverture d'une boîte de message d'erreur, lorsqu'une requête d'un client distant ou d'une macro échouait, par exemple un calcul demandé avec `calc` qui échouait sur le signal sélectionné, ou un fichier qui ne pouvait pas être lu avec `load_from_files` : le client n'était pas informé de l'erreur, et DataLab plantait lorsqu'il s'exécutait sans affichage (par exemple en arrière-plan, pour des tests ou de l'automatisation). L'erreur est désormais signalée au client, sous la forme d'une `xmlrpc.client.Fault` avec XML-RPC ou d'une erreur HTTP avec l'API Web (500 pour un calcul en échec, 404 pour une fonction de calcul

inconnue), et affichée dans la barre d'état de DataLab ; cela s'applique aussi aux jeux de données d'un fichier HDF5 qui n'ont pas pu être importés. Les macros et les clients distants reçoivent désormais une exception là où ils poursuivaient leur exécution après la fermeture de la boîte de message (corrige l'issue #404)
- Correction de l'échec de toutes les requêtes de l'API Web au bout de 30 secondes, avec une erreur HTTP 500 (« Main thread execution timed out »), lorsque le serveur de l'API Web avait été démarré par un client distant avec `start_webapi_server` (XML-RPC), y compris pour les requêtes qui auraient dû échouer immédiatement, comme `calc` avec une fonction de calcul inconnue ; l'opération demandée pouvait même être effectuée plus tard, après le renvoi de l'erreur. Les requêtes sont désormais servies comme lorsque le serveur est démarré depuis le menu Fichier > API Web (corrige l'issue #403)
- Correction des images à coordonnées non uniformes qui perdaient celles-ci lors d'un transfert par l'API Web (`GET` et `PUT /objects/{name}/data`) : elles étaient reconstruites avec des coordonnées uniformes par défaut. Le format de données NPZ inclut désormais ces coordonnées ; les archives écrites par les versions antérieures restent lisibles. Les clients disposant de leur propre lecteur NPZ, comme DataLab-Kernel, doivent prendre en charge les nouvelles entrées pour en bénéficier

Assistant IA :
- Correction de l'absence du lien Enregistrer dans les macros lorsqu'une conversation est rouverte depuis Historique… : une macro proposée par l'assistant lors d'une session précédente ne pouvait plus être enregistrée. Les conversations rouvertes affichaient aussi tous les appels d'outils comme réussis, y compris ceux en échec ; elles affichent désormais leur résultat réel (corrige l'issue #447)
- L'enregistrement d'une macro proposée par l'assistant propose désormais un lien Afficher dans le panneau de macros : le panneau de macros étant regroupé en onglets avec l'assistant IA, la macro enregistrée restait cachée derrière la conversation
- Correction de l'assistant qui répondait à une demande d'"écriture de macro en plaçant le code dans sa réponse, d'où il ne pouvait pas être enregistré dans le panneau de macros : il propose désormais la macro sans l'exécuter (la boîte de dialogue de confirmation indique qu'elle ne sera pas exécutée) et offre le lien Enregistrer dans les macros. Un lien Enregistrer dans les macros est également proposé sous chaque bloc de code Python de ses réponses, sauf si la création de macros est désactivée

Sécurité :
- Ouvrir un espace de travail reçu d'une autre personne ne permet plus de faire charger ou exécuter par DataLab un autre code installé sur l'ordinateur. Les paramètres enregistrés avec les objets, les résultats d'analyse et les actions de l'historique nomment leur classe, et DataLab importait le module ainsi nommé puis appelait l'objet désigné, ce qu'un fichier modifié pouvait exploiter, par exemple pour faire quitter DataLab. Seules les classes de paramètres de DataLab, Sigima, guidata et des plugins installés sont désormais utilisées ; les autres paramètres enregistrés sont signalés comme illisibles (corrige l'issue #407)

5.2 Version 1.3

5.2.1 DataLab Version 1.3.1

🔧 Améliorations depuis la version 1.3.0

Compatibilité :
- Mise à jour de la version minimale requise de Sigima de 1.2.0 à 1.3.0 : les images gaussienne, rampe 2D, réseau sinusoïdal, anneaux, étoile de Siemens et sinus cardinal 2D sont désormais créées en coordonnées physiques (sur leur domaine [xmin, xmax] × [ymin, ymax]) au lieu de coordonnées en pixels, sauf si la nouvelle option « Coordonnées physiques » est désactivée. Les images créées avec des versions antérieures conservent leurs coordonnées en pixels lorsqu'elles sont recréées

5.2.2 DataLab Version 1.3.0

✧ Nouvelles fonctionnalités

Panneau d'historique :

DataLab enregistre désormais vos actions et vous permet de les rejouer. Un nouveau panneau Historique suit chaque action appliquée aux signaux et aux images, de la création d'objets et du chargement de fichiers au traitement, à l'analyse et aux modifications des régions d'intérêt (implémente Issue #90).

- Sessions d'enregistrement : les actions sont regroupées en sessions, une session active par panneau. L'enregistrement est désactivé par défaut et est proposé lors de la création ou du chargement d'un objet ; les politiques de session configurables contrôlent la manière dont les objets plugin et macro sont attachés aux sessions
- Chaînes de traitement : chaque session est présentée comme une chaîne de traitement linéaire, rendant immédiatement lisible la filiation d'un résultat
- Relecture : relire une session entière, ou la parcourir action par action, pour reproduire un flux de travail sur de nouvelles données. Les ajustements de courbes interactifs et les profils d'intensité sont enregistrés comme des actions déterministes et rejoués sans rouvrir leurs boîtes de dialogue
- Édition sur place : les paramètres de traitement, d'analyse et de création d'objets peuvent être modifiés directement depuis l'historique, et les étapes en aval sont recalculées en cascade
- Duplication : dupliquer une chaîne pour explorer une variante sans toucher à l'originale
- Persistance : les historiques sont stockés dans l'espace de travail HDF5 et peuvent également être exportés et importés en tant que fichiers `.dlhist` autonomes
- Vérifications de compatibilité : avant la relecture, DataLab valide les actions enregistrées par rapport à l'état actuel de l'espace de travail et signale ce qui peut ou ne peut pas être appliqué
- La touche Suppr supprime les entrées sélectionnées dans le panneau Historique (même comportement que dans les panneaux Signal et Image), et ne s'applique que lorsque l'arborescence de l'historique a le focus
- Une boîte de dialogue de résolution s'affiche lors de la relecture d'une chaîne de traitement interrompue par des objets supprimés : choisissez « Réparer et continuer » pour élaguer les étapes cassées et rejouer les étapes valides restantes, ou annulez sans rien modifier
- Une page de documentation dédiée et un tutoriel pas à pas sont disponibles

Assistant IA intégré :

DataLab est désormais livré avec un panneau d'ancrage optionnel Assistant IA qui peut converser avec vous, inspecter l'espace de travail, créer et traiter des signaux et des images, et écrire et exécuter des macros Python (implémente Issue #316).

- Fonctionne avec n'importe quel endpoint compatible OpenAI : OpenAI, GitHub Models, Azure OpenAI, Ollama, LM Studio, etc. Un fournisseur factice est disponible pour essayer l'assistant hors ligne, sans clé API
- L'assistant peut appeler un ensemble d"outils intégrés : lister et inspecter les objets, introspecter les opérations disponibles, créer des signaux et des images synthétiques, charger des fichiers, appliquer tout traitement enregistré, déclencher des actions de plugins, et créer et exécuter des macros
- Chaque action modifiant l'espace de travail nécessite une confirmation explicite, avec un aperçu des paramètres et un aperçu du code source de la macro avec coloration syntaxique lorsque une macro est proposée. Les outils en lecture seule peuvent être approuvés automatiquement
- Les conversations sont conservées, peuvent être consultées et rechargées, et exportées en Markdown ; l'utilisation des jetons et la taille du contexte sont affichées
- Les paramètres (fournisseur, modèle, clé API, URL de base, température, délai d'attente, limite d'itérations, approbation automatique) sont regroupés dans un onglet dédié Assistant IA de la boîte de dialogue des préférences, avec un bouton de test de connexion
- La clé API est stockée en clair dans le fichier de configuration de DataLab, et la boîte de dialogue des paramètres l'indique explicitement

Palette de commandes :

- Une palette de commandes de style VSCode vous permet de rechercher et d'exécuter n'importe quelle commande de menu en tapant une partie de son nom ou de son chemin de menu, par exemple « Traitement ▸ Analyse de Fourier ▸ FFT » (implémente Issue #337)
- Trois points d'entrée : une zone de recherche dans le coin supérieur droit de la barre de menu, une entrée « Palette de commandes… » en haut du menu Aide, et le raccourci clavier `Ctrl+Shift+P`

— La correspondance approximative classe les résultats, donc quelques caractères suffisent pour atteindre la bonne commande, avec une navigation complète au clavier

Support de l'analyse spectrale :

Suite au travail d'analyse spectrale réalisé dans Sigima, DataLab bénéficie de l'interface correspondante (implémente Issue #309).

— Nouvelle fonctionnalité Analyse > Extraire les positions des pics, stockant les pics détectés sous forme de tableau de marqueurs XY rendu sous forme de croix sur le graphique
— Nouvelle boîte de dialogue interactive Analyse > Détection de pics…, pour ajuster visuellement le seuil et la distance minimale avant de stocker le résultat
— Nouvelle fonctionnalité Opérations > Créer un signal à partir d'un tableau de marqueurs…, construisant un signal en bâtons à partir d'un tableau de marqueurs XY — pratique pour superposer des lignes de référence sur un spectre
— Les résultats des marqueurs peuvent être étiquetés dans les tableaux de résultats (`#1`, `#2`, … pour les marqueurs XY, `a`, `b`, `c`, … pour les marqueurs d'axe), de sorte que chaque ligne peut être associée au marqueur correspondant tracé sur le graphique. Cela est contrôlé par une nouvelle option « Afficher les étiquettes des marqueurs dans les tableaux de résultats » dans les paramètres
— Les étiquettes de résultat prennent désormais en charge une position et un décalage personnalisables

Régions d'intérêt du signal tracées le long de la courbe :

— Les ROI du signal ne sont plus tracées sous forme d'un seul rectangle jaune pleine hauteur : le remplissage translucide est désormais découpé selon la courbe du signal avec une ligne de base à `y = 0`, et chaque ROI utilise une couleur différente tirée d'une palette cyclique de dix couleurs (implémente Issue #314)
— Les couleurs sont dérivées de l'indice de la ROI, de sorte qu'une ROI donnée conserve la même couleur malgré les suppressions, les réorganisations et la réouverture de l'éditeur
— Les échantillons non finis sont filtrés avant le tracé, de sorte que les signaux avec des lacunes sont rendus proprement ; sur un axe Y logarithmique, la ligne de base revient au bas de la zone de dessin
— La persistance des ROI reste inchangée : seules les coordonnées sont stockées, les propriétés visuelles sont recalculées au moment du rendu

Régions d'intérêt issues de la détection (masque automatique) :

— La détection de contours, de blobs, de pics et de cercles de Hough peut créer des régions d'intérêt directement à partir des caractéristiques détectées, transformant les contours d'intensité en ROI polygonales éditables utilisables par toutes les fonctionnalités de traitement prenant en charge les ROI (implémente Issue #294)

Découverte des plugins tiers via une variable d'environnement :

— Ajout de la prise en charge de la variable d'environnement `DATALAB_PLUGINS`, permettant de spécifier un ou plusieurs répertoires comme chemins de recherche supplémentaires pour les plugins
— Plusieurs répertoires peuvent être listés en utilisant le séparateur de chemin du système d'exploitation (`;` sous Windows, `:` sous Linux/macOS), selon la même convention que `PYTHONPATH`
— Les répertoires listés sont ajoutés aux chemins de recherche existants au démarrage et sont automatiquement pris en compte par le mécanisme de découverte des plugins
— Les répertoires inexistants sont ignorés silencieusement (un avertissement est consigné dans le fichier journal), ainsi une variable d'environnement obsolète sur une autre machine n'empêchera pas le démarrage de DataLab

Expérience utilisateur de gestion des plugins :

La boîte de dialogue de configuration des plugins a été repensée pour faciliter l'installation, l'inspection et le dépannage des plugins tiers (implémente Issue #302).

— Répertoires de plugins supplémentaires persistants : plusieurs répertoires supplémentaires peuvent être ajoutés, modifiés et supprimés depuis la boîte de dialogue et sont enregistrés dans la configuration de DataLab, à côté de la liste en lecture seule des répertoires par défaut
— Boutons Ouvrir le fichier et Afficher dans le dossier par plugin, pour inspecter ou modifier un script de plugin sans parcourir le système de fichiers
— Un bouton « Appliquer et recharger les plugins » et une entrée de menu Plugins > Recharger les plugins, pour prendre en compte les modifications des plugins sans redémarrer DataLab
— Un horodatage de dernier chargement affiché pour la session en cours, pour confirmer que les modifications ont été prises en compte
— Filtrage de la liste des plugins (tous / activés / désactivés / avec erreurs), un interrupteur global activer/désactiver, et une option pour masquer les avertissements de compatibilité pour les plugins hérités v0.20

— Une action Afficher dans le dossier est désormais disponible pour le fichier de configuration de DataLab, depuis le visualiseur d'installation et de configuration

Éditeur de macros :

— Récupération après plantage : les macros non enregistrées sont automatiquement sauvegardées, et DataLab propose de les récupérer au prochain démarrage
— Boîte de dialogue Macros récentes, listant les macros utilisées lors des sessions précédentes et permettant d'en rouvrir une en un seul clic
— Modèles de macros, pour commencer à partir d'un squelette fonctionnel (macro simple, traitement d'image, appel de méthode générique)
— Complétion de code et un widget chercher & remplacer dans l'éditeur de macros

Titres d'objets :

— Les identifiants courts intégrés dans les titres d'objets (par ex. `s001`, `i012`) sont désormais rendus comme des liens cliquables dans les arbres de signaux et d'images, de sorte que les objets sources d'un calcul peuvent être sélectionnés en un seul clic

Traitement de remplacement des valeurs spéciales (signal et image) :

DataLab fournit désormais une fonction de traitement dédiée « Remplacer les valeurs spéciales » qui détecte et remplace les valeurs `NaN`, `+Inf` et `-Inf` dans les signaux et les images. La fonctionnalité est disponible sous Traitement → Ajustement des niveaux → Remplacer les valeurs spéciales dans les panneaux Signal et Image.

— Chaque cible (`NaN`, `+Inf`, `-Inf`) peut être traitée indépendamment avec sa propre stratégie
— Stratégies pour les signaux : ne rien faire, remplacer par zéro / constante / minimum / maximum / moyenne / médiane, supprimer les points affectés, remplissage avant, remplissage arrière, interpolation (linéaire, spline, quadratique, cubique, PCHIP), minimum / maximum / moyenne / médiane des N voisins
— Stratégies pour les images : ne rien faire, remplacer par zéro / constante / minimum / maximum / moyenne / médiane, minimum / maximum / moyenne / médiane des N voisins
— La boîte de dialogue des paramètres affiche des badges colorés indiquant le nombre (et le pourcentage) d'échantillons `NaN`, `+Inf` et `-Inf` trouvés dans l'objet source, offrant une visibilité immédiate sur ce qui sera modifié
— Lorsqu'une stratégie de voisinage est sélectionnée, un aperçu en direct du noyau montre la forme du voisinage qui sera utilisé pour le remplacement
— Les images entières sont traitées explicitement : comme les valeurs `NaN` et infinies ne peuvent pas exister dans les données entières, la boîte de dialogue explique que l'opération n'est pas applicable et empêche un traitement accidentel, tout en préservant le type de données d'image d'origine sans conversion inutile en flottant

Modifications

DataLab repose désormais sur SigimaX :

— Toutes les parties génériques et au niveau de l'application de DataLab (squelette de la fenêtre principale, système de configuration, widgets de tracé amovibles, espace de travail et navigateur HDF5, visualiseur de journal, écran de démarrage, barre d'état, boîtes de dialogue scientifiques et adaptateurs PlotPy) ont été extraites dans une nouvelle bibliothèque réutilisable, SigimaX, et DataLab en dérive désormais au lieu de maintenir ses propres copies (implémente Issue #182)
— Il s'agit d'un refactoring interne : les flux de travail, les paramètres et les fichiers existants ne sont pas modifiés, mais cela réduit considérablement le code dupliqué et permet de créer d'autres applications scientifiques Qt sur la même base
— Par conséquent, DataLab nécessite désormais SigimaX ≥ 1.0.1, et ses versions minimales sont alignées sur celles qu'impose SigimaX : Sigima ≥ 1.2.0, guidata ≥ 3.15.0 et PlotPy ≥ 2.11.0

Système de configuration :

— La configuration de DataLab a été migrée vers le système d'options typées fourni par SigimaX : les options ont désormais un type explicite, une valeur par défaut et une description, et sont validées lors de la lecture ou de l'écriture
— Les paramètres enregistrés par les versions précédentes sont migrés automatiquement au premier démarrage ; la disposition du fichier de configuration est sinon préservée

Package `cdlclient` archivé :

— Le package autonome `cdlclient` est obsolète et archivé. Son client XML-RPC léger est désormais fourni par Sigima, sous la forme de `sigima.client.SimpleRemoteProxy` (implémente Issue #183)

— Ses widgets Qt compagnons ont été repris par SigimaX : la boîte de dialogue de progression de connexion est désormais `sigimax.widgets.connection.ConnectionDialog`, utilisable avec tout proxy distant exposant une méthode `connect()` bloquante
— La documentation ne fait plus référence à `cdlclient`

Les outils de détection préservent désormais les régions d'intérêt existantes :

— Les régions d'intérêt créées par la détection de pics, de blobs, de cercles de Hough et de contours sont désormais ajoutées aux ROI existantes au lieu de les remplacer, sans afficher de boîte de dialogue de confirmation destructive

Les résultats d'analyse sont désormais actualisés à la demande plutôt qu'automatiquement :

— Les résultats d'analyse (statistiques, FWHM, centroïde, détection de pics/contours/blobs, etc.) ne sont plus recalculés automatiquement lorsque vous modifiez une région d'intérêt, transformez les données ou modifiez les propriétés de l'objet
— Les résultats d'analyse existants sont désormais laissés intacts après de telles modifications, évitant des effets de bord inattendus et des résultats susceptibles de devenir trompeurs lorsque les données ne correspondent plus aux paramètres d'analyse enregistrés
— La relecture ordinaire et les mutations ordinaires ne recalculent pas les analyses ; toutefois, en mode d'édition de l'historique, la modification de paramètres en amont recalcule les actions d'analyse en aval par propagation en cascade
— L'action familière « Recalculer » (menu Édition, `Ctrl+R`) actualise désormais à la fois les résultats de traitement et d'analyse, vous permettant de maîtriser pleinement le moment où les analyses sont mises à jour
— Pour les développeurs, l'ancien point d'entrée `recompute_analysis` au niveau du panneau a été renommé `recompute_selected` ; `recompute_analysis` désigne désormais une autre fonction auxiliaire au niveau du processeur, dédiée à l'actualisation explicite des analyses 1-vers-0

🛠 Corrections de bugs

Régions d'intérêt :

— Les régions d'intérêt créées par une fonction de détection (blob, contour, pic ou détection de cercle de Hough) réapparaissant immédiatement après avoir été supprimées ou modifiées — les ROI de détection sont désormais des ROI ordinaires, entièrement modifiables (corrige Issue #329)
— Les régions d'intérêt supprimées réapparaissant après un recalcul automatique en post-traitement
— Correction d'un plantage lorsque l'index de la région d'intérêt était hors limites lors de la création d'un titre de ROI

Profils d'intensité :

— Correction du bouton « Réinitialiser la sélection » des dialogues de profil d'intensité ne supprimant pas la sélection précédente : après réinitialisation et dessin d'une nouvelle région, le profil extrait correspondait toujours à la première sélection (corrige Issue #322)
— Correction du recalcul du profil utilisant les paramètres d'un autre objet lorsque plusieurs objets étaient sélectionnés

Palette de commandes :

— Correction des résultats de recherche étant non pertinents pour les requêtes courtes — taper « rota » ne renvoie plus de commandes non liées avant les entrées de rotation (corrige Issue #345)

Largeur à mi-hauteur :

— Une régression FWHM signalée par un utilisateur et observée dans DataLab v0.20.1 (une largeur aberrante, beaucoup trop petite) a été identifiée grâce à un test de non-régression basé sur le signal original, désormais inclus dans les données de test de Sigima. Le calcul était déjà correct dans la v1.2.x et reste correct dans cette version (corrige Issue #356)

Plugins :

— Correction des fonctions de traitement des plugins échouant dans les processus workers générés après un rechargement de plugin

Fenêtre principale :

— Correction du dock de tracé visible ne suivant pas l'onglet du panneau actuel dans certaines situations, en particulier lorsqu'une disposition persistante laissait le dock Macro, Historique ou Assistant IA levé au démarrage

Enregistrement de fichiers :

— Correction d'une erreur levée lors de l'enregistrement d'un signal dans un fichier dans certaines situations

Panneau d'historique :

- — Correction de la lecture des actions de chargement ne faisant rien lorsque les objets chargés avaient été supprimés alors que l'enregistrement était actif — les objets sont maintenant rechargés à partir de leurs fichiers
- — Correction des chaînes de traitement dupliquées interférant avec leur chaîne source : la lecture de session n'affecte désormais que ses propres objets
- — Correction de l'abandon de la lecture de session lorsque la session contenait une suppression d'objet enregistrée — les suppressions enregistrées qui ne peuvent plus être appliquées en toute sécurité (incohérence de l'état capturé, cibles recréées plus tôt dans la même lecture, ou cibles appartenant à une autre session) sont maintenant ignorées avec un avertissement
- — Suppression de la mise en évidence jaune pouvant persister sur les entrées de l'historique après une lecture échouée
- — Correction des étapes obsolètes n'ayant aucune indication visuelle dans l'arborescence de l'historique — les actions laissées obsolètes par un recalcul échoué ou interrompu, ou par des modifications de paramètres en attente de propagation, sont maintenant mises en évidence avec un arrière-plan ambre et une info-bulle suggérant de les relire
- — Correction des plantages potentiels ou des historiques corrompus lors du clic sur les commandes du panneau Historique (Rejouer, Supprimer, Dupliquer, …) alors qu'une longue lecture ou un recalcul était encore en cours — les commandes sont maintenant indisponibles jusqu'à la fin de l'exécution
- — Correction d'une seule entrée d'historique corrompue (par exemple, une ROI enregistrée par une version incompatible) empêchant le chargement d'un fichier `.dlhist` entier ou de l'historique d'un espace de travail HDF5 — l'action affectée se charge maintenant comme incompatible tandis que le reste de l'historique se charge normalement
- — Correction de l'arrêt de l'enregistrement de l'opération de l'utilisateur lorsqu'un objet sélectionné n'avait pas de données
- — Correction du maintien en surbrillance de la session active après le chargement d'un fichier HDF5
- — Correction du changement vers le panneau Signal lors de la suppression d'une action d'historique d'image
- — Correction d'une erreur de génération de macro (`object has no attribute uuid`) pour les chaînes de traitement 2-à-1
- — Amélioration des performances : l'ouverture de nombreux fichiers ou la relecture de longues chaînes de traitement ne bloque plus l'interface pendant que le panneau Historique vérifie à plusieurs reprises la compatibilité des actions

📖 Documentation

- — Ajout d'une page complète sur la panneau Historique et d'un tutoriel étape par étape couvrant l'enregistrement, la relecture, la modification des paramètres et l'import/export de session
- — Ajout de pages cas d'utilisation illustrant DataLab en photonique, spectroscopie et contrôle non destructif
- — Documentation de l'ancien syntaxe du sélecteur de jeux de données HDF5
- — Refonte de la mise en page de la documentation : barre de navigation plus compacte, métadonnées Open Graph et métadonnées de citation logicielle
- — Ajout d'un exemple de plugin montrant des lignes de contour iso-niveau sur une image
- — Mise à jour des traductions françaises sur toutes les nouvelles pages de documentation et celles modifiées

5.3 Version 1.2

5.3.1 DataLab Version 1.2.2

🛠 Corrections de bugs depuis la version 1.2.1

Création et ajustement de pics de signal :

- — Les boîtes de dialogue de création et d'ajustement gaussien, lorentzien et de Voigt expriment désormais l'amplitude comme la hauteur de pic signée au-dessus de la ligne de base, dans les unités Y du signal. Les objets portant des paramètres de pic basés sur l'aire historiques restent affichables et conservent leurs échantillons d'origine ; leur édition ou réévaluation nécessite désormais une conversion explicite confirmée par l'utilisateur (corrige [Issue #350](
- — Les résultats interactifs multi-gaussiens et multi-lorentziens conservent désormais chaque position de pic détectée (`x0_i`) dans les métadonnées d'ajustement canoniques, permettant d'exporter des modèles théoriques complets ou

de les réévaluer sur un autre axe X sans ajouter de contrôles de centre à la boîte de dialogue d'ajustement (corrige Issue #351)
- La fonctionnalité « Évaluer l'ajustement » est désormais disponible pour tous les ajustements interactifs, et plus seulement pour les ajustements de pics : les courbes d'ajustement linéaire, polynomial, exponentiel, sinusoïdal, CDF, planckien, double demi-gaussienne et exponentiel par morceaux peuvent maintenant être réévaluées sur un autre axe X ou exportées en tant que modèles théoriques (corrige Issue #355). Les courbes d'ajustement calculées par des versions antérieures portent des paramètres nommés dans la langue de l'interface et ne peuvent pas être réévaluées : DataLab le signale désormais explicitement et ces courbes doivent être recalculées.
- Les scripts et les clients distants peuvent migrer explicitement les métadonnées d'ajustement historiques en récupérant un objet avec `get_object()`, en convertissant une copie avec `convert_legacy_peak_fit_params()` de Sigima, puis en mettant à jour l'objet existant avec `set_object()` ; la conversion ne régénère jamais les échantillons du signal.
- Correction des curseurs de paramètres des ajustements interactifs dont les bornes excluaient des solutions atteignables : le curseur `B` de l'exponentielle ne permettait pas d'exprimer une décroissance, l'amplitude de la fonction de répartition ne permettait pas d'exprimer une transition descendante, et les bornes d'amplitude de l'exponentielle par morceaux étaient inversées pour des estimations initiales négatives (corrige Issue #352)
- Correction de la boîte de dialogue d'ajustement planckien désignant le paramètre `x0` comme la longueur d'onde du pic — le maximum du modèle se situe à environ `1,007 * x0 / sigma`, les deux ne coïncident donc que lorsque `sigma` vaut 1

Import HDF5 :
- Correction d'un plantage lors de l'exécution d'un calcul sur un objet importé depuis un fichier HDF5 contenant des attributs de référence d'objet ou de région (par ex. les tableaux `DIMENSION_LIST` / `REFERENCE_LIST` produits par `h4toh5convert`) — ces attributs ne peuvent pas être sérialisés vers les processus de calcul et sont désormais ignorés au lieu d'être copiés dans les métadonnées de l'objet (corrige Issue #346)

Assistant d'import de texte :
- Correction d'un plantage sporadique de l'application à la fermeture de l'assistant d'import de texte — l'aperçu graphique intégré est désormais libéré de manière déterministe au lieu d'être détruit plus tard par le ramasse-miettes (corrige Issue #335)

Gestion du répertoire de travail :
- DataLab ne modifie plus le répertoire de travail de son processus lors de l'ouverture ou de l'enregistrement d'un fichier. Le dernier dossier utilisé reste mémorisé pour les boîtes de dialogue de fichiers, mais le répertoire lui-même n'est plus verrouillé pendant toute la durée de vie de l'application sous Windows (il peut être déplacé, renommé ou supprimé pendant que DataLab s'exécute), et les macros ou scripts lancés depuis DataLab n'héritent plus d'un répertoire de travail inattendu

Empaquetage :
- Correction de fichiers de données manquants dans le paquet distribué : tous les fichiers `.h5` de test/démonstration et les fichiers `.template` sont désormais inclus dans le wheel et la distribution source

Documentation depuis la version 1.2.1

- Ajout d'une nouvelle page « Écosystème DataLab Platform » situant les uns par rapport aux autres DataLab (bureau), DataLab-Web (édition navigateur), DataLab-Kernel (noyau Jupyter) et Sigima (moteur de calcul) ; l'édition navigateur est désormais mise en avant sur la page d'accueil et la page de prise en main, la compatibilité web des plugins est documentée, et la feuille de route a été actualisée (interface web et noyau Jupyter livrés)
- Ajout des instructions d'installation sous NixOS : DataLab est empaqueté pour NixOS et distribué via la NGI Forge, grâce à une contribution de l'équipe Nix@NGI dans le cadre du financement de DataLab par le NGI0 Commons Fund
- Ajout d'une section « Événements » à la documentation, en commençant par les détails de la présentation EuroSciPy 2026
- Correction de la formule de création de signal polynomial, qui mentionnait un terme `y0` et un nombre quelconque de coefficients alors que les paramètres n'exposent en réalité que `a0` à `a5`, et documentation de l'ajustement sigmoïde, disponible mais non documenté (corrige Issue #353)
- Alignement des formules de modèles de signaux documentées sur la nouvelle paramétrisation en hauteur des modèles gaussien, lorentzien et de Voigt

— Actualisation des exemples de la documentation « Édition » des signaux et des images, qui affichaient encore des titres d'objets produits par des versions antérieures (corrige Issue #354)
— La visite de la racine du site de documentation redirige désormais vers la version française ou anglaise en fonction de la langue du navigateur, au lieu de toujours servir les pages en anglais
— Mise à jour des traductions françaises sur toutes les pages de documentation nouvelles et modifiées

🔧 Améliorations depuis la version 1.2.1

Compatibilité :
— Mise à jour de la version minimale requise de Sigima de 1.1.3 à 1.1.6 (corrections de calcul et paramètres de hauteur de pic versionnés introduits dans la version 1.1.6)

5.3.2 DataLab Version 1.2.1 (05/06/2026)

🛠 Corrections de bugs depuis la version 1.2.0

Menus du panneau Signal :
— Restauration de l'entrée « Étalonnage linéaire » dans le menu « Traitement > Transformation des axes » du panneau Signal — elle avait été supprimée par inadvertance lors d'une réorganisation des menus et n'était plus accessible depuis la barre de menus, alors même que le calcul sous-jacent restait disponible (corrige Issue #312)
Panneau Signal — curseur de plage Y :
— Correction de l'annotation du curseur de plage Y affichant une inégalité incorrecte (par ex. `5 < y < 2`) et un Δy négatif lorsque le curseur supérieur était glissé en dessous du curseur inférieur — l'annotation affiche désormais toujours les valeurs dans l'ordre croissant avec une largeur de plage positive (corrige Issue #306)
Panneau Image — échelle logarithmique de l'axe Z :
— Correction de l'action « Axe Z logarithmique base 10 » de la barre d'outils restant systématiquement grisée dans le panneau Image — l'outil fonctionne désormais correctement pour tous les types d'images (corrige Issue #313, corrigé en amont dans PlotPy ≥ 2.10.0)
Panneau Image — détection de contours :
— Correction de la détection de contours ellipse et cercle produisant des résultats incorrects (positions et tailles erronées) en raison de coordonnées X/Y inversées et de conversions d'unités manquantes dans le code d'ajustement de modèle scikit-image (corrige Issue #326, corrigé en amont dans Sigima ≥ 1.1.3)
Éditeur de ROI :
— Correction de la boîte de dialogue de l'éditeur de ROI ignorant les échelles d'axes non linéaires (par ex. logarithmique) du graphique source — la boîte de dialogue préserve désormais la même configuration d'échelle d'axes que le panneau principal (corrige Issue #315)
Thème et affichage :
— Correction des couleurs des marqueurs et formes des graphiques revenant aux valeurs par défaut de PlotPy (jaune) au lieu des couleurs configurées de DataLab (rouge) lorsque le mode de couleur était explicitement défini sur « clair » ou « sombre » dans les paramètres — les surcharges de couleurs personnalisées sont désormais réappliquées après chaque changement de thème (corrige Issue #297)
— Correction des éléments d'interface (texte, icônes, dispositions des boîtes de dialogue) ne s'adaptant pas correctement aux paramètres d'affichage haute résolution (High-DPI) — le rendu HTML des DataSet, les tailles d'icônes et les dimensions de la zone d'affichage s'adaptent désormais au facteur d'échelle du système (corrige Issue #317, corrigé en amont dans guidata ≥ 3.14.4)
Éditeur de propriétés d'image :
— Correction du champ de saisie de la taille de pixel (`Δx` / `Δy`) corrompant les valeurs saisies — la plupart des chiffres tapés après le premier étaient interprétés comme des décimales en raison d'une boucle de mise à jour réactive dans le widget d'édition de paramètres (corrige Issue #320, corrigé en amont dans guidata ≥ 3.14.4)
Renommage de la variable d'environnement de débogage :
— Renommage de la variable d'environnement de débogage de `DEBUG` à `DATALAB_DEBUG` — le nom générique `DEBUG` entrait en conflit avec des conventions largement utilisées par des tiers (Django, Flask, outils Node.js, systèmes CI) et pouvait réinitialiser silencieusement le fichier de configuration utilisateur au démarrage lorsqu'il était défini pour des raisons non liées. Définir `DATALAB_DEBUG=1` active désormais le mode débogage ; la variable `DEBUG` seule est ignorée (corrige Issue #319)

API de contrôle à distance :

— Exposition de `get_current_object_uuid()` sur l'API proxy, la rendant disponible via XML-RPC et l'API Web

🔧 Améliorations depuis la version 1.2.0

Compatibilité :

— Mise à jour de la version minimale requise de guidata de 3.14.3 à 3.14.4 (problème de haute résolution et de mise à l'échelle de l'écran, correction du champ de saisie DataSet, ajout de la commande CLI de compilation sécurisée)

— Mise à jour de la version minimale requise de PlotPy de 2.8.2 à 2.10.0 (correction de l'échelle logarithmique de l'axe Z, visibilité du bouton de débordement de la barre d'outils en thème sombre, optimisation PythonQwt 0.16.0)

— Mise à jour de la version minimale requise de Sigima de 1.1.2 à 1.1.3 (correction de la détection de contours ellipse et cercle)

5.3.3 DataLab Version 1.2.0 (20/04/2026)

✧ Nouvelles fonctionnalités depuis la version 1.1

Boîte de dialogue de configuration des plugins :

DataLab propose désormais une boîte de dialogue de configuration des plugins (accessible via « Plugins > Configurer les plugins… ») qui offre un contrôle total sur la gestion des plugins tiers :

— Activez ou désactivez les plugins individuels à l'aide de cases à cocher, ou basculez tous les plugins en même temps avec une case à cocher principale à trois états

— Filtrez les plugins par statut : tous, activés, désactivés ou plugins avec des erreurs d'importation

— Affichez les détails des plugins, y compris la version, l'auteur et les descriptions longues extensibles directement dans la boîte de dialogue

— Les plugins avec des erreurs d'importation sont affichés en haut avec leur trace complète, ce qui facilite le diagnostic des problèmes d'installation

— Le widget de texte extensible utilisé pour les descriptions longues calcule sa largeur préférée à partir d'un contexte de mesure fixe, garantissant une mise en page stable et un basculement fiable de « Afficher la description complète » quelles que soient les redimensionnements de la boîte de dialogue ou le rendu hors écran

Rechargement à chaud des plugins :

— Les plugins tiers peuvent désormais être rechargés à l'exécution sans redémarrer DataLab, via « Plugins > Recharger les plugins »

— L'activation ou la désactivation des plugins dans la boîte de dialogue Préférences ou la boîte de dialogue de configuration des plugins prend effet immédiatement - aucun redémarrage n'est nécessaire

— Le menu Plugins, le widget d'état et les actions des plugins sont automatiquement actualisés après les modifications de configuration

Détection de multi-instance :

DataLab détecte désormais lorsqu'une autre instance est déjà en cours d'exécution et avertit l'utilisateur avant d'ouvrir une deuxième instance :

— Utilise un mécanisme de fichier de verrouillage basé sur le PID qui prend en charge plusieurs instances simultanées (comptage de références)

— Les PID obsolètes des processus plantés sont automatiquement nettoyés

— Prise en charge multiplateforme (Windows, macOS, Linux) utilisant la détection de processus spécifique à la plateforme

— La fermeture d'une instance ne supprime plus incorrectement le fichier de verrouillage lorsque d'autres instances sont encore en cours d'exécution

Synchronisation du contraste de l'éditeur ROI d'image :

— L'éditeur ROI d'image partage désormais les paramètres de contraste (LUT) avec le panneau de l'image source

— L'ajustement du contraste dans l'éditeur ROI est reflété dans le panneau principal et vice versa

— Les contrôles de contraste sont entièrement réactivés dans la boîte de dialogue de l'éditeur ROI d'image

API de contrôle à distance — renvoyer les objets modifiés vers DataLab :

L'API proxy (XML-RPC et Web API) expose désormais une nouvelle méthode `set_object` qui met à jour un signal ou une image existant dans DataLab à partir d'une copie modifiée obtenue via `get_object` (corrige Issue #305) :

- Auparavant, les modifications des propriétés des objets (par ex. `dx`, `dy`, `x0`, `y0`, `title`) effectuées sur le résultat de `get_object` étaient perdues car `get_object` renvoie une copie — `set_object` fournit désormais un flux aller-retour propre
- Fonctionne pour les objets signal et image : les éléments de résultats calculés attachés à l'objet sont préservés lors de la mise à jour, de sorte que la mise à jour d'un `ImageObj` ne déclenche plus d'incompatibilité de type
- Le panneau des propriétés est automatiquement rafraîchi après `set_object`, de sorte que les propriétés modifiées (titre, unités, axes, incertitudes, etc.) sont immédiatement visibles dans l'interface graphique

🕮 Documentation depuis la version 1.1

- Ajout de la documentation de l'API pour le module `datalab.objectmodel`
- Ajout de captures d'écran pour la boîte de dialogue « Coller les métadonnées » (panneaux de signal et d'image)
- Mise à jour de la documentation des plugins pour décrire la nouvelle boîte de dialogue de configuration, le flux de travail de rechargement à chaud et les assistants de l'API des plugins
- Mise à jour du guide de développement des plugins tiers avec de nouvelles références de modèles et des informations sur la couverture des tests
- Enrichissement de la référence de l'API Web : documentation des options de transfert binaire (`?compress=false` pour des téléchargements NPZ non compressés plus rapides, `?overwrite=true` pour le remplacement atomique d'objets existants), du point d'entrée `PUT /objects/{name}` qui met à jour un objet sur place en préservant son identité, son groupe d'appartenance et sa position, ainsi que de la nouvelle section « Calcul » listant les points d'entrée `/select` et `/calc` permettant de piloter les calculs DataLab à distance
- Mise à jour des traductions françaises sur toutes les pages de documentation nouvelles et modifiées

🔧 Améliorations depuis la version 1.1

Compatibilité :

- Prise en charge officielle de pandas 3.0.x (contrainte de dépendance mise à jour de `< 3.0` à `< 3.1`)
- Mise à jour de la version minimale requise de Sigima de 1.1.0 à 1.1.2 pour bénéficier des dernières corrections et améliorations du moteur de calcul
- Ajout de la prise en charge héritée de la variable d'environnement `WINPYDIRBASE` pour les déploiements basés sur WinPython

Renforcement du système de plugins :

- La découverte et l'enregistrement des plugins sont désormais résilients aux échecs d'importation de tiers tout en préservant le rapport d'erreurs dans la console, les journaux et la boîte de dialogue de configuration
- Les sous-menus des plugins sont désormais défilables pour éviter le débordement lorsque de nombreuses entrées de plugins sont enregistrées

Outils de développement :

- Nouveau script `run_with_env.py` pour exécuter des tâches dans plusieurs contextes d'environnement Python (WinPython, venv, etc.)
- Simplification de la gestion des variables d'environnement en supprimant le système `DATALAB_ENV_LOADED`
- Correction de la tâche complète Coverage dans VS Code pour utiliser correctement le wrapper `run_with_env.py`

🛠 Corrections de bugs depuis la version 1.1

Espace de travail HDF5 - Sérialisation des tables :

- Correction de la non-suppression des métadonnées appelables lors de l'enregistrement/chargement HDF5 des objets `TableResult`
- Correction de la restauration incorrecte des valeurs d'énumération basées sur des chaînes dans les résultats de table après un aller-retour HDF5
- Correction de la non-survie de l'attribut `column_formats` lors de l'aller-retour HDF5 pour les sorties `TableResult` et `TableResultBuilder`

Système de plugins :

- Correction de l'`AttributeError` dans la boîte de dialogue de configuration des plugins lors du clic sur « Afficher la description complète » (référence d'attribut incorrecte)
- Correction des erreurs d'importation de plugins étant silencieusement ignorées lorsqu'elles se produisaient avant l'initialisation de la console interne

5.4 Version 1.1

5.4.1 DataLab Version 1.1.1 (08/04/2026)

> Remarque : Il s'agit d'une version de maintenance qui corrige un seul bug affectant uniquement la version exécutable autonome de DataLab pour Microsoft Windows. Si vous avez installé DataLab via `pip`, `conda`, ou si vous utilisez DataLab sur Linux ou macOS, vous n'êtes pas affecté par ce bug et il n'est pas nécessaire de mettre à jour vers cette version.

🛠 Corrections de bugs depuis la version 1.1.0

Les fonctionnalités de rotation/symétrie échouent ou se bloquent dans la version autonome :

- Correction des fonctionnalités de rotation et de symétrie d'image (« Symétrie horizontale », « Symétrie verticale », « Rotation 90° gauche/droite », « Symétrie diagonale ») qui plantaient avec un `ImportError` ou se bloquaient indéfiniment dans la version autonome Windows (build PyInstaller)
- Le problème était causé par le worker exécuté en mode multiprocessus qui devait importer la chaîne complète des modules GUI de DataLab lors du dé-pickling du wrapper de transformation géométrique, déclenchant des imports circulaires dans l'exécutable PyInstaller
- Le code de post-traitement géométrique a été déplacé vers un module autonome sans dépendances GUI, de sorte que le processus de travail ne déclenche plus la chaîne d'importation GUI de DataLab
- Cela n'était pas reproductible avec la version package Python de DataLab, uniquement avec l'exécutable autonome
- Ceci clôture Issue #304 - Les fonctionnalités de rotation/symétrie échouent ou se bloquent dans la version autonome (PyInstaller)

5.4.2 DataLab Version 1.1.0 (02/02/2026)

✧ Nouvelles fonctionnalités

API Web pour l'intégration HTTP/JSON :

DataLab propose désormais une API Web HTTP/JSON moderne comme alternative à l'interface XML-RPC existante. C'est la voie d'intégration recommandée pour les nouveaux projets, notamment pour les notebooks Jupyter et les outils web.

- Accessible via : Fichier → API Web → Démarrer le serveur API Web (ou définir la variable d'environnement `DATALAB_WEBAPI_ENABLED=1`)
- Avantages clés :
  - JSON pour les métadonnées et NPZ pour les données binaires (transfert efficace de grands tableaux)
  - Authentification par jeton Bearer pour la sécurité
  - Conventions REST avec documentation OpenAPI
  - Compatibilité WASM/Pyodide (fonctionne dans les environnements navigateur)
- Points d'accès inclus :
  - Listage, récupération, création et suppression d'objets
  - Gestion des métadonnées et transfert de données binaires (format NPZ)
  - API de calcul pour exécuter des opérations de traitement à distance
- Découverte automatique : DataLab-Kernel trouve et se connecte automatiquement à une instance DataLab en cours d'exécution sans configuration manuelle
- Sécurité : Liaison localhost par défaut, avec contournement optionnel du jeton localhost pour simplifier le développement local

Compatibilité PyQt6 :

- DataLab prend désormais entièrement en charge PyQt6 en plus de PyQt5
- Correction de la récupération de la géométrie de l'écran pour utiliser `primaryScreen()` pour la compatibilité Qt5-Qt6
- Le pipeline CI inclut désormais les tests PyQt6 pour garantir une compatibilité continue

Améliorations de l'API de contrôle à distance :

> Remarque : Ces nouvelles fonctionnalités concernent également l'API des commandes de macro, car les macros utilisent la même interface proxy que les clients de contrôle à distance.

- Ajout de la méthode `call_method()` à la classe `RemoteProxy` pour appeler n'importe quelle méthode publique sur la fenêtre principale de DataLab ou les panneaux :
  - Permet l'accès programmatique à des opérations non exposées via des méthodes proxy dédiées (par exemple, `remove_object`, `delete_all_objects`, `get_current_panel`)
  - Prend en charge la résolution automatique des méthodes : vérifie d'abord la fenêtre principale, puis le panneau actuel si la méthode n'est pas trouvée
  - Le paramètre optionnel `panel` permet de cibler un panneau spécifique (« signal » ou « image »)
  - Exécution thread-safe : les opérations GUI s'exécutent automatiquement dans le thread principal pour éviter le gel
  - Gestion appropriée des exceptions : les exceptions levées lors de l'exécution de la méthode (par exemple, tenter d'appeler des méthodes privées) sont capturées et propagées au client sous forme d'objets `xmlrpc.client.Fault` avec le message d'erreur original
  - Un nouvel exemple de macro `test_call_method.py` démontre l'utilisation : suppression d'objets, obtention du panneau actuel et opérations spécifiques au panneau
  - Étend les capacités d'automatisation pour des flux de travail de macro avancés et le contrôle d'applications externes
- Ajout de la méthode `remove_object()` à l'interface proxy (locale et distante) pour la suppression sélective d'objets
  - Supprime l'objet actuellement sélectionné du panneau actif
  - Paramètre optionnel `force` pour ignorer la boîte de dialogue de confirmation
  - Complète la méthode existante `reset_all()` qui efface l'ensemble de l'espace de travail (y compris toutes les macros)

Dépendances :

- Mise à jour de la dépendance Sigima vers la version 1.1.0 qui inclut de nouvelles fonctionnalités et corrections de bugs
- Ajout des dépendances pour l'API Web : FastAPI, Uvicorn et Pydantic (incluses dans l'installation standard)

5.5 Version 1.0

5.5.1 DataLab Version 1.0.4 (02/02/2026)

🛠 Correctifs depuis la version 1.0.3

Compatibilité du calcul du barycentre :

- Correction de l'échec du calcul du barycentre avec NumPy 2.4.0 et scikit-image 0.26.0 en raison de la nouvelle optimisation einsum qui n'accepte pas les tableaux masqués
- Les données d'image sont désormais converties explicitement en un tableau NumPy standard avant le calcul du barycentre
- Ceci clôture Issue #289 - Problème de compatibilité avec NumPy 2.4.0 et scikit-image 0.26.0

5.5.2 DataLab Version 1.0.3 (19/12/2025)

🛠 Correctifs depuis la version 1.0.2

Le chargement des images avec une seule ligne ou colonne échoue :

- Correction du chargement des fichiers d'images avec une seule ligne ou colonne (par exemple, certains fichiers de spectroscopie SIF avec une forme `(1, N)`) provoquant une `IndexError` et empêchant l'affichage de l'image
- Le problème sous-jacent se trouvait dans la fonction de conversion des coordonnées en bacs de PlotPy qui supposait au moins 2 points
- Ce correctif est inclus dans PlotPy 2.8.3

Plage LUT copiée incorrectement lors du traitement des images :

- Correction des images traitées héritant de la plage LUT (paramètres de contraste d'affichage) de l'image source, ce qui entraîne une visualisation incorrecte lorsque les valeurs de données changent de manière significative
- Lors de l'application d'opérations telles que la correction de l'offset, l'image résultante ajuste désormais automatiquement sa plage d'affichage pour s'adapter aux nouvelles valeurs de données au lieu d'utiliser la plage LUT de l'image originale
- Auparavant, les utilisateurs devaient ajuster manuellement la LUT pour voir correctement les images traitées (elles pouvaient apparaître complètement noires ou avec un contraste incorrect)
- Ceci clôture Issue #288 - Plage LUT copiée incorrectement lors du traitement des images
- Ce correctif est inclus dans Sigima 1.0.4

Fenêtres de tracé séparées - Rapport d'aspect incorrect pour les coordonnées non uniformes :

- Correction des fenêtres de tracé séparées (par ex. « Voir dans une nouvelle fenêtre », éditeurs de ROI, dialogues de profil) qui n'utilisaient pas la même configuration de rapport d'aspect que le gestionnaire de tracé intégré
- Lors de l'affichage d'images avec des coordonnées non uniformes, le tracé principal déverrouille correctement le rapport d'aspect pour un affichage approprié, mais les fenêtres séparées n'héritaient pas de ce paramètre
- Toutes les boîtes de dialogue d'image héritent désormais correctement des paramètres de rapport d'aspect, assurant un comportement d'affichage cohérent sur toutes les fenêtres de tracé
- Ceci clôture Issue #287 - Les fenêtres de graphique séparées n'utilisent pas la même configuration de rapport d'aspect que le graphique intégré

Supprimer tous les résultats - AttributeError lorsque la ROI a été supprimée :

- Correction de l'action « Supprimer tous les résultats » qui échouait avec `AttributeError: 'NoneType' object has no attribute 'get_single_roi_title'` lorsque les résultats contiennent des informations de ROI mais que la ROI a été ensuite supprimée de l'objet
- Le problème se produisait lorsque :
  1. Exécution d'une analyse avec une ROI (par ex. Centroïde sur une image avec ROI rectangulaire)
  2. Suppression de la ROI de l'objet
  3. Tentative de suppression de tous les résultats via « Analyse > Supprimer les résultats > Supprimer tous les résultats »
- Le correctif ajoute une vérification de valeur None appropriée avant d'accéder au titre de la ROI
- Ceci clôture Issue #286 - « Supprimer tous les résultats » échoue avec AttributeError lorsque la ROI a été supprimée

Recalcul automatique de l'analyse - Paramètres obsolètes après suppression des résultats :

- Correction des paramètres d'analyse qui n'étaient pas effacés lors de la suppression des résultats, ce qui pouvait provoquer un comportement de recalcul automatique inattendu lors de la modification des ROI
- Après la suppression des résultats (via « Supprimer tous les résultats » ou suppression individuelle), la modification de la ROI déclenchait le recalcul de l'analyse supprimée car les paramètres restaient dans les métadonnées de l'objet
- Le correctif assure que les paramètres d'analyse sont correctement effacés avec les résultats, empêchant le recalcul automatique indésirable
- Ceci clôture Issue #285 - Les paramètres d'analyse ne sont pas effacés après la suppression des résultats

ROI rectangulaire dessinée à l'envers cause des statistiques NaN :

- Correction des statistiques de ROI rectangulaire retournant des valeurs NaN lorsque la ROI était dessinée « à l'envers » (du coin inférieur droit vers le coin supérieur gauche au lieu du coin supérieur gauche vers le coin inférieur droit)

- Le problème est survenu parce que la conversion des coordonnées de la ROI produisait des valeurs de largeur/hauteur négatives, provoquant l'échec de la génération du masque
- La génération du masque de ROI fonctionne désormais correctement quelle que soit la direction dans laquelle le rectangle a été dessiné
- Ceci clôture Issue #284 - ROI rectangulaire dessinée à l'envers cause des statistiques NaN
- Ce correctif est inclus dans Sigima 1.0.4

Rafraîchissement du graphique - Les ROI et annotations persistent après « Tout supprimer » :

- Correction des ROI et annotations restant occasionnellement visibles après l'exécution de l'action « Tout supprimer » depuis le menu Édition
- Le problème est survenu parce que le widget de tracé n'était pas explicitement invité à se redessiner après la suppression de tous les éléments de tracé et des éléments de forme
- Le graphique se rafraîchit désormais correctement après la suppression de tous les objets, garantissant une vue propre sans nécessiter de changement de panneau
- Le bug était intermittent et difficile à reproduire systématiquement, donnant l'impression que les objets étaient « coincés » sur le graphique

Étiquettes de résultats - Faux message « omis » pour les résultats à une seule ligne :

- Correction des étiquettes de résultats (par exemple, les caractéristiques des impulsions) affichant incorrectement le message « X lignes supplémentaires omises » lors de l'affichage de résultats à une seule ligne sans troncature réelle
- L'anomalie s'est produite parce que `len(adapter.result)` était utilisé pour obtenir le nombre de lignes, mais cela renvoie en fait le nombre d'en-têtes de colonnes, pas les lignes de données
- Pour les caractéristiques des impulsions avec 13 colonnes, cela a amené l'étiquette à penser qu'il y avait 13 lignes alors qu'il n'y en avait qu'une seule, ce qui a conduit à un faux message « 12 lignes supplémentaires omises »
- Le correctif garantit un comptage précis des lignes en utilisant `len(df)` à la place, évitant ainsi les fausses notifications de troncature
- Ceci clôture Issue #281 - Faux message « omis » des étiquettes de résultats

Vue des signaux - Les éléments XRangeSelection persistent après la suppression des signaux :

- Correction des éléments XRangeSelection et DataInfoLabel (créés par les outils de statistiques de courbes de PlotPy) restant visibles sur le graphique après la suppression de tous les signaux
- Lors de l'activation de l'outil XRangeSelection depuis la barre d'outils PlotPy, puis de la suppression du ou des signaux, ces objets graphiques créés par l'outil n'étaient pas nettoyés
- La logique de nettoyage se déclenche désormais correctement lorsqu'il n'y a aucun objet sélectionné (après la suppression du dernier), et pas seulement lorsqu'un seul objet est sélectionné
- Ceci clôture Issue #280 - Les éléments XRangeSelection persistent après la suppression de tous les signaux

Contrôle distant - Plantage lors de l'ajout d'un objet via le proxy :

- Correction du plantage `AttributeError: 'NoneType' object has no attribute 'setText'` lors de l'ajout d'objets via `RemoteProxy.add_object()` dans les macros
- Le plantage se produisait dans `update_tree()` lorsque la vue arborescente se désynchronisait temporairement avec le modèle d'objets (par exemple, lors de l'ajout d'objets avec des configurations de métadonnées spécifiques ou des ROIs via le proxy distant)
- La vue arborescente se reconstruit désormais automatiquement si des incohérences sont détectées, assurant un ajout d'objets robuste dans tous les scénarios
- Ceci clôture Issue #279 - AttributeError dans update_tree() lors de l'ajout d'un objet via RemoteProxy

Icônes - Avertissements Qt SVG sous Linux :

- Correction des avertissements « qt.svg: Invalid path data; path truncated » apparaissant 4 fois au démarrage sous Linux/Ubuntu
- Nettoyage des fichiers d'icônes SVG pour supprimer les métadonnées Inkscape/Sodipodi et correction de la syntaxe des chemins dans deux fichiers d'icônes qui utilisaient une continuation implicite de commande de courbe de Bézier cubique (non prise en charge par le moteur de rendu SVG de Qt sous Linux)
- Ceci clôture Issue #278 - Correction des avertissements « qt.svg: Invalid path data; path truncated » sous Linux

Extraction de ROI - KeyError lors de l'extraction des ROI :

- Correction de l'erreur `KeyError` lors de l'extraction des ROI à partir d'images avec des ROI générées par la détection de blobs ou d'autres fonctions d'analyse

- — Le problème est survenu parce que l'éditeur de ROI essayait d'accéder à des éléments de tracé qui n'avaient pas encore été créés (par exemple, lorsque le « rafraîchissement automatique » était désactivé ou dans certains scénarios de sélection)
- — La boîte de dialogue d'extraction de ROI gère désormais correctement les éléments de tracé manquants
- — Ceci clôture Issue #276 - KeyError lors de l'extraction des ROI à partir d'images avec des résultats de détection de blobs

Développement de plugins - Boucle infinie lors de l'itération sur les objets d'un groupe :

- — Correction de la boucle infinie se produisant dans les plugins lors de l'itération sur les identifiants d'objets retournés par `ObjectGroup.get_object_ids()` tout en ajoutant simultanément de nouveaux objets au même groupe
- — La méthode retournait une référence directe à la liste interne au lieu d'une copie, ce qui faisait que l'itération incluait les objets nouvellement ajoutés et bouclait indéfiniment
- — Cela affectait particulièrement les flux de travail des plugins qui bouclent sur les signaux/images d'un groupe et ajoutent des résultats au même groupe ou à un autre groupe
- — Le correctif garantit que `get_object_ids()` retourne désormais une copie de la liste, rendant l'itération sûre même lors de la modification du groupe
- — Ceci clôture Issue #274 - Boucle infinie lors de l'itération sur les objets d'un groupe tout en ajoutant de nouveaux objets

Visibilité des marqueurs :

- — Correction des couleurs du marqueur en croix (affiché en appuyant sur la touche Alt sur le graphique) héritées de PlotPy étant trop pâles sur fond blanc, et d'autres propriétés de marqueurs explicitement configurées pour un meilleur contraste
- — Correction des couleurs d'annotation par défaut héritées de PlotPy étant trop pâles sur fond blanc - les annotations de segments et de formes glissées utilisent désormais un vert plus vif (`#00ff55`) pour une meilleure visibilité

Disposition du panneau des macros :

- — Correction du panneau de la console des macros prenant un espace vertical excessif à la première ouverture - l'éditeur de scripts obtient désormais correctement 70 % de l'espace et la console 30 % par défaut, garantissant une édition confortable sans redimensionnement manuel

Nommage des macros :

- — Correction du nommage automatique des macros générant des noms en double après le rechargement de l'espace de travail à partir de fichiers HDF5
- — Lorsque les macros étaient désérialisées à partir d'espaces de travail enregistrés ou importées à partir de fichiers, le compteur interne utilisé pour générer des noms par défaut (par exemple, « macro_01 », « macro_02 ») n'était pas mis à jour, ce qui pouvait entraîner la réutilisation d'un nom existant pour la prochaine nouvelle macro
- — Le compteur se synchronise désormais avec les noms de macros existants après le chargement, garantissant un nommage séquentiel unique
- — Les noms de macros par défaut ont été simplifiés du format traduit « Sans titre XX » au format neutre en termes de langue « macro_XX » pour une meilleure cohérence entre les locales

Exécution des macros :

- — Correction des erreurs de syntaxe lors de l'utilisation de f-strings avec des guillemets imbriqués dans les macros (par exemple, `f'text {func("arg")}'` fonctionne désormais correctement)
- — Correction des caractères Unicode corrompus dans la sortie de la console des macros sous Windows - les caractères spéciaux comme ✓, ☼ et → s'affichent désormais correctement au lieu d'afficher du texte illisible

Console interne - Opérations sur l'espace de travail HDF5 :

- — Correction du blocage de l'application lors de l'appel de `open_h5_files()` depuis la console interne - la console s'exécute dans un thread séparé et la création d'éléments d'interface Qt (boîtes de dialogue de progression) depuis des threads non principaux provoque des blocages
- — Ajout de nouvelles méthodes d'API sans interface graphique `load_h5_workspace()` et `save_h5_workspace()` qui peuvent être appelées en toute sécurité depuis la console interne ou tout contexte de script sans dépendances à l'interface graphique
- — Ces méthodes sont également disponibles via l'API de contrôle distant pour plus de cohérence
- — Note : `load_h5_workspace()` ne prend en charge que les fichiers HDF5 natifs de DataLab ; pour importer des fichiers HDF5 arbitraires, utilisez des macros avec `RemoteProxy`
- — Ceci clôture Issue #275 - Blocage de la console lors de l'appel de `open_h5_files()` depuis la console interne

Boîtes de dialogue de curseur de signal - Saisie décimale avec les paramètres régionaux :

- Correction de l'échec de la saisie de valeurs décimales dans les boîtes de dialogue de curseur de signal lors de l'utilisation de paramètres régionaux avec la virgule comme séparateur décimal (par exemple, locales française, allemande)
- Le champ de saisie Y dans les boîtes de dialogue « Première abscisse à y=… », « Ordonnée à x=… » et « Largeur totale à y=… » accepte désormais systématiquement le point comme séparateur décimal, quelle que soit la locale du système
- Ceci clôture Issue #276 - Échec de la saisie décimale avec les paramètres régionaux utilisant la virgule comme séparateur décimal

Titre du profil radial :

- Correction du suffixe en double dans le titre de l'image résultante lors de l'extraction du profil radial d'une image (par exemple, `radial_profile(i019)|center=(192.500, 192.500)|center=(192.500, 192.500)` au lieu de `radial_profile(i019)|center=(192.500, 192.500)`)
- Ce correctif est fourni par Sigima 1.0.4

ROI de grille - Paramètres d'espacement manquants pour les grilles non uniformes :

- Correction de la fonctionnalité ROI de grille ne fonctionnant pas correctement pour les images où les sous-images ne remplissent pas toute la zone de l'image (par exemple, les réseaux de points laser avec des espaces entre les points)
- Ajout des paramètres manquants `xstep` et `ystep` pour contrôler l'espacement horizontal et vertical entre les centres des ROI, en pourcentage de la largeur/hauteur de cellule calculée automatiquement (par défaut 100 % = grille uniformément répartie)
- Auparavant, la grille était toujours supposée être uniformément répartie sur toute l'image, ce qui échouait lorsqu'il y avait un décalage ou des espaces importants entre les caractéristiques
- Les utilisateurs peuvent désormais ajuster l'espacement des ROI indépendamment de la taille des ROI pour extraire avec précision des grilles à partir de données réelles telles que des réseaux de points laser, des motifs de diffraction ou toute caractéristique régulièrement espacée avec des espaces
- Ce correctif est fourni par Sigima 1.0.4
- Ceci clôture Issue #282 - Paramètres d'espacement manquants pour la ROI de grille

Recalcul automatique de l'analyse - Calculs redondants O(n^2) lors de l'ajout de ROI :

- Correction d'un problème de performance sévère où l'ajout d'une ROI à N images sélectionnées avec des résultats d'analyse (p. ex. Statistiques, Centroïde) déclenchait N^2 calculs au lieu de N
- Lorsque 10 images étaient sélectionnées avec des statistiques pré-calculées, l'ajout d'une ROI affichait 10 barres de progression de 10 calculs chacune (100 au total) au lieu d'une seule barre de progression avec 10 calculs
- Le problème se produisait car `auto_recompute_analysis()` appelait `compute_1_to_0()` qui, par défaut, traite tous les objets sélectionnés, pas seulement l'objet spécifique en cours de recalcul
- Ajout du paramètre `target_objs` à `compute_1_to_0()` pour permettre de spécifier quels objets traiter, et mise à jour de `auto_recompute_analysis()` pour l'utiliser
- Ceci clôture Issue #283 - Calculs redondants O(n^2) lors de l'ajout de ROI à plusieurs images avec des résultats d'analyse

Visualisation des résultats - Segments de résultats d'analyse difficiles à voir :

- Correction des marqueurs de résultats d'analyse (FWHM, caractéristiques des impulsions, etc.) difficiles ou impossibles à voir sur les graphiques de signaux
- Ces indicateurs de mesure apparaissaient en noir sur des fonds blancs ou en blanc sur des fonds sombres, les faisant se fondre dans le graphique
- Ils s'affichent désormais en vert vif avec des lignes plus épaisses, les rendant clairement visibles dans tous les thèmes de couleur
- Suppression également de la zone remplie qui obscurcissait les données de signal sous-jacentes - les marqueurs affichent désormais uniquement les lignes de délimitation indiquant la région mesurée

5.5.3 DataLab Version 1.0.2 (03/12/2025)

🛠 Correctifs depuis la version 1.0.1

Calibration de l'axe des signaux - Remplacer X par le Y d'un autre signal :

- Ajout de la nouvelle opération « Remplacer X par le Y d'un autre signal » dans le menu Traitement > Transformation d'axe pour les flux de travail de calibration des signaux
- Répond à une fonctionnalité critique manquante signalée par les utilisateurs qui avaient besoin d'appliquer une calibration en longueur d'onde ou des transformations similaires aux données de spectroscopie
- L'opération combine deux signaux : elle utilise les valeurs Y d'un signal comme nouvelles coordonnées X pour les valeurs Y d'un autre signal
- Contrairement au mode X-Y (qui rééchantillonne et interpole), cette opération utilise directement les tableaux Y sans interpolation, préservant ainsi les valeurs de calibration exactes
- Nécessite que les deux signaux aient le même nombre de points - affiche un message d'erreur clair si les tailles ne correspondent pas
- Transfert automatique des métadonnées : l'étiquette/unité de l'axe X est prise à partir de l'étiquette/unité Y du signal de calibration
- Emplacement dans le menu : Traitement > Transformation d'axe > « Remplacer X par le Y d'un autre signal »
- Ceci clôture Issue #273 - Calibration de l'axe des signaux manquante : aucun moyen de remplacer X par Y d'un autre signal

Mode X-Y :

- Le traitement Mode X-Y pour les signaux a été déplacé vers Traitement > Transformation des axes > « Mode X-Y » pour une meilleure découvrabilité
- La nuance entre le mode X-Y (qui rééchantillonne/interpole) et la nouvelle opération « Remplacer X par le Y d'un autre signal » a été clarifiée dans la documentation

Persistance du paramètre de verrouillage de la LUT :

- Correction du paramètre « Verrouiller la LUT lors de la mise à jour des données » ne persistant pas dans Paramètres > Visualisation > Images > Paramètres de visualisation d'image par défaut
- Le paramètre `keep_lut_range` n'était pas enregistré dans la configuration, ce qui faisait que la case à cocher se décochait systématiquement après validation (ajout de l'option manquante `ima_def_keep_lut_range` dans la configuration)
- Ceci clôture Issue #270 - Persistance du paramètre de verrouillage de la LUT dans les paramètres de visualisation d'image par défaut

Création de signaux personnalisés :

- Correction de l'erreur `AttributeError: 'NoneType' object has no attribute 'T'` lors de la création d'un signal personnalisé à partir du menu
- Ceci clôture Issue #269 - Création de signaux personnalisés : `AttributeError` lors de la création d'un signal à partir du menu

Exécution des macros :

- Correction de l'erreur `UnicodeEncodeError` lors de l'exécution de macros qui impriment des caractères Unicode (par exemple, des flèches →) sur les systèmes Windows avec certaines locales, par exemple cp1252 (ceci clôture Issue #263)
- Le sous-processus de la macro utilise désormais automatiquement l'encodage UTF-8 pour stdout et stderr, éliminant ainsi la nécessité d'ajouter manuellement `sys.stdout.reconfigure(encoding='utf-8')` au début de chaque macro.

Prise en charge des coordonnées non uniformes :

- Les coordonnées des ROI sont désormais automatiquement arrondies à une précision appropriée lors de la définition interactive des ROI à partir de formes géométriques, évitant ainsi des décimales excessives tout en maintenant une précision raisonnable par rapport à l'échantillonnage des données (1/10e de la période d'échantillonnage pour les signaux, 1/10e de l'espacement des pixels pour les images)
- Les coordonnées des ROI sont également arrondies lors de leur affichage dans la boîte de dialogue « Éditer numériquement », empêchant les erreurs d'arithmétique en virgule flottante d'afficher des décimales excessives (par exemple, 172.29999999999995 est désormais affiché sous forme de 172.3)
- Ceci clôture Issue #266 - Décimales excessives dans les coordonnées des ROI

Gestion des ROI polygonales :

— Correction de l'erreur `ValueError: Buffer has wrong number of dimensions` lors de la création de masques à partir de ROI polygonales dans l'éditeur de ROI
— Le PolygonalTool initialisait incorrectement les coordonnées des ROI sous forme de liste imbriquée au lieu d'une liste plate, ce qui provoquait l'échec du calcul du masque

Boîte de dialogue d'ouverture de fichiers HDF5 :

— Correction du bug où le choix de l'utilisateur dans la boîte de dialogue de confirmation « effacer l'espace de travail » était ignoré lors de l'ouverture de fichiers HDF5
— Lorsque l'utilisateur cliquait sur « Non » dans la boîte de dialogue, le paramètre de configuration par défaut était appliqué au lieu de respecter le choix de l'utilisateur
— Ceci clôture Issue #267 - La boîte de dialogue d'ouverture de fichiers HDF5 ignore le choix de l'utilisateur

Mise à jour de l'axe dans l'onglet Création :

— Correction du graphique ne se mettant pas à jour lors de la modification uniquement des paramètres xmin/xmax pour les signaux de distribution (Zéro, Normal, Poisson, Uniforme) dans l'onglet Création
— Le problème est survenu parce que le calcul du hash des données ne prenait en compte que les valeurs Y, de sorte que les modifications des limites de l'axe X n'étaient pas détectées
— Le graphique se rafraîchit désormais correctement lorsque n'importe quel paramètre d'axe change, même si les valeurs Y restent identiques
— Ceci clôture Issue #268 - L'axe de l'onglet Création ne se met pas à jour pour les signaux de distribution

Statistiques ROI avec ROI hors limites :

— Correction de l'erreur `ValueError: zero-size array to reduction operation minimum which has no identity` lors du calcul des statistiques sur des images avec ROI s'étendant au-delà des limites de la zone de dessin
— Le problème est survenu lorsqu'un ROI s'étendait partiellement ou complètement en dehors des limites de l'image, ce qui entraînait des tranches de tableau vides lors du calcul des statistiques
— Les boîtes englobantes des ROI sont désormais correctement limitées aux limites de l'image, et les ROI complètement hors limites renvoient des valeurs statistiques NaN
— Cette correction est implémentée dans la bibliothèque Sigima (voir Issue #1 - `ValueError` lors du calcul des statistiques sur ROI s'étendant au-delà des limites de l'image)

Sélection des onglets du panneau des propriétés des objets :

— Correction du comportement de sélection des onglets dans le panneau des propriétés des objets pour le rendre plus prévisible et convivial
— L'onglet Propriétés est désormais toujours affiché par défaut lors du changement d'objet, offrant une navigation cohérente
— Les onglets Création, Traitement et Analyse apparaissent désormais automatiquement une seule fois après leurs événements déclencheurs respectifs (création d'objet, traitement 1-vers-1 ou calcul d'analyse), puis reviennent à l'onglet Propriétés pour les sélections ultérieures
— Cela élimine le comportement déroutant où l'onglet persistait ou changeait arbitrairement en fonction des sélections précédentes
— Ceci clôture Issue #271 - Amélioration du comportement de sélection des onglets du panneau des propriétés des objets

5.5.4 DataLab Version 1.0.1 (18/11/2025)

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 version corrective qui résout les limitations de l'installateur Windows de la version 1.0.0 (publiée le 16/11/2025), où il n'était pas possible d'installer DataLab V1.0 aux côtés des versions majeures antérieures (V0.20 et antérieures) malgré le support de coexistence prévu. Cette version ne contient aucune nouvelle fonctionnalité ni modification fonctionnelle.

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

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

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

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)

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

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

🔧 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 l'étalonnage polynomial 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)

🛠 Correctifs depuis la version 0.20

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 :

🔒 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.

ⓘ 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)

5.6 Version 0.20

5.6.1 DataLab Version 0.20.1 (15/07/2025)

🛠 Corrections de bugs :

- Correction de l'"Issue #233 - Plantage brutal lors de la tentative d'activation de l'outil de statistiques de courbe sur un signal nul
- Correction de l'"Issue #184 - Le style du marqueur de courbe change de manière inattendue en « Carré » après la validation de la boîte de dialogue « Paramètres… »
- Correction de l'"Issue #117 - Le médian mobile du signal de DataLab plante sur Linux avec `mode='mirror'` : `free(): invalid next size (normal)` (c'est un bug dans SciPy v1.15.0 à v1.15.2, qui a été corrigé dans SciPy v1.15.3)
- Correction de l'"Issue #186 - Les fichiers texte d'image avec une virgule comme séparateur décimal ne peuvent pas être ouverts directement (seul l'assistant d'importation le supporte)
- Correction de l'"Issue #238 - Les fichiers texte d'image avec un délimiteur final entraînent des données avec une colonne superflue lorsqu'ils sont ouverts directement (ceci ne se produit pas dans l'assistant d'importation)
- Correction de l'"Issue #239 - L'assistant d'importation de texte ne conserve pas les titres et unités définis par l'utilisateur
- Correction de l'"Issue #240 - L'assistant d'importation de texte ne conserve pas le type de données défini par l'utilisateur (par exemple `int16`, `float32`, etc.)
- Correction de l'"Issue #235 - L'assistant d'importation de texte : ajout de la prise en charge de l'importation de fichiers de signal avec des valeurs entières
- Correction de l'"Issue #236 - L'assistant d'importation de texte : ajout de la prise en charge des fichiers `.mca`
- Correction de l'"Issue #243 - Affichage dans une nouvelle fenêtre (image) : les outils de profil d'intensité sont parfois désactivés (corrigé dans PlotPy v2.7.5)

5.6.2 DataLab Version 0.20.0 (28/04/2025)

⚙ 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`

5.7 Version 0.19

5.7.1 DataLab Version 0.19.2 (28/04/2025)

🛠 Correctifs :

— 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)

5.7.2 DataLab Version 0.19.1 (08/04/2025)

🛠 Correctifs :

— 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

5.7.3 DataLab Version 0.19.0 (31/03/2025)

✹ 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

🛠 Correctifs :
- 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

5.8 Version 0.18

5.8.1 DataLab Version 0.18.2 (16/03/2025)

ⓘ 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.)

🛠 Correctifs :
- 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

5.8.2 DataLab Version 0.18.1 (13/12/2024)

⚙ 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

🛠 Correctifs :

- 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

5.8.3 DataLab Version 0.18.0 (14/11/2024)

ⓘ 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.

🛠 Correctifs :

- Correction de l'"Issue #103 - `proxy.add_annotations_from_items` : la couleur de la forme de cercle semble être ignorée.

5.9 Version 0.17

5.9.1 DataLab Version 0.17.1 (01/10/2024)

ⓘ 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

🛠 Correctifs :

- 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

5.9.2 DataLab Version 0.17.0 (02/08/2024)

ⓘ 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

🛠 Correctifs :

- 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

5.10 Version 0.16

5.10.1 DataLab Version 0.16.4 (09/07/2024)

Cette version concerne une maintenance mineure.

🛠 Correctifs :

- 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`

5.10.2 DataLab Version 0.16.3 (03/07/2024)

🛠 Correctifs :

- 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

5.10.3 DataLab Version 0.16.2 (01/07/2024)

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)

🛠 Correctifs :

- 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)

5.10.4 DataLab Version 0.16.1 (21/06/2024)

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

🛠 Correctifs :

- 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

5.10.5 DataLab Version 0.16.0 (13/06/2024)

🛠 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

🛠 Correctifs :

— 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)

5.11 Version 0.15

5.11.1 DataLab Version 0.15.1 (03/05/2024)

🛠 Correctifs :
— 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

5.11.2 DataLab Version 0.15.0 (11/04/2024)

📦 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

🛠 Correctifs :

  - 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)

5.12 Version 0.14

5.12.1 DataLab Version 0.14.2 (22/03/2024)

⚠ 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)

🛠 Correctifs :

  - 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

5.12.2 DataLab Version 0.14.1 (18/03/2024)

🎉 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)

🛠 Correctifs :

- 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`

5.12.3 DataLab Version 0.14.0 (05/03/2024)

🐞 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`

🛠 Correctifs :

- 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

5.13 Version 0.12

5.13.1 DataLab Version 0.12.0 (16/02/2024)

 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>/σ(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

🛠 Correctifs :

- 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

5.14 Version 0.11

5.14.1 DataLab Version 0.11.0 (23/01/2024)

✹ 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, …)

🛠 Correctifs :

- 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

5.15 Version 0.10

5.15.1 DataLab Version 0.10.1 (22/12/2023)

Note : La V0.10.0 (publiée le 22/12/2023) a été presque immédiatement remplacée par la V0.10.1 en raison d'une correction de bug 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.

🛠 Correctifs :

- 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`

5.16 Version 0.9

5.16.1 DataLab Version 0.9.2 (15/12/2023)

🛠 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

5.16.2 DataLab Version 0.9.1 (01/12/2023)

🛠 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

5.16.3 DataLab Version 0.9.0 (29/11/2023)

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.