The Journal of Open Source Software

lys: interactive multi-dimensional data analysis and visualization platform

Asuka Nakamura [1]

1 RIKEN Center for Emergent Matter Science, Japan

DOI: 10.21105/joss.05869

Software

- Review
- Repository
- Archive

Editor: Elizabeth DuPre

Reviewers:

- @kuadrat
- @ziatdinovmax
- @pr4deepr

Submitted: 02 September 2023
Published: 14 December 2023

Summary

Analyzing and visualizing scientific data is an essential part of scientific research, by which researchers investigate complex phenomena behind it. In particular, the recent development of an open-source scientific Python ecosystem enables us to utilize state-of-the-art analysis and visualization. However, investigation of multi-dimensional data array still requires substantial coding, preventing intuitive and flexible analysis/visualization. `lys` is a Python-based multi-dimensional data analysis and visualization platform that provides graphical user interfaces (GUIs) to intuitively and flexibly manipulate multi-dimensional data arrays and publication-quality graphics. Massive multi-dimensional data over hundreds of gigabytes can be analyzed via automatic parallel calculation behind the GUI when `lys` is run on high-performance computers (HPCs). As well as the user-friendly GUIs, `lys` also provides flexibility for experts through its character user interface (CUI). The hybrid GUI/CUI architecture in `lys` enables an intuitive, low-code, parallel, flexible, and extensible multi-dimensional data analysis. `lys` is designed as a versatile data analysis and visualization platform that can handle all analysis processes from data loading to publication-quality figure generation. These features of `lys` enable us to minimize the time required for data analysis and visualization for a broad range of users.

Statement of need

Data analysis and visualization are indispensable parts of scientific research. Understanding experimental and simulated data deeply is essential for extracting complex phenomena behind them. To this end, intuitive and fast analysis/visualization is highly required. There are several well-known software that focus on analysis and visualization, such as `IGOR Pro` (WaveMetrics Inc., 2023) (RRID:SCR_000325) and `MATLAB` (The MathWorks Inc., 2023) (RRID:SCR_001622), as well as domain-specific software in respective fields. Although they have been utilized for scientific research for a long time, recent progress in experimental/computational methods has changed the situation. The data sizes obtained from experiments and calculations have increased rapidly over the past decade, reaching more than terabytes in many fields. As a result, the general analysis ecosystem maintained by the large scientific open-source community has an advantage in state-of-the-art analysis compared to proprietary/domain-specific software. Python and its scientific libraries provide one of the most popular analysis/visualization environments at the moment. In the scientific Python ecosystem, flexible and fast analysis of numerical arrays has been done by several popular libraries such as `numpy` (Harris et al., 2020), `scipy` (Virtanen et al., 2020), `dask` (Rocklin, 2015) combined with visualization tools such as `matplotlib` (Hunter, 2007), `pyqtgraph` (L. Campagnola & Moore, 2023) and `Mayavi` (Ramachandran & Varoquaux, 2011). Development of `Jupyter Notebook` (Kluyver et al., 2016) and related libraries further enhance the capability of interactive data analysis. However, most of these libraries require users to be familiar with low-level application programming interfaces (APIs), which prevents intuitive analysis and visualization. In particular, when the analyzed data is a more than 3-dimensional array, even simple interactive visualization usually requires Python code of tens of lines. Such on-demand analysis and visualization programs should be modified (and tested) when the data change, although a very similar process is frequently applied to data with different dimensions. Furthermore, the code for such an analysis should be preserved to guarantee the reproducibility of scientific results. To solve these problems, several Python-based tools such as `data-slicer` (Kramer & Chang, 2021) and `napari` (Chiu & Clack, 2022) have been developed. Although both of them offer sophisticated GUIs, they are not designed as a general research platform. `data-slicer` focuses on quick data inspection and visualization rather than publication-quality figure generation. `napari` is mainly developed for multi-dimensional image data, and therefore visualization of other data types such as curves, contours, vectors is limited. In such situations, a versatile Python-based analysis and visualization platform that can handle all scientific analysis processes, from data loading to publication-quality figure generation of a variety of data types, is highly required.

The multi-dimensional data analysis and visualization platform, `lys`, offers a graphical user interface (GUI) for intuitive analysis and visualization of multi-dimensional arrays. It employs `dask` as a backend, which can be used for easy parallel calculations on high-performance clusters (HPCs). Publication-quality and fast data visualizations are provided by `matplotlib` and `pyqtgraph`, respectively. `lys` is a low-code system where most analysis and visualization processes can be done from the GUI without any knowledge of respective libraries. In particular, a tool for interactive and fast analysis of multi-dimensional arrays has been developed. This tool allows all analysis processes to be exported as a single file, ensuring scientific reproducibility. In contrast to such a user-friendly GUI, `lys` can be easily extended because it employs a hybrid CUI/GUI architecture. Users can edit and run their own Python code in `lys` to extend the functionalities of `lys`.

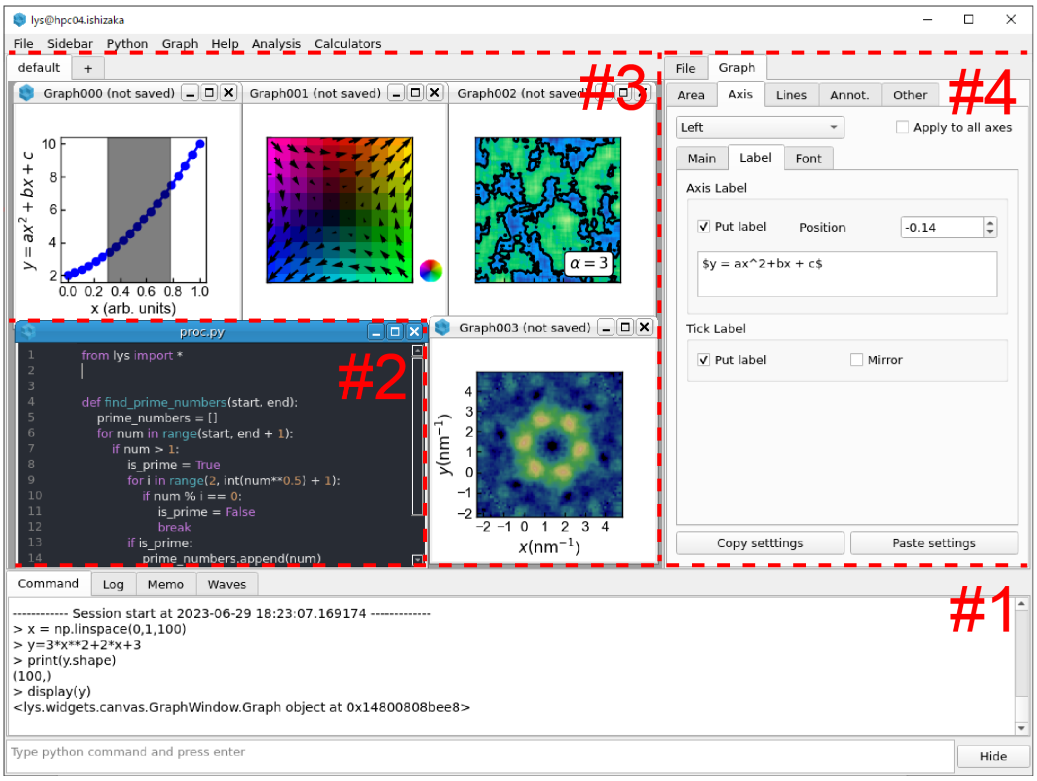
The philosophy of `lys` is to serve as a versatile data analysis and visualization platform, rather than a basic image viewer/analysis program. It was developed to minimize the time required for data analysis and visualization for researchers. All of the processes from loading data to generating publication-quality figures can be done in `lys`. In addition, these processes (including user-defined Python code) are stored in a single directory and can be used to reproduce the results. The rich features of `lys` significantly reduce the time for analysis/visualization of multi-dimensional arrays.

Overview

`lys` is a hybrid GUI/CUI platform oriented towards multi-dimensional data analysis. Figure 1 shows the main features of `lys`. Arbitrary Python commands can be executed from an integrated Python shell (#1). User-defined Python scripts can be edited by the internal editor (#2) and can be executed. `matplotlib` graphs that contain curves, images, vector fields, and RGB images can be displayed (#3) and edited via GUI in the sidebar (#4).

`MultiCut` is a central tool in `lys`, which enables intuitive, low-code, parallel, flexible, and extensible analysis for multi-dimensional arrays. In the following, the data analysis and visualization processes are explained, using three-dimensional movie data $A(x_i, y_j, t_k)$ as an example (i, j, k represent indices of the array). The data analysis in `MultiCut` is done in four steps. First, the original N-dimensional data A is modified by preprocessing as $A' = P(A)$, where P is an arbitrary function that translates N dimensional data to M dimensional data. In the example case in Fig. 2, impulsive noise in the original data is removed using $3 \times 3 \times 3$ median filter:

$$A'(x_i, y_j, t_k) = \mathcal{M}[A(x_i, y_j, t_k)],$$



```python
from lys import *


def find_prime_numbers(start, end):
    prime_numbers = []
    for num in range(start, end + 1):
        if num > 1:
            is_prime = True
            for i in range(2, int(num**0.5) + 1):
                if num % i == 0:
                    is_prime = False
                    break
            if is_prime:
                prime_numbers.append(num)
```

Left ☐ Apply to all axes

☑ Put label Position -0.14

$y = ax^2+bx + c$

☑ Put label ☐ Mirror

```
------------ Session start at 2023-06-29 18:23:07.169174 -------------
> x = np.linspace(0,1,100)
> y=3*x**2+2*x+3
> print(y.shape)
(100,)
> display(y)
<lys.widgets.canvas.GraphWindow.Graph object at 0x14800808bee8>
```

Type python command and press enter

Figure 1: Screenshot of lys. Users can execute arbitrary Python commands from embedded CUI (#1), which can be extended by the user-defined scripts (#2). Matplotlib figures that contain curves, images, vector fields, and RGB images can be displayed (#3). These figures can be edited via GUI in the sidebar (#4).

where $P = \mathcal{M}$ represents median filter to remove the noise. Since the median filter does not change the dimension of data, $N = 3$ equals M in this case. The preprocessing is used for heavy analysis that requires whole N-dimensional data. Second, MultiCut generates an arbitrary number of 2-dimensional images and 1-dimensional curves from M-dimensional A' by taking a summation along arbitrary axes. In the example case in Fig. 2, an image $I(x_i, y_j)$ and a curve $C(t_k)$ is generated as:

$$I(x_i, y_j) = \sum_{t_k} A'(x_i, y_j, t_k),$$

$$C(t_k) = \sum_{x_i, y_j} A'(x_i, y_j, t_k).$$

The range of the summation is specified from the GUI as described later. These images and curves are used for visualization in step 4. Third, each image and curve can be individually modified by postprocessing. In Fig. 2, Fourier transformation along time axis $\mathcal{F}_t$ is applied to time-dependent image intensity $C(t_k)$:

$$C'(\omega_l) = \mathcal{F}_t[C(t_k)].$$

Different from the preprocessing (step 1), the postprocessing function accesses only an image or a curve. In addition, the postprocessing should be executed within a short time (< 0.1 s) because it is repeatedly called whenever the summation range in step 2 is changed. Finally, these analyzed data are displayed in a GUI, where users can modify the ranges of the summation interactively. Once the summation range from step 2 is changed, the postprocess is recalculated and the result is automatically updated. A four-step calculation enables flexible analysis of multi-dimensional data. In the above example, the spectrum of the image intensity $C'(\omega_l)$ within the user-specified image region can be interactively analyzed and displayed while the time-consuming median filter is done only once in step 1. It should be noted that a multi-dimensional array of arbitrary dimensions can be analyzed and visualized by `MultiCut` although the given example is for a 3-dimensional case. Constructing such an interactive analysis system is a hard task in conventional Python systems. Once the interactive analysis system is set up using MultiCut, the settings for the analysis can be exported as a file and reused. This guarantees the scientific reproducibility of the data analysis, which can be verified by other scientists. In addition, all of the processes in `lys` are implemented using `dask` arrays, and therefore all calculations can be performed in parallel when they are done on HPC systems.

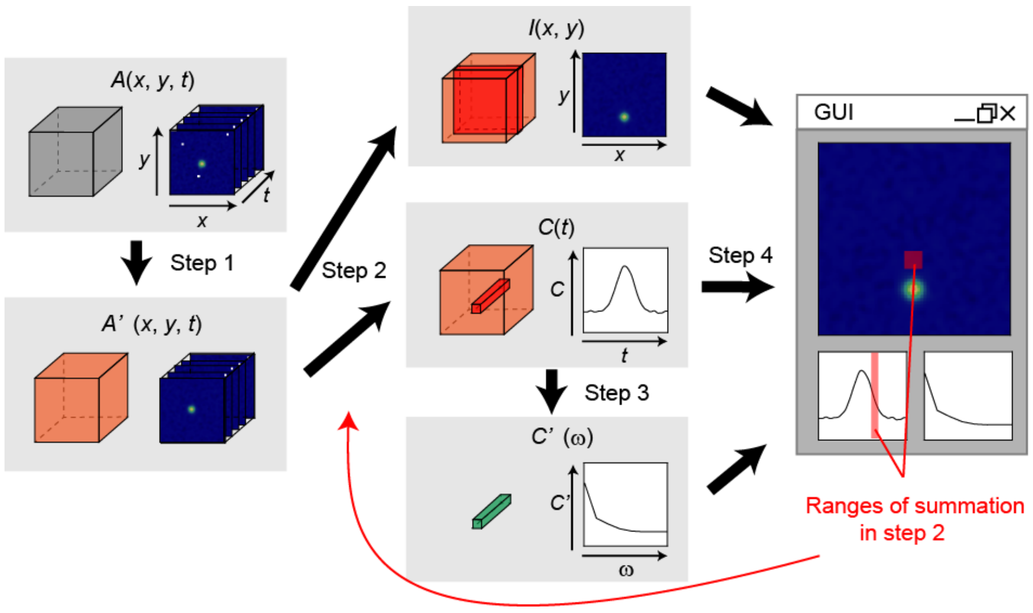


Figure 2: Example for 3-dimensional time-dependent data analysis and visualization by MultiCut. The median filter is applied to the original data (Step 1), from which several curves and images are generated (Step 2). Each curve and image are independently processed in Step 3, and visualized in a single GUI window (Step 4).

In addition to the features described above, `lys` provides some basic analysis such as data fitting and array editor GUIs. Combining these functionalities of `lys` offers intuitive, low-code, fast, and flexible analysis to users not familiar with Python while preserving the extensibility for experts.

As compared to other analysis/visualization software, `lys` has several advantages. First, it employs Python (`numpy`/`dask`) as a backend, and therefore a variety of scientific computing libraries such as `scipy` can be used. This cannot be achieved by similar software such as `IGOR Pro` and `MATLAB`. Second, `lys` is open-source software. Users can verify the software and modify it if needed, which cannot be realized in proprietary software. Third, `lys` offers interactive GUIs represented by `MultiCut`. Although there has been much effort to realize sophisticated GUIs such as `napari` (Chiu & Clack, 2022) and `data-slicer` (Kramer & Chang, 2021), this is still very limited in the scientific Python ecosystems so far. Fourth, `lys` can treat massive multi-dimensional arrays of more than hundreds of gigabytes through `dask`. The coexistence of intuitive GUI and fast parallel calculation is very limited in other similar software/libraries so far. Finally, `lys` is a general platform for data analysis and visualization. All of the processes from loading data to generating publication-quality figures can be done in `lys`. Although `data-slicer` offers similar and interactive data manipulation, it is data inspection and visualization software rather than the general platform for research scientists. While `napari` also offers similar functionalities for multi-dimensional images, `lys` is designed to handle a variety of data types (images, curves, contours, vector fields, and RGB images) to serve as a general-purpose platform.

Projects using the software

As `lys` is a general-purpose multi-dimensional data analysis system, it has been used in many works within the last five years, particularly for our experiments and simulations. Simple visualization functionalities are used for the analysis of movies obtained by ultrafast electron diffraction and microscopy (Asuka Nakamura et al., 2018, 2020). A pre-release version of MultiCut was used for analyzing the propagation of nanometric acoustic waves (Asuka Nakamura et al., 2023) and magnetic-texture dynamics (Takahiro Shimojima et al., 2021). Analyzing massive five-dimensional data sets obtained by five-dimensional scanning transmission electron microscopy (A. Nakamura et al., 2022; T. Shimojima et al., 2023a; T. Shimojima et al., 2023b) was also achieved using parallel calculations on an HPC, demonstrating the scalability of `lys`. It was also used for the postprocessing of finite-element simulation results (A. Nakamura et al., 2021).

Acknowledgements

We acknowledge contributions from Yusuke Chiashi, Jumpei Koga, Dongxue Han, and comments from Takahiro Shimojima and Kyoko Ishizaka. This work was partially supported by a Grant-in-Aid for Scientific Research (KAKENHI) (Grant No. 21K14488).

References

Chiu, C.-L., & Clack, N. (2022). Napari: A Python multi-dimensional image viewer platform for the research community. *Microscopy and Microanalysis*, *28*(S1), 1576–1577. https://doi.org/10.1017/S1431927622006328

Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., Kerkwijk, M. H. van, Brett, M., Haldane, A., Río, J. F. del, Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, *9*(3), 90–95. https://doi.org/10.1109/MCSE.2007.55

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., Ivanov, P., Avila, D., Abdalla, S., & Willing, C. (2016). *Jupyter notebooks – a publishing format for reproducible computational workflows* (F. Loizides & B. Schmidt, Eds.; pp. 87–90). IOS Press.

Kramer, K., & Chang, J. (2021). Visualization of multi-dimensional data – the data-slicer package. *Journal of Open Source Software*, *6*(60), 2969. https://doi.org/10.21105/joss.02969

L. Campagnola, K. Lyons, & Moore, O. (2023). PyQtGraph: Scientific graphics and GUI library for Python. In *GitHub repository*. GitHub. https://github.com/pyqtgraph/pyqtgraph

Nakamura, Asuka, Shimojima, T., Chiashi, Y., Kamitani, M., Sakai, H., Ishiwata, S., Li, H., & Ishizaka, K. (2020). Nanoscale imaging of unusual photoacoustic waves in thin flake VTe2. *Nano Letters*, *20*(7), 4932–4938. https://doi.org/10.1021/acs.nanolett.0c01006

Nakamura, A., Shimojima, T., & Ishizaka, K. (2021). Finite-element simulation of photoinduced strain dynamics in silicon thin plates. *Structural Dynamics*, *8*(2). https://doi.org/10.1063/4.0000059

Nakamura, A., Shimojima, T., & Ishizaka, K. (2022). Visualizing optically-induced strains by five-dimensional ultrafast electron microscopy. *Faraday Discuss.*, *237*, 27–39. https://doi.org/10.1039/D2FD00062H

Nakamura, Asuka, Shimojima, T., & Ishizaka, K. (2023). Characterizing an optically induced sub-micrometer gigahertz acoustic wave in a silicon thin plate. *Nano Letters*, *23*(7), 2490–2495. https://doi.org/10.1021/acs.nanolett.2c03938

Nakamura, Asuka, Shimojima, T., Matsuura, M., Chiashi, Y., Kamitani, M., Sakai, H., Ishiwata, S., Li, H., Oshiyama, A., & Ishizaka, K. (2018). Evaluation of photo-induced shear strain in monoclinic VTe2 by ultrafast electron diffraction. *Applied Physics Express*, *11*(9), 092601. https://doi.org/10.7567/APEX.11.092601

Ramachandran, P., & Varoquaux, G. (2011). Mayavi: 3D Visualization of Scientific Data. *Computing in Science & Engineering*, *13*(2), 40–51. https://doi.org/10.1109/mcse.2011.35

Rocklin, M. (2015). Dask: Parallel computation with blocked algorithms and task scheduling. *Proceedings of the 14th Python in Science Conference*. https://doi.org/10.25080/majora-7b98e3ed-013

Shimojima, T., Nakamura, A., & Ishizaka, K. (2023a). Development of five-dimensional scanning transmission electron microscopy. *Review of Scientific Instruments*, *94*(2). https://doi.org/10.1063/5.0106517

Shimojima, T., Nakamura, A., & Ishizaka, K. (2023b). Development and applications of ultrafast transmission electron microscopy. *Microscopy*. https://doi.org/10.1093/jmicro/dfad021

Shimojima, Takahiro, Nakamura, A., Yu, X., Karube, K., Taguchi, Y., Tokura, Y., & Ishizaka, K. (2021). Nano-to-micro spatiotemporal imaging of magnetic skyrmion's life cycle. *Science Advances*, *7*(25), eabg1322. https://doi.org/10.1126/sciadv.abg1322

The MathWorks Inc. (2023). *MATLAB (R2023b)*. The MathWorks Inc. https://www.mathworks.com

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., … SciPy 1.0 Contributors. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, *17*, 261–272. https://doi.org/10.1038/s41592-019-0686-2

WaveMetrics Inc. (2023). *IGOR pro 9.05*. WaveMetrics Inc. https://www.wavemetrics.com