ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-TUCPR02

DATA EXPLORATION AND ANALYSIS WITH Jupyter NOTEBOOKS

H. Fangohr*[1], M. Beg, M. Bergemann, V. Bondar, S. Brockhauser[2,3], C. Carinan, R. Costa, F. Dall'Antonia, C. Danilevski, J. C. E, W. Ehsan, S. G. Esenov, R. Fabbri, S. Fangohr, G. Flucke, C. Fortmann[4], D. Fulla Marsa, G. Giovanetti, D. Goeries, S. Hauf, D. G. Hickin, T. Jarosiewicz[5], E. Kamil, M. Karnevskiy, Y. Kirienko, A. Klimovskaia, T. A. Kluyver, M. Kuster, L. Le Guyader, A. Madsen, L. G. Maia, D. Mamchyk, L. Mercadier, T. Michelat, J. Möller, I. Mohacsi, A. Parenti, M. Reiser, R. Rosca, D. B. Rueck, T. Rüter, H. Santos, R. Schaffer, A. Scherz, M. Scholz, A. Silenzi, M. Spirzewski[5], J. Sztuk, J. Szuba, S. Trojanowski[5], K. Wrona, A. A. Yaroslavtsev, J. Zhu
European XFEL GmbH, Schenefeld, Germany
J. Reppin, F. Schlünzen, M. Schuh, DESY, Hamburg, Germany
E. Fernandez-del-Castillo, G. Sipos, EGI Foundation, Amserdam, Netherlands
T. H. Rod, J. R. Selknaes, J. W. Taylor, ESS, Copenhagen, Denmark
A. Campbell, A. Götz, J. Kieffer, ESRF, Grenoble, France
J. Hall, E. Pellegrini, J. F. Perrin, ILL, Grenoble, France

[1] also at University of Southampton, Southampton, United Kingdom
[2] also at University of Szeged, Szeged, Hungary
[3] also at Biological Research Center of the Hungarian Academy of Sciences, Szeged, Hungary
[4] also at Max-Planck-Inst. for Evolutionary Biology, Plön, Germany
[5]also at NCBJ, Otwock, Poland

Abstract

Jupyter notebooks are executable documents that are displayed in a web browser. The notebook elements consist of human-authored contextual elements and computer code, and computer-generated output from executing the computer code. Such outputs can include tables and plots. The notebook elements can be executed interactively, and the whole notebook can be saved, re-loaded and re-executed, or converted to read-only formats such as HTML, LaTeX and PDF. Exploiting these characteristics, Jupyter notebooks can be used to improve the effectiveness of computational and data exploration, documentation, communication, reproducibility and re-usability of scientific research results. They also serve as building blocks of remote data access and analysis as is required for facilities hosting large data sets and initiatives such as the European Open Science Cloud (EOSC). In this contribution we report from our experience of using Jupyter notebooks for data analysis at research facilities, and outline opportunities and future plans.

INTRODUCTION

Data analysis and data science studies often begin with interactive exploration, often already during the experiment. Short feedback cycles are crucial for this: the person exploring the data should be free to quickly try different data analysis options and see the results with minimal mental overhead. As the analysis progresses, the focus shifts to recording and communicating findings and how they were reached. Whether someone shares their analysis or keeps it for their own reference, they will need an explanation of what was done, and a record of the code used and the results, to establish confidence and to serve as a base for further work.

Digital notebook interfaces are an increasingly popular way to meet these needs. A notebook is a document combining free text with code which can be interactively executed, producing results inline which are saved as part of the notebook. Such interfaces have been familiar in computational mathematics software such as Mathematica and SageMath for many years. More recently, notebooks have spread to general programming and data science, in particular as Jupyter notebooks (formerly IPython notebooks) [1, 2].

Figure 1 shows an example Jupyter notebook introducing some of the capabilities. This example is available online [3], where it can also be interactively executed using the Binder service [4].

Jupyter is a notebook interface which can work with various programming languages, thanks to backends known as *kernels* which can be installed individually. Jupyter notebooks are viewed and edited through an interface in the web browser. This can be run by an individual on their own computer, but it is also relatively straightforward to provide remote access to Jupyter running on a server. Jupyter notebooks are typically used through common web browsers, but they can also be viewed and edited in a desktop application called *nteract* [5]. Very recently, a useful review on collaborative data science with Jupyter notebooks has become available [6].

In this paper, we discuss our experiences of using Jupyter notebooks at a range of research facilities and will illustrate this with use cases from European XFEL (EuXFEL) to high-

* hans.fangohr@xfel.eu

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-TUCPR02


```
In [1]: # Import Python and libraries we need later
        %matplotlib inline
        from numpy import exp, cos, linspace
        import pylab
        from ipywidgets import interact
```

Mathematical model: We would like to understand $f(t, \alpha, \omega) = \exp(-\alpha t)\cos(\omega t)$

Code: Here is an implementation:

```
In [2]: def f(t, alpha, omega):
            """Computes and returns exp(-alpha*t) * cos(omega*t)"""
            return exp(-alpha * t) * cos(omega * t)
```

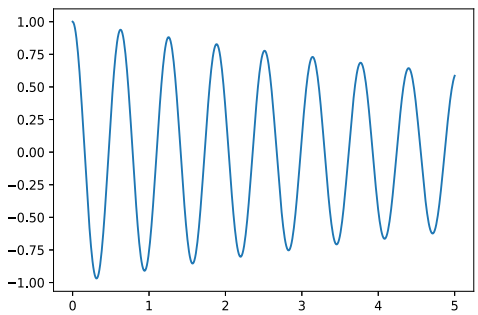
Interactive exploration: We can execute the function for values of t, α and ω:

```
In [3]: f(t=0.1, alpha=1, omega=10)
Out[3]: 0.48888574340060287
```

Or produce a plot (in a function `plot_f` so it can be re-used for different parameters):

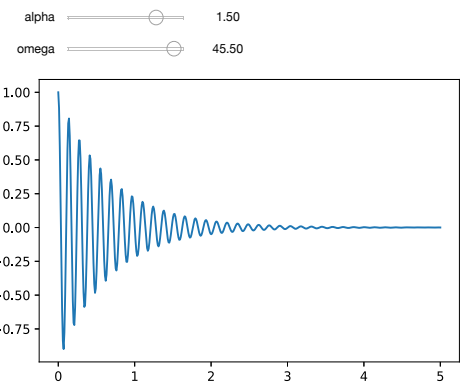
```
In [4]: def plot_f(alpha, omega):
            ts = linspace(0, 5, 500)    # 500 points in interval [0, 5]
            ys = f(ts, alpha, omega)
            pylab.plot(ts, ys, '-')
```

```
In [5]: plot_f(alpha=0.1, omega=10)    # call function and create plot
```



Using interaction *widgets*, we can re-trigger execution of the `plot_f` function via GUI elements such as sliders.

```
In [6]: interact(plot_f, alpha=(0, 2, 0.1), omega=(0, 50, 0.5));
```



Conclusion: We observe that parameter α is responsible for damping, and ω for the frequency.

Figure 1: An example of a Jupyter notebook showing that textual description (including LaTeX commands) can be mixed with code and outputs such as numbers and graphs, produced by executing the code. In the last cell, we have created two sliders for the parameters α and ω which can now be varied by moving the sliders with the mouse or entering numbers to the right of the slider. As the numbers are changed or the slider is moved, the plot is re-calculated and updated in real time. This notebook example can be inspected and executed in the cloud using a web browser [3].

light how notebooks can be useful for science at large-scale experimental facilities with high volumes of data.

DRIVING ANALYSIS FROM NOTEBOOK WITH TAILORED PYTHON LIBRARY

The Spectroscopy & Coherent Scattering (SCS) instrument at EuXFEL makes extensive use of Jupyter notebooks on a daily basis, both during user experiments and during commissioning of the instrument components, as well as weeks or months after the data were recorded for more detailed offline analysis. To avoid long code snippets in notebooks, we developed a Toolbox package [7] that comprises several functions that are used on a daily basis at the instrument. The package is built on the facility's library for reading data files and provides higher level functionality tailored for the SCS instrument.

An example of typical analysis and use of notebooks is presented in Fig. 2. One well-established technique at SCS is X-ray absorption spectroscopy (XAS): the photon energy of the Free Electron Laser (FEL) beam is monochromatized by a grating and varied by scanning its angle. The X-ray pulse energy is measured before and after interaction with a sample, thus allowing for energy-resolved absorption measurements. The incoming X-ray pulse energy (I_0) is measured by a gas monitor (XGM). The transmitted pulse intensity (I_1) is then measured by the Transmitted Intensity Monitor (TIM): fluorescence induced by the X-ray on a CVD diamond screen is detected by Micro-Channel Plates (MCP). The data produced represent a digitized time trace of the MCP, consisting of peaks proportional to the transmitted pulse intensity. As a first step of analysis, the XGM data, monochromator data and TIM data need to be synchronised by train id, a unique integer used to label the X-ray bunches arriving with a frequency of 10Hz. Each train contains up to 2700 X-ray pulses. XGM and TIM data also have to be aligned by pulse id within the train. Secondly, the peak integration parameters of the TIM digitized traces must be checked and integration is performed. Finally, the absorption, defined as $-\log(I_1/I_0)$, is computed and binned as a function of the monochromator energy. These three steps are condensed into three lines of code, as seen in Fig. 2.

Due to the interactivity of the notebook, one can make sanity checks at each step involved, for instance to display and adjust the integration parameters of the TIM trace, or to change which detector is used as either I_0 or I_1, or to compute different XAS spectra for specific pulses in the bunch train to see changes induced by the X-rays.

COLLECTION OF NOTEBOOK RECIPES FOR TYPICAL TASKS

The Materials Imaging and Dynamics instrument (MID) uses the EuXFEL beam with 10 trains per second and currently up to 300 X-ray pulses per train. With a 1 million pixel detector and 2 bytes per pixel for raw data, a dataset of over one 1 TB is measured at MID just from that detector in less that 3 minutes.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-TUCPR02

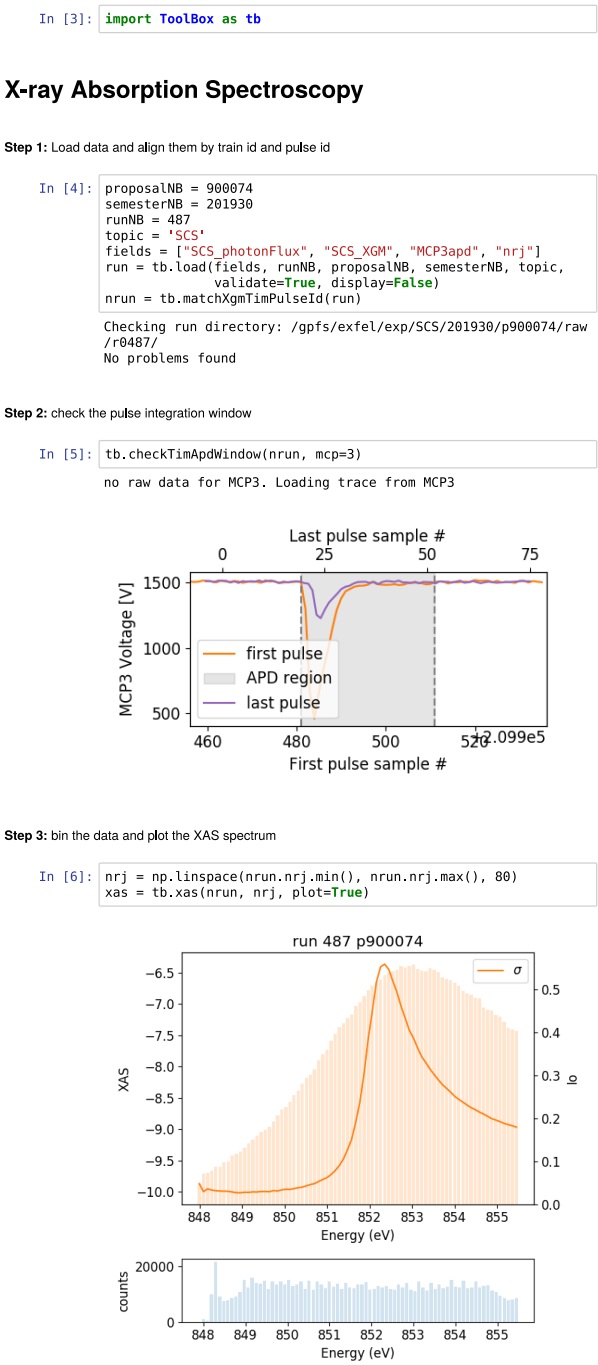


Figure 2: Example notebook as used at the SCS instrument.

Crucial for the success of every experiment is reliable, efficient and flexible data analysis that allows scientists to interpret the data and to make decisions about the next steps of the experiment through the course of a five-day beamtime.

For each experiment – which is also referred to as a *proposal* – a dedicated proposal folder is created with subdirectories for experimental data, analysis scripts, preliminary analysis results, etc. Additionally, we pilot for MID an approach where each proposal folder contains an independent copy of the software environment to run the Jupyter notebooks, so that analysis code will not be broken later by an unexpected update to a new version of a library in a centrally managed software environment. This approach further allows to add additional software on top of the standard software environment with specialized programs for the current experiment.

So far, the software environment has been provided as an Anaconda distribution installation, but we are piloting the use of Singularity [8] for software provision at EuXFEL: multiple Singularity images (each being a single file) can be used and archived to preserve changing or tailored software environments. We also find that the many files that come with a typical Anaconda distribution put stress on the facility's file system that was desiged to handle large files, not many very small files: this problem disappears if the Anaconda distribution is installed inside the Singularity container.

In order to document the building blocks of a successful data analysis and to make them accessible to scientists who are not software experts, we provide a collection of example Jupyter notebooks for the users of the MID instrument. Each notebook deals with a particular data analysis task. Alongside the actual code, detailed information explains the theory behind the analysis and how it is translated into code. Markdown cells allow for emphasising and structuring plain text and including mathematical formulae and tables, which makes it easier to understand the different steps. Each user group at MID gets a copy of these notebooks in their proposal directory on the Maxwell cluster, where they can execute the examples and copy code snippets to serve as a starting point for their own analyses.

The MID scientists also use Jupyter notebooks as a form of electronic logbooks to keep a chronological record of their experiments and analyses. While the ELOG software [9] is widely used at European XFEL, the convenience of including executable code and its output in Jupyter notebooks make them desirable as an alternative logbook. A number of extensions enhance Jupyter for this purpose, including provision of a table of contents in the notebook interface, and allowing the user to collapse ('fold') parts of the notebook based on headings or code structure – this makes navigation of long notebooks more convenient.

ONLINE DATA ANALYSIS – LIVE ANALYSIS FROM DATA STREAM

The success of an experiment at a photon facility can depend on fast decision making capability during the beamtime, often a time scale of seconds for (near-realtime) feedback is desired to adjust experimental parameters. Due to the versatility of experiments, the requirements for this kind of feedback can vary significantly from experiment to experiment.

During the experiment, trains of data are written to disk and (parts of) the data can simultaneously be made accessible through a ZeroMQ-stream over the network [10] for near-realtime analysis on dedicated computers. Here we outline a

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-TUCPR02

solution to provide a flexible configuration of online analysis configurations [11].

Online analysis routines are broken down into small units of analysis and plotting functionality which are implemented in a pool of software modules (here Python classes) from which the a combination of tools can be selected. The class objects can provide statistical information like histograms or fits of probability distributions, and trends and correlation plots of experiment parameters like intensity or contrast values etc. Each class produces (at least) one graphical representation of the output.

We have found that Jupyter notebooks can be used as a framework for such online data analysis to gather and execute these analysis and visualisation tools from the pool and display the results (see Fig. 3 for an example).

Matplotlib is used to create the main figure for displaying the results. The figure consists of a grid of axes where single plots can span multiple columns and rows of the grid to increase the readability and to emphasize particular information. The axes handles are passed upon instantiation to the main class of each analysis tool. This class further has to provide a method that creates and updates the graphs. Additional optional arguments can be passed to set specific parameters for each analysis routine.

Data access is provided through a reader class object that receives data from the ZeroMQ-stream. The analysis classes are instantiated with the same reader-class instance as first argument to ensure that every analysis routine in the list of selected analysis instances uses the same data. A copy of the data itself is accessible for the analysis classes through properties of the reader instance.

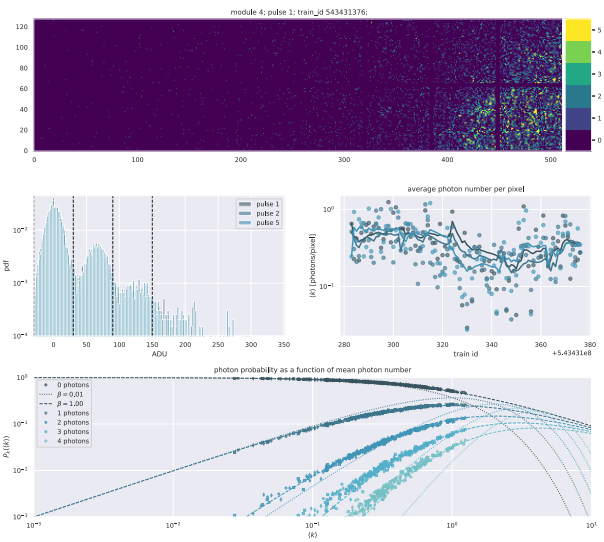


Figure 3: A section of the main figure of an online data analysis Jupyter notebook, which – when used during an experiment – updates in near-real time. The graphs show a module of the AGIPD detector (first row), histograms of pulse intensities and the average scattering intensity as a function of bunch train id (second row) and the distribution of photon events (third row).

After setting up the figure and instantiating the reader and the analysis class instances, an infinite loop is started. In every iteration, the reader fetches and processes data from the stream. Then, every analysis-class instance is called to perform its analysis and update its graphs. The loop can be stopped by interrupting the kernel in the notebook, for example to modify the selection of analysis classes and thus plots before restarting the live display.

Jupyter notebooks may thus present an attractive user interface for such flexible and modular online analysis software – we believe the flexibility to re-configure the online analysis without expert support is an attractive feature of the described approach.

DETECTOR CALIBRATION AND NOTEBOOK-AS-A-SCRIPT APPROACH

From raw detector data, best estimates of physical quantities such as photons per pixel are generated as a facility service [12]. One of EuXFEL's Megapixel detectors can produce 10 – 15 GB/s amounting to Petabytes of data for the entire stay of a user group. In order to process these data volumes efficiently, file-based calibration is executed concurrently on the Maxwell HPC cluster via SLURM scheduling.

The various steps in the correction and calibration of stored data are defined using Jupyter notebooks: the necessary combination of calls of the processing library are assembled in a Jupyter notebook so that the calibrated data set is computed as the notebook execution progresses (see Fig. 4 left). As part of the processing, diagnostic plots are created in the notebook in the appropriate places. One particular advantage of this notebook-as-a-script approach is that it offers the detector scientists a convenient tool for rapid prototyping and refinement of calibration methods, especially as much of the feedback needed to validate and evaluate these methods is in the form of plots, which can be displayed inline. Therefore, Jupyter notebooks enable a versatile production and development environment for data calibration pipelines, which are tailored to individual detector models. Each notebook defines a set of input parameters using *nbparameterise* [13] relating to detector settings, calibration options and data access. The initial default values are adjusted by an expert user who runs the notebook through a dedicated command-line tool. Facility users can also trigger the calibration pipeline through a web service.

Besides the calibrated data, each execution of a notebook also produces a document with plots and tables as an automatically generated report on data quality (Fig. 4 right). The report functionality takes advantage of the Jupyter ecosystem: the notebook defining the pipeline is converted into a PDF file including the output and notes, structured by Markdown headings. This avoids the need to mix code for assembling a report into the calibration and plotting routines. Specifically, the conversion uses *nbsphinx* [14] to integrate with the *Sphinx* documentation tool, which uses LaTeX to produce a PDF.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-TUCPR02

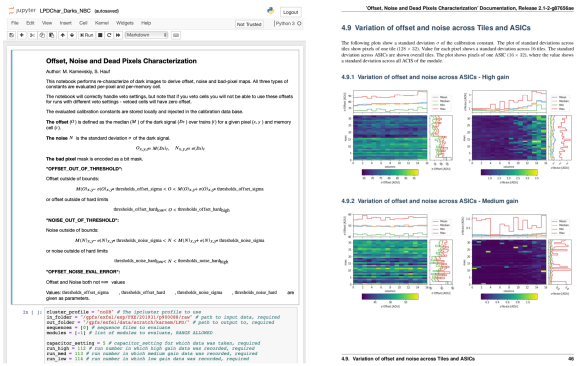


Figure 4: Example from notebook-as-a-script approach. *(Left):* notebook with calibration pipeline instructions. *(Right):* auto-generated report from executing notebook.

CONDUCTING SIMULATIONS STUDIES FROM NOTEBOOK

SPB/SFX is EuXFEL's instrument for single particles, clusters, and biomolecules and serial femtosecond crystallography. The start-to-end simulation platform SIMEX [15] was developed to simulate various types of experiments performed at advanced photon sources including synchrotron radiation and XFEL facilities. In a start-to-end simulation, every part of the experiment from the photon source to the photon detector, including the X-ray optical beam shaping and the interaction with the sample, is simulated such that the effects of the source spectral, temporal and spatial structure, X-ray optical components, and radiation damage to the sample are taken into account.

To simplify the usage of the platform, we implemented a unified API in a Python library for the users and beamline scientists to conduct the requested simulation effectively. With the help of Jupyter notebooks, the simulation platform can be easily exposed as an online service. Thus, there is no need for the users to install the simulation platform locally, to deal with the complicated dependencies or to change their operating systems. Some of the compute-intensive simulations such as particle in cell, molecular dynamics – to describe the photon-matter interaction process – and diffraction simulations for large atomic systems, require high performance computing resources. The Jupyter notebook-based online-simulation service provides a solution for those users who lack the resources to perform the simulation.

At every step, the simulation can be parameterized and executed in the Jupyter notebook as shown in Fig. 5. The results can be displayed and saved in each step for users to adjust the parameters immediately.

BLENDING GUI-DRIVEN AND SCRIPTED DATA ANALYSIS

There are different styles and user interfaces for data analysis and data exploration [16]:

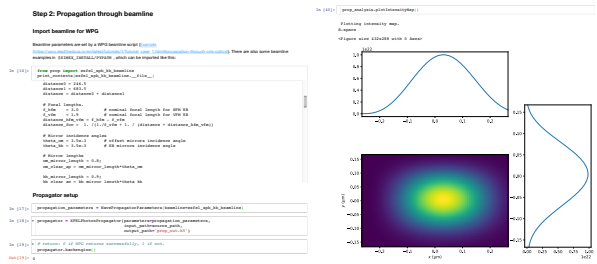


Figure 5: Driving a simulation through the Jupyter notebook: An example of the photon propagation step of SIMEX, showing how the input and output of each step can be displayed and saved.

(i) data analysis driven by scripts: The author of a script can, in principle, carry out any analysis operation they like as they have a general purpose programming language to request or even implement the particular operation. Two advantages of this approach are its generalism and reproducibility: any problem can be analysed and scripts can be re-executed to repeat the same analysis. Disadvantages of the method are the strictly sequential execution, the lack of modularity and the inherent separation of the script from documentation and outputs.

(ii) data analysis driven through a GUI: Another approach is to create a graphical user interface (GUI) where the user clicks buttons and other graphical elements to control data analysis. This can work well if the data analysis task is clearly defined and does not change frequently; it is a convenient method to inspect data sets with a fixed structure. It is advantageous that using the a GUI does not require any programming skills. However, it is difficult to change the analysis that is pre-defined in the GUI. Furthermore, GUI-Applications usually lack the mechanism or the implementation to capture a full record of GUI-events, which renders GUI-based analysis generally not reproducible.

Whether approach (i) or (ii) is preferred is determined by personal taste and the data analysis task at hand – both extremes have their justified place.

Using notebooks is similar to approach (i) in that analysis steps can be defined by writing computer code. However, where pre-defined notebooks are provided, it also shows some similarity to the GUI approach (ii): by providing a template notebook that already contains the right analysis commands, a user without programming knowledge can use the notebook (by executing one cell after another) without having to write code themselves.

Furthermore, through a library known as Jupyter Widgets [17], it is possible to create GUI elements inside a Jupyter notebook. It is thus possible to have a *blended approach* that allows to change from scripted analysis to GUIs and back within the notebook. We have used this approach in *karabo_data_interactive*, a tool to interactively browse through images from X-ray pixel detectors at European XFEL, facilitating exploratory analysis in notebooks (Fig. 6).

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-TUCPR02

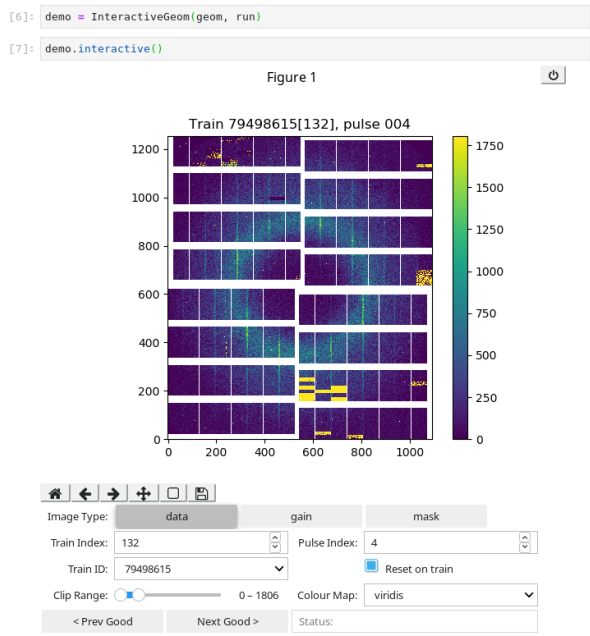


Figure 6: An example of a the *karabo_data_interactive* widget. Users can create the widget from a large dataset, and then interactively browse through different frames of detector images, change the type of image being looked at (e.g. gain or mask), zoom/pan the image, and change the colour range to bring out different details in the data.

We note that the reproducibility of notebooks (i.e. the ability to re-compute the same results by re-executing the code) is currently broken when widgets are used: the widget may require the user to repeat the clicks carried out before. Solutions are discussed in the Jupyter community but not implemented yet: if the widgets' state was saved together with the notebook, and the widget state could be recreated when the notebook is loaded, this problem could be addressed.

DOCUMENTING A SOFTWARE LIBRARY

karabo_data is a Python library [10] that provides a convenient API for accessing data stored in files, working with the specific layout of HDF5 files created by EuXFEL's data acquisition system. It also offers some support for processing the data, particularly assembling images from X-ray pixel detectors composed of several separate modules. The library is open source [18], and is used by both external user groups and research groups within the facility.

Jupyter notebooks form an important part of the documentation for *karabo_data*, presenting executable examples for various use cases. Compared to presenting examples as scripts, notebooks allow for richer annotation, structuring the document using headings, and showing output such as plots inline. It is also easy for the library developers to validate a new feature interactively in a notebook, and then refine this into an example document.

Online documentation for *karabo_data* (https://karabo-data.readthedocs.io/en/latest/) is built with *Sphinx*, and *nbsphinx* [14] is used to integrate notebooks seamlessly into this, including rich output.

nbval [19] is also used to execute the example notebooks as part of the library's test suite. If running the examples produced an error, this would be highlighted as a failing test. This does not replace the test suite, but provides an extra layer of assurance based on realistic use cases.

Documentation in Jupyter notebooks can further exploit the Binder service [20] to provide the documentation notebooks as *live and executable* notebooks that users can execute in an anonymous short-lived environment in the cloud: being able to experiment interactively with a library to explore its capabilities, and to start from ready-made simple examples, can be a very effective way of learning about it quickly. To use such interactive documentation, only a web browser is required (no software installation) as the Binder service creates the required software environment on demand. A simple prototype of notebook execution through Binder is available in reference [3]; whereas a real-world example of executable documentation in computational science can be found, for example, in reference [21].

JupyterHub AND GENERIC RESEARCH ENVIRONMENTS

Jupyter notebooks are increasingly popular with facility staff and facility users. The case studies above describe some of the infrastructure and set-ups. For generic data analysis in Jupyter notebooks, the facility provides *karabo_data* (see above) as a Python module to provide convenient access to EuXFEL data files. This library makes facility data available as objects from standard Python data science libraries, such as *numpy* arrays, *pandas* data series and dataframes, and *xarray* arrays [22].

A JupyterHub installation is provided [23]: A JupyterHub instance appears to the user as a website from which notebooks can be created, opened, used and saved. Typically this is connected to the user's local account at the facility. For the JupyterHub at DESY/EuXFEL, users authenticate with their normal account, and have access to files in their home directory and to their experiment data.

For compute-intensive notebooks, it is possible to allocate dedicated nodes with user-specified hardware configuration (e.g. GPUs) from the Maxwell computer cluster to a running JupyterHub session. Substantial resources of the cluster can be consumed directly from the notebook for example through slurm-magic and dask-clusters. For interactive sessions that are dominated by the users contemplating the code or the output from a computation, a JupyterHub session can be started on a shared node.

As the JupyterHub installation appears as a website, it can be used remotely from anywhere on the Internet: this is of particular value as due to the size of data sets created, the analysis needs to take place remotely.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-TUCPR02

VISION FOR EUROPEAN OPEN SCIENCE CLOUD

As part of the Photon and Neutron Science Open Science Cloud project PaNOSC [24] we are working towards a data analysis framework that allows remote interactive data analysis of selected data sets over the Internet. A backbone of this vision are Jupyter notebooks that encapsulate the particular analysis proceduces for different types of experiments, and which can be saved but also re-executed through access points in the European Open Science Cloud (EOSC).

An important use case for this framework is the reproducible re-execution of data analysis for publications which are, for example, based on research facility data: We suggest to describe the analysis in a Jupyter notebook, and archive the notebook with required software as metadata together with the (raw or preprocessed) data for the publication. Once fully developed, the EOSC could provide access to a JupyterHub instance in which scientists (and in principle the interested public) can find and access select publications, and then access and re-execute the notebooks of the publication: this execution would re-create the central statements of the paper, such as tables, numbers and figures.

This reproducibility is of particular value to effective scientific research: if the study can be reproduced in a cloud environment within minutes of finding it on the EOSC portal, and all required analysis steps are transparently available – for example through such a notebook – then it is a matter of minutes or hours for scientists to modify and extend and thus re-use this study to work towards new research insights. For the vast majority of existing published results it can take months to reproduce the results in the absence of such metadata and infrastructure.

For the technical realisation of this vision, we need to execute those notebooks in a computational environment that has either been preserved from when it was created (such as a Docker or Singularity container), or in an environment that can be recreated on the fly (as is done in the BinderHub [4, 20] approach). Furthermore, the original data set needs to be accessible from this environment: small data sets can be transferred to where the computation is to take place, but for larger data sets the only feasible option is to perform the computation physically close to the data set [25].

To turn this vision into reality – or in the first instance work towards a prototype to demonstrate it – we need buy-in from scientists to produce such reproducible analysis. In principle, there is a an understanding that any scientific work should be reproducible. In practice, this is not always the case: journals have not enforced the publication of detailed analysis steps and authors have not always pushed to reveal their particular analysis algorithms and workflows with all details. However, increasingly, journals and research councils push for the publication of all data analysis steps (see for example Nature's policy on reproducibility requirements for new submissions [26] which essentially makes it compulsory for every author to be able to reveal their precise analysis steps on request by another reader). As stated in [27], programs used for data analysis implement algorithms which contain the scientific models. While models can be described in scientific articles, only the implementation describes the management of all corner-cases and is hence needed for reproducibility: only open-source software allows full reproducibility.

It is furthermore required that scientists are technically able and have the resources to express their analysis in Jupyter notebooks. For most scripted processes, this should be possible (the notebook may just call the script in the most extreme scenario). In cases the scientist need access to specific data analysis tools and corresponding graphical user interfaces, an alternative solution is proposed by the PaNOSC project which provides remote access to graphical terminals.

From the user perspective of such a service, it will only be necessary to operate a web browser to re-execute the data analysis: the required software installation and provision is done on the server side.

DISCUSSION AND SUMMARY

We have reported a number of different and overlapping use cases that exploit Jupyter notebooks and the associated tools from Project Jupyter. The notebook is an exciting technology, user interface and tool that is increasingly attracting users in academia, industry and commerce.

We discuss some of the challenges associated with projects moving as quickly as Jupyter.

We have highlighted some use cases of the Binder service [3, 4, 20], for example to provide interactive documentation and reproducible publications, but also interactive training environments that do not require software installation at the user side. While this offers exciting opportunities and opens up avenues for better and more effective science, there is currently no sustainable model to provide the required cloud compute resources. At the moment, the cloud computing resources for the mybinder service [20] are sponsored by a few companies, and the operation is carried out voluntarily by members of Project Jupyter. We hope that Binder [4] services will emerge as part of the European Open Science Cloud, or indeed – potentially with required authentication – from larger research institutions such as universities, facilities, research councils or publishers.

We also note that interactive plotting in notebooks is occasionally brittle: not all browsers support all packages, occasionally a feature works in the classic notebook but not in the new interface called JupyterLab.

In summary, we believe that there are many use cases where Jupyter notebooks and the Jupyter ecosystem as a computational tool and method can make research more effective. We have shared use cases and a vision for better science with better software infrastructure for the future.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-TUCPR02

ACKNOWLEDGEMENTS

We acknowledge financial support from the OpenDreamKit Horizon 2020 European Research Infrastructures project (#676541) and the Photon and Neutron Open Science Cloud (PaNOSC) project (#823852).

REFERENCES

[1] F. Pérez and B. E. Granger, "IPython: a system for interactive scientific computing," *Computing in Science and Engineering*, vol. 9, no. 3, pp. 21–29, May 2007.

[2] T. Kluyver *et al.*, "Jupyter Notebooks -- A Publishing Format for Reproducible Computational Workflows," in *Positioning and Power in Academic Pub-lishing: Players, Agents and Agendas*, F. Loizides and B. Schmidt, Ed. Amsterdam, Netherlands: IOS Press BV, 2016, pp. 87–90.

[3] H. Fangohr, "Jupyter notebook demo: basics, widgets and Binder, 10.5281/zenodo.3463132," https://github.com/fangohr/jupyter-demo, Sep 2019.

[4] Project Jupyter *et al.*, "Binder 2.0 - reproducible, interactive, sharable environments for science at scale," in *Proceedings of the 17th Python in Science Conference*. SciPy, 2018. doi:10.25080/majora-4af1f417-011

[5] K. Kelley, S. Abdalla, L. Geiger, S. Sturgis, J. Detlefs, and contributors, "nteract," https://nteract.io/.

[6] K. M. Mendez, L. Pritchard, S. N. Reinke, and D. I. Broadhurst, "Toward collaborative open data science in metabolomics using jupyter notebooks and cloud computing," *Metabolomics*, vol. 15, no. 10, Sep. 2019. doi:10.1007/s11306-019-1588-0

[7] "Code toolbox for SCS instrument at European XFEL," https://in.xfel.eu/gitlab/SCS/ToolBox

[8] G. M. Kurtzer, V. Sochat, and M. W. Bauer, "Singularity: Scientific containers for mobility of compute," *PLOS ONE*, vol. 12, no. 5, pp. 1–20, 05 2017. doi:10.1371/journal.pone.0177459

[9] S. Ritt, "Elog," https://elog.psi.ch/elog/.

[10] H. Fangohr *et al.*, "Data Analysis Support in Karabo at European XFEL," in *Proc. of 16th Int. Conf. on Accelerator and Large Experimental Control Systems (ICALEPCS'17)*, Barcelona, Spain, Oct 2017, pp. 245–252. doi:10.18429/JACoW-ICALEPCS2017-TUCPA01

[11] M. Reiser, Ph.D. dissertation, 2019.

[12] M. Kuster *et al.*, "Detectors and calibration concept for the european xfel," *Synchrotron Radiation News*, vol. 27, no. 4, pp. 35–38, 2014. doi.org/10.1080/08940886.2014.930809

[13] T. Kluyver, "nbparameterise," https://github.com/takluyver/nbparameterise

[14] M. Geier and contributors, "nbsphinx," https://github.com/spatialaudio/nbsphinx

[15] C. Fortmann-Grote *et al.*, "Start-to-end simulation of single-particle imaging using ultra-short pulses at the European X-ray Free-Electron Laser," *IUCrJ*, vol. 4, no. 5, pp. 560–568, Sep. 2017.

[16] M. Beg, R. A. Pepper, and H. Fangohr, "User interfaces for computational science: A domain specific language for OOMMF embedded in Python," *AIP Advances*, vol. 7, no. 5, p. 056025, May 2017. doi:10.1063/1.4977225

[17] Project Jupyter and community, "Jupyter widgets," https: //github.com/jupyter-widgets

[18] European XFEL, "karabo_data," https://github.com/European-XFEL/karabo_data

[19] D. Cortés-Ortuño *et al.*, "nbval," https://github.com/computationalmodelling/nbval

[20] Project Jupyter and community, "Binderhub," https://binderhub.readthedocs.io

[21] M. Beg and H. Fangohr, "discretisedfield python package," https://discretisedfield.readthedocs.io, 2019, for interactive execution look for "launch Binder" button in each section, for example section "Visualising the field using k3d".

[22] T. C. Corporation and Community, "xarray," https://xarray.pydata.org

[23] DESY/EuXFEL, "JupyterHub for Maxwell cluster," https://max-jhub.desy.de

[24] "Photon and neutron open science cloud (PaNOSC)," http://panosc.eu, 2019, funded under H2020 grant agreement 823852.

[25] B. Grenier and E. Fernandez, "Open data analysis with eosc-hub services," in *Book of Abstracts: Cloud Services for Synchronisation and Sharing – CS3 Workshop – INFN Rome January 2019*, 2019. https://indico.cern.ch/event/726040/contributions/3252075/

[26] "Data-access practices strengthened," *Nature*, vol. 515, no. 7527, pp. 312–312, Nov. 2014. http://www.nature.com/articles/515312a

[27] K. Hinsen, "A carrot not a stick," *Nature Physics*, vol. 15, p. 727, 2019. doi:10.1038/s41567-019-0627-0