Gouillart *et al. Adv Struct Chem Imag (2016) 2:18*
DOI 10.1186/s40679-016-0031-0

Advanced Structural and Chemical Imaging

RESEARCH Open Access

Analyzing microtomography data with Python and the scikit-image library

Emmanuelle Gouillart[1*], Juan Nunez-Iglesias[2] and Stéfan van der Walt[3]

Abstract

The exploration and processing of images is a vital aspect of the scientific workflows of many X-ray imaging modalities. Users require tools that combine interactivity, versatility, and performance. `scikit-image` is an open-source image processing toolkit for the Python language that supports a large variety of file formats and is compatible with 2D and 3D images. The toolkit exposes a simple programming interface, with thematic modules grouping functions according to their purpose, such as image restoration, segmentation, and measurements. `scikit-image` users benefit from a rich scientific Python ecosystem that contains many powerful libraries for tasks such as visualization or machine learning. `scikit-image` combines a gentle learning curve, versatile image processing capabilities, and the scalable performance required for the high-throughput analysis of X-ray imaging data.

Keywords: `Scikit-image`, Python, Image processing library, 3D image

Background

The acquisition time of synchrotron tomography images has decreased dramatically over the last decade, from hours to seconds [29]. New modalities such as single-bunch imaging provide a time resolution down to the nanosecond for radiography [41]. However, the time subsequently spent in processing the images has not decreased as much, so that the outcome of a successful synchrotron imaging run often takes weeks or even months to be transformed into scientific results.

Transforming billions of pixels and voxels to a few meaningful figures represents a tremendous data reduction. Often, the sequence of operations needed to produce these data is not known beforehand, or might be altered due to artifacts [31], or to an unforeseen evolution of the sample. Image processing necessarily involves trial and error phases to choose the processing workflow. Therefore, image processing tools need to offer at the same time enough flexibility of use, a variety of algorithms, and efficient implementations to allow for fast iterations while adjusting the workflow.

Several software applications and libraries are available to synchrotron users to process their images. ImageJ [1, 48] and its distribution Fiji [45] is a popular general-purpose tool for 2D and 3D images, thanks to its intuitive menus and graphical tools, and the wealth of plugins contributed by a vivid community [46]. Software specialized in analyzing synchrotron data is available as well, such as XRDUA [17] for diffraction images obtained in powder diffraction analysis, or for 3D images, commercial tools such as Avizo 3D software (TM), or ToolIP/MAVIkit [19] are appreciated for an intuitive graphical pipeline and advanced 3D visualization. Some synchrotrons have even developed their own tools for volume processing, such as Pore3D [9] at the Elettra facility. Alternatively, the use of a programming language gives finer control, better reproducibility, and more complex analysis possibilities, provided classical processing algorithms can be called from libraries—thereby limiting the complexity of the programming task and the risk of bugs. Matlab (TM) and its image processing toolbox are popular in the academic community of computer vision and image processing. The Python language is widely used in the scientific world and in synchrotron facilities. As a general-purpose language, Python is used in synchrotrons to control device servers [8, 15, 51], to access raw data

*Correspondence: emmanuelle.gouillart@nsup.org
[1] Surface du Verre et Interfaces, UMR 125 CNRS/Saint-Gobain, 93303 Aubervilliers, France
Full list of author information is available at the end of the article

Springer Open

© The Author(s) 2016. This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

of X-ray detectors [27], to reconstruct tomography volumes from radiographs [23, 32], and in data processing packages for macromolecular crystallography [3], azimuthal integration of diffraction data [4], or fluorescence analysis [50, 52].

`scikit-image` [54] is a general-purpose image processing library for the Python language, and a component of the ecosystem of Python scientific modules commonly known as Scientific Python [34]. Like the rest of the ecosystem, `scikit-image` is released under a permissive open-source license and is available free of charge. Most of `scikit-image` is compatible with both 2D and 3D images, so that it can be used for a large number of imaging modalities, such as microscopy, radiography, or tomography. In this article, we explain how `scikit-image` can be used for processing data acquired in X-ray imaging experiments, with a focus on microtomography 3D images. This article does not intend to be a pedagogical tutorial on `scikit-image` for X-ray imaging, but rather to explain the rationale behind the package, and provide various examples of its capabilities.

Overview and first steps

In this section, we provide a short overview of the typical use patterns of `scikit-image`, illustrated by short snippets of code. Since Python is a programming language, the user interacts with data objects and images through code, which is either entered and executed in an interactive interpreter, or written in text files (so-called scripts) that are executed.

Images are manipulated as numerical arrays, each with a single, uniform data type. This common format guarantees interoperability with other libraries and straightforward access to and interpretation of computer memory. The N-dimensional (2D, 3D, ...) numerical array object is provided by the `NumPy` module [53].

In image processing in Python, one of the first tasks then is to generate `NumPy` arrays, which is often achieved by reading data from files. We read one 2-dimensional image from a file and display it as follows:

```
from skimage import io
im = io.imread('figure_gallery.png')
io.imshow(im)
```

`skimage` is the name under which `scikit-image` is imported in Python code. Note that functions (such as `imread` that reads an image file, or `imshow` that displays an image) are found in thematic submodules of `skimage`, such as `io` for Input/Output.

A stack of 2D images, such as tomography slices generated by a reconstruction algorithm, can be opened as an image collection or a 3D array:

```
>>> from skimage import io

>>> image_collection = io.imread_collection('*.tif')
>>> image_3d = io.concatenate(image_collection)
>>> print(image_3d.shape)
(800, 1024, 1024)
```

Raw data formats can be opened using the `NumPy` functions `fromfile` (to load the array into memory) or `memmap` (to keep the array on disk). The following code creates an array from a raw image file of unsigned 16-bit integers with a header of 1024 bytes:

```
im = np.memmap('image.edf', shape=(2048, 2048),
     offset=1024, dtype=np.uint16)
```

For every raw data specification, it is thus very easy to write a reader using `np.memmap` (see for example https://github.com/jni/python-redshirt). `hdf5` files are accessed using modules such as `h5py`, `pytables`.

`scikit-image` has a simple Application Programming Interface (API), based almost exclusively on functions. Most functions take an image (i.e., a multi-dimensional array) as an input parameter:

```
>>> from skimage import filters
>>> im_gaussian = filters.gaussian(im)
```

Optional parameters can be passed as Python *keyword arguments*, in addition to the image parameter.

```
>>> im_gaussian = filters.gaussian(im, sigma=3,
     mode='wrap')
```

A few functions require several arrays to be passed, such as the watershed segmentation algorithm that takes as parameters the image to be segmented, and an image of markers from which labels are propagated:

```
>>> from skimage import morphology
>>> labels = morphology.watershed(im, markers)
```

Therefore, the image processing workflow can be seen as a directed graph (a richer structure than a linear pipeline), where nodes are image-shaped arrays, and edges are functions of `scikit-image` transforming the arrays (see Fig. 1).

Most functions transparently handle 2D, 3D, or even higher-dimensional images as arguments, so the same

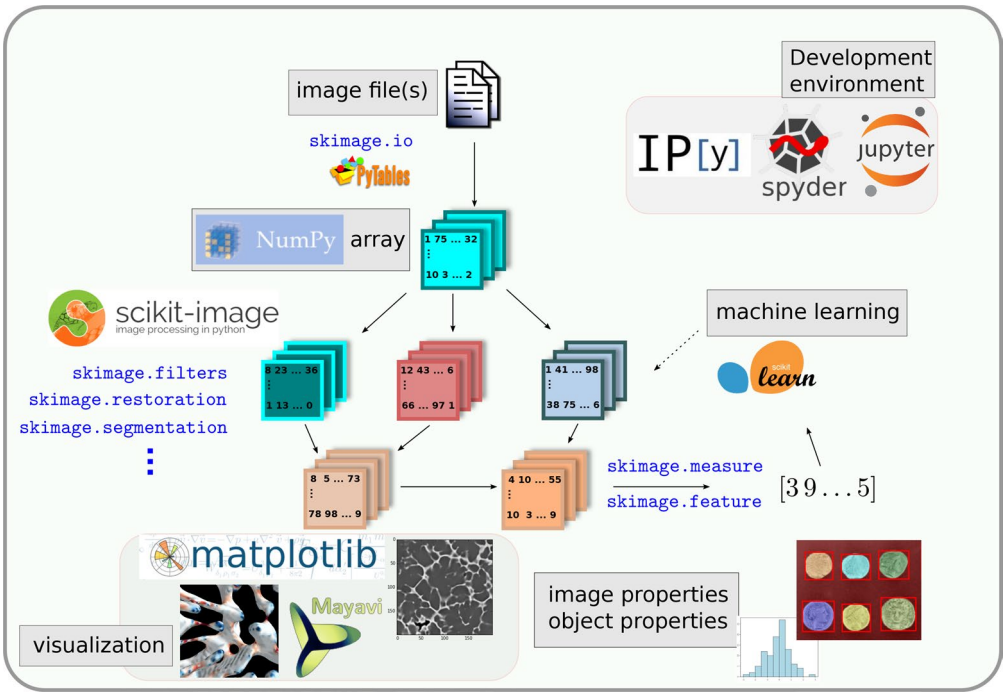


Fig. 1 `scikit-image` and the Scientific Python ecosystem. Images are opened from files as `NumPy` arrays. Functions of `scikit-image` transform image arrays into other arrays with the same dimensions, or into arrays of numbers corresponding to features of the image. The output of scikit-image functions can be passed to other Python modules relying on `NumPy` arrays, such as `SciPy` or `scikit-learn`. Image-shaped arrays are transformed into visualizations with `matplotlib` (2D) or `Mayavi` (3D). A variety of environments is available for code development and execution, from classical IDEs to Jupyter notebooks

functions can be used to process tomography, microscopy, or natural images. The rest raise an error when passed a 3D argument:

```
>>> filters.prewitt(im)
[...]
ValueError: The parameter 'image' must be a 2-↩
    dimensional array
```

However, the proportion of functions supporting 3D images is always increasing, thanks to the many contributors to the library.

While the majority of functions return processed images, returns can also be numerical value(s) such as pixel coordinates of objects of interest or statistical information about the image:

```
>>> from skimage import exposure
>>> counts, bins = exposure.histogram(im)
>>> counts.shape
(256,)
```

The Python ecosystem

The benefits of `scikit-image` for image processing come not only from the features of the package alone, but also from the rich environment surrounding scientific Python [34, 38]. Figure 1 illustrates how several components of this ecosystem combine into a sophisticated image processing workflow.

NumPy arrays are the cornerstone of the Scientific Python ecosystem, and of `scikit-image` operations in particular. NumPy "one-liners" include cropping or downsampling an image, or retrieving pixels corresponding to a given label in a segmentation. To illustrate the compactness of NumPy code, consider modifying pixel values below a threshold. This operation can be written as

```
im[im < 0.5] = 0
```

exploiting the ability to index arrays with boolean arrays, also called *masking*. `NumPy` uses memory sparingly and avoids making new copies of arrays whenever possible, an important requirement when dealing with the gigabyte-sized images of tomography. For example, cropping a subvolume as follows does not create a copy of the original array

```
sub_volume = im[100:-100, 100:-100, 100:-100]
```

but instead refers to the correct memory offsets in the original.

Interpreter and development environment While several interpreters are available to execute Python instructions and scripts interactively, the most popular in the scientific world is IPython [37, 44]. IPython is an advanced interpreter, which integrates syntax highlighting, text auto-completion, a debugger, introspection and profiling methods, and online help. Several integrated development environments (IDEs) come bundled with IPython, together with other components such as a text editor. Notable examples include Spyder (Fig. 2), PyCharm, and Visual Studio Code.

The Jupyter notebook [26] is a web application that grew out of the IPython project. Jupyter notebooks provide an interactive development environment within a web browser, where live code can be enriched by explanatory text, equations, and visualizations (Fig. 3). Jupyter notebooks render directly as webpages on GitHub, making them a straightforward tool to publish online a script and its output. As of July 2016, more than 500,000 Jupyter notebooks were posted on GitHub, demonstrating their wide adoption by the community as workflow-sharing tools (http://archive.ipython.org/media/SciPy2016JupyterLab.pdf).

Visualization libraries Visualizing images is an important component of the image processing workflow, used to inspect the final result and to adjust the parameters of intermediate processing operations. `matplotlib` [24] is the most popular 2D plotting library of the Python ecosystem. It can be used to visualize 2D data such as color or grayscale images, and 1D data such as contour lines, outlines of segmented regions, histograms of gray levels. Although `matplotlib` has simple 3D plotting capabilities, we recommend using the `mayavi` module [42] for applications requiring advanced 3D visualization, such as tomography. `mayavi` is based on the VTK toolkit. It exposes a simple API for visualizing data passed as `numpy` arrays. For example, visualizing the surface of binary data can be written as

```
# synthetic binary array from skimage.data
im = data.binary_blobs(length=400, n_dim=3).↩
    astype(np.uint8)
# visualization
from mayavi import mlab
mlab.contour3d(im)
mlab.outline()
```

(see Fig. 4 for the resulting visualization).

For more advanced visualizations, a large majority of VTK capabilities can be accessed through mayavi's pipeline API. `mayavi` offers a good trade-off between simplicity of use for common operations, and accessibility to more sophisticated capabilities such as responsive visualizations.

Advanced toolkits for signal processing and data science `scikit-image` is only one Python module that can be used for data processing, among many others. A very popular module is `scikit-learn` [36], a Python module for machine learning using `NumPy` arrays. Local features of an image (such as local statistics of gray levels, or geometric points of interest) or features of segmented objects (e.g., geometrical and intensity characteristics of segmented particles) can be extracted with functions from `skimage.feature` (see Fig. 1). It is then possible to use a *classification* algorithm from `scikit-learn` to label pixels (a segmentation task) or to classify whole images or objects that have already been segmented. The near-universal use of `NumPy` arrays ensures the interoperability between these packages, so that just a few lines of code are sufficient to create these sophisticated workflows.

The modularity of the Scientific Python ecosystem may be confusing at first sight, but the core modules of this ecosystem are almost perfectly compatible, thanks to the shared use of NumPy arrays and common development practices (although they are developed in parallel by different teams). Several "distributions," such as Anaconda or Canopy, bundle together the most popular libraries, including `scikit-image`.

Image processing capabilities

Capabilities `scikit-image` offers most classical image processing operations, such as exposure and color adjustment, filtering, segmentation, feature extraction, geometric transformations, and measurements of region characteristics. In addition to common operations, some advanced algorithms are also implemented; a selection of which is illustrated in Fig. 5. In the following, we briefly illustrate how `scikit-image` can be used for some typical image processing tasks encountered when analyzing tomographic images: denoising, mid-range feature detection, segmentation, and measurement of region properties. For the sake of brevity, other tasks such as contrast manipulation or geometric transformations are not described here; the interested reader is referred to the documentation of `scikit-image`.

Tomographic images often suffer from artifacts or poor signal-to-noise ratio. Therefore, *denoising* data is often the first step of an image processing workflow. Several denoising filters are available for restoring these images, ranging from general-purpose median and bilateral filters to those more suited to specific applications. For example, total-variation denoising [12, 20] is ideal for restoring piecewise-constant images (see Fig. 5a), such as images with a small number of phases encountered in materials

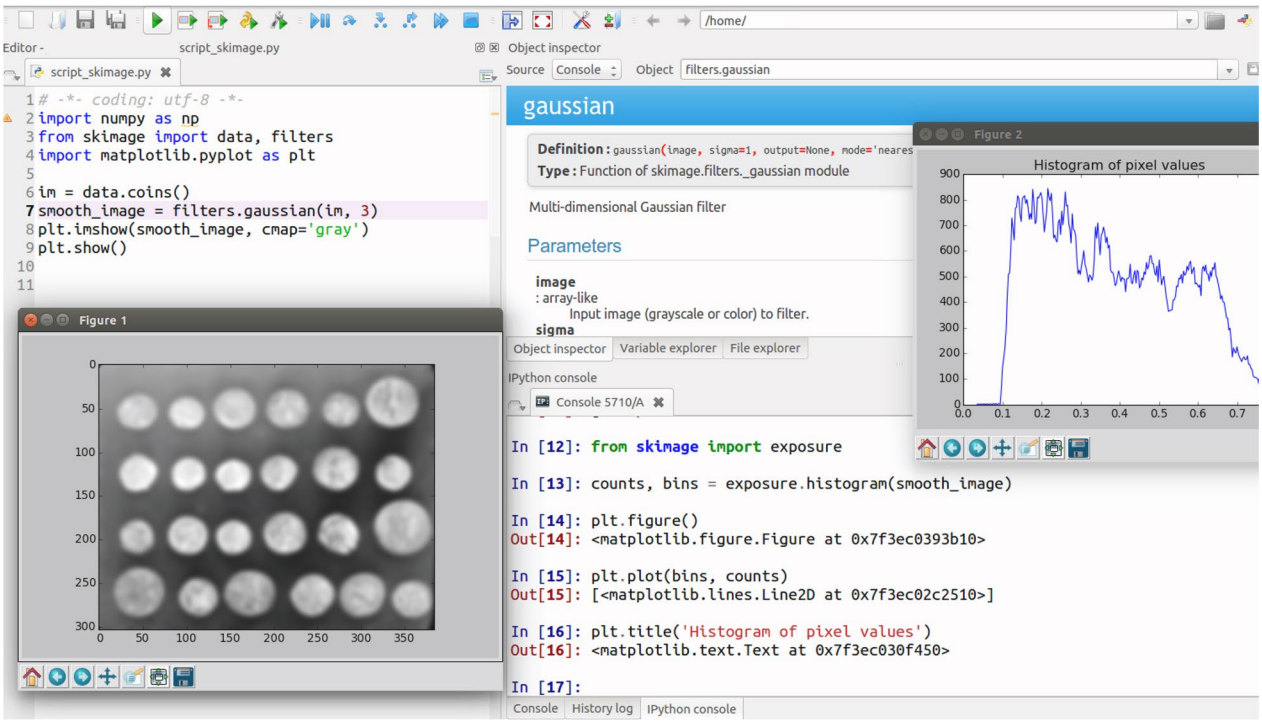


Fig. 2 The Spyder IDE integrates a text editor (with syntax highlighting), the IPython interpreter, as well as a panel for code introspection (online help, variable explorer, ...)

science [7]. Conversely, images with a fine-grained texture are better preserved with non-local means denoising, a patch-based algorithm [10] (see Fig. 5a).

Detecting the presence of objects or extracting pixels corresponding to objects (a task known as segmentation) is an important task of image analysis for medical or materials science applications. `scikit-image` offers a wide variety of functions for *detecting geometrical features* of interest in an image. In order to detect thin boundaries, the ridges of an image can be identified as regions for which the leading eigenvalue of the local Hessian matrix is high (see Fig. 5b). In the Fourier space, peaks in 2D Bragg diffraction patterns can be extracted using blob detection methods [4], such as the Laplacian of Gaussian method (see Fig. 5b).

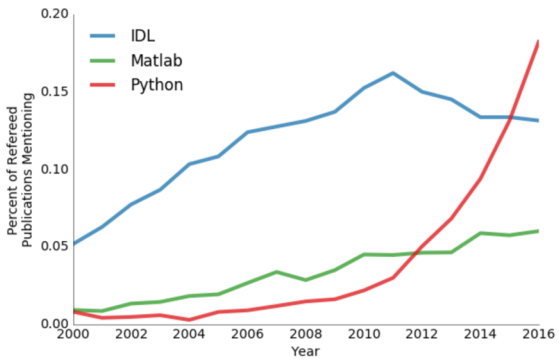
Segmentation of regions of interest can be achieved using one of the various strategies, depending on the characteristics of the image. Images with a clear contrast between regions can be segmented automatically, thanks to several thresholding algorithms, including an adaptive local thresholding algorithm aimed at images with contrast variations. Super-pixel algorithms [2, 18] create an over-segmentation of images in super-pixels, by grouping pixels that are close together both in color- and spatial distance (see Fig. 5c). Region-growing algorithms, such as the morphological watershed or the random walker [22], propagate the labels of user-defined markers through the image (see Fig. 5c). The active contour algorithm [25] fits snake contours to features of the image, such as edges or high-brightness regions.

Following segmentation, the characteristics of labeled regions (particles, porosities, organs, ...) resulting from a segmentation can be measured using the `measure` submodule. The different connected components (e.g., bubbles or non-touching particles) of a binary image are labeled with the `measure.label` function. Properties of labeled regions such as size, extent, center of mass, or mean intensity value are accessed with `measure.regionprops` (see Fig. 5d). Local characteristics of a region can be retrieved as well: Fig. 5d shows how the local diameter of open porosity is measured by combining a skeletonization of the porosity channels, and the distance transforms to the other phase measured on the skeleton.

Performance Given the large size of tomography datasets, the execution speed of image processing operations is of critical concern. `scikit-image` relies mostly on calls to `NumPy` operations, of which most are performed in optimized compiled code (C or Fortran). Performance-critical parts of `scikit-image` that cannot call efficient `NumPy` code are implemented in `Cython`. `Cython` [5] is an extension of the Python language that supports

```
In [8]: def trendlines(queries, norm=False):
            fig, ax = plt.subplots()
            for lang in languages:
                counts = queries[lang]
                x = counts[:, 0]
                y = np.copy(counts[:, 1])
                if norm:
                    y /= counts[:, 2]
                ax.plot(x, y * 100, label=lang, lw=4, alpha=0.8)
            ax.set_xlim(np.min(x), np.max(x))
            ax.set_xlabel('Year')
            ax.set_ylabel('Percent of Refereed\nPublications Mentioning')
            ax.legend(loc='upper left', frameon=False)
            remove_chartjunk(ax, ['top', 'right'])

In [9]: trendlines(results, norm=True)
        plt.savefig('python-vs-matlab-vs-IDL-in-astro.pdf')
```



There you have it: some time in early 2015, Python overtook IDL as the most mentioned (and probably the most used) programming language in astronomy!

Fig. 3 The Jupyter notebook allows mixing of computer code (*top*), plot and text output (*middle*), and free-form narrative text (*bottom*). This makes it ideal to record and report code-based analyses. Screenshot from [33]

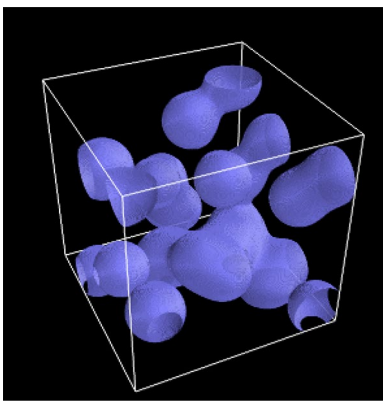


Fig. 4 Simple 3D visualization realized with Mayavi

explicit type declarations, and is compiled directly to C. Therefore, the performance of `scikit-image` can be close to the one of the libraries written in a compiled language such as C++ or Java. For example, computing the watershed segmentation of a 2000 × 2000 array of floats into 1000 regions took about 1 s using 1 CPU of an off-the-shelf laptop, and 10 s for a 256 × 256 × 256 image segmented into 2000 regions. Similar timescales were obtained with `mahotas`, a Python package implemented exclusively in C++ [14] (with a slight advantage for `mahotas`).

However, basic `scikit-image` code runs on a single core. Computing workstations and servers used for X-ray imaging typically have several tens of cores. Parallelization of the computing workflow can be achieved in multiple ways. The most trivial parallelization scheme consists of applying the same workflow to different images, on different cores. However, finer-grained parallelization is preferable when prototyping the processing workflow.

An easy solution consists in dividing an image into smaller images (with or without overlap, depending on the operation), and to apply the same operation on the different sub-images, on different cores. Creating overlapping chunks is easy with the dedicated function `view_as_windows` (or `view_as_blocks` for contiguous non-overlapping chunks):

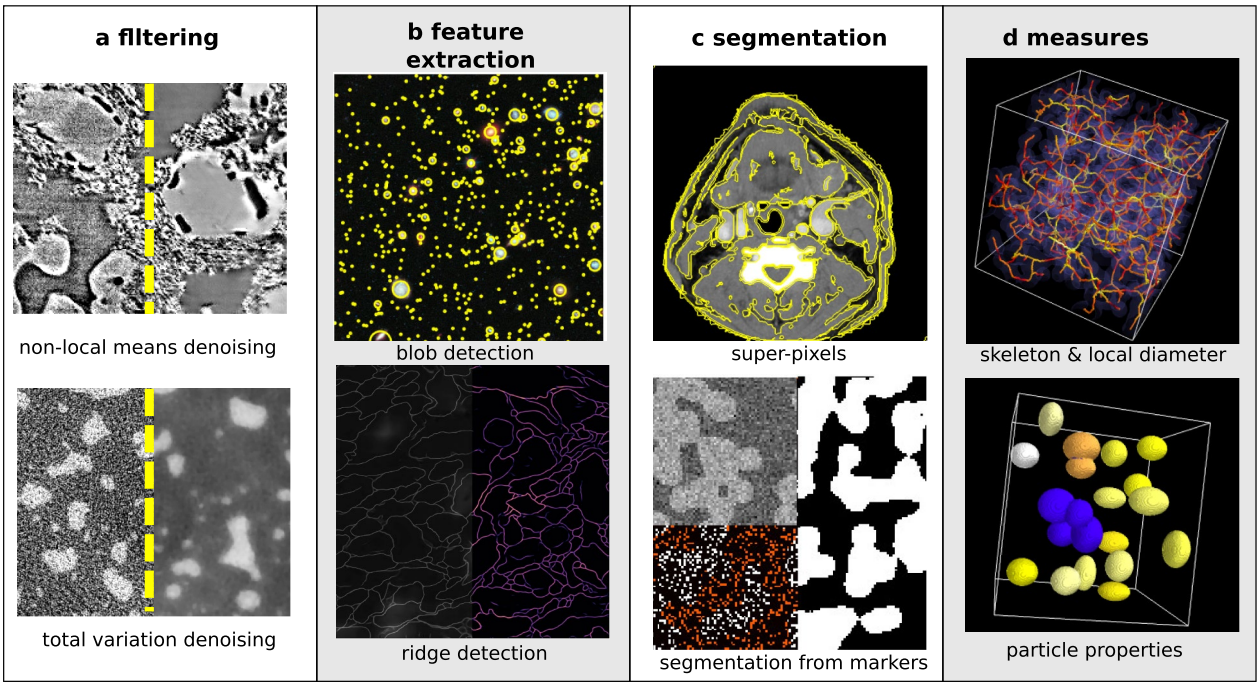


Fig. 5 Typical image processing operations with **`scikit-image`**. Data are synthetic, unless stated otherwise. **a** Filtering—*Top* non-local means denoising of an image with a fine-grained texture, acquired by in situ synchrotron microtomography during glass melting [21]. *Bottom* total-variation denoising of an image with two phases, corresponding to phase-separating silicate melts observed by in situ tomography [7]. **b** Feature extraction—*Top* Hubble deep field (NASA, public domain), blob detection using the Laplacian of Gaussian method. *Bottom* ridge detection using the leading eigenvalue of the Hessian matrix, neuron image from CREMI challenge (https://cremi.org/data/). **c** Segmentation—*Top* super-pixel segmentation of a CT slice of the human head [13], using Felzenszwalb's algorithm [18]. *Bottom* random walker segmentation (*right*) of noisy image (*top-left corner*), using histogram-determined markers (*bottom-left corner*). **d** Measures—*Top* visualization of local diameter (color-coded on the skeleton curve) of an interconnected phase (represented in *violet*). *Bottom* particles color-coded according to their extent

```
>>> from skimage import util, data
>>> im = data.camera()
>>> im.shape
(512, 512)
>>> # chunks of size 134 with 8-pixel overlap
>>> chunks = util.view_as_windows(im, (134, ↩
    134), step=126)
>>> chunks.shape
(4, 4, 134, 134)
```

The `joblib` library enables easy parallel processing. Looping over the different blocks, and dispatching the computation over several cores, is realized with the following syntax:

```
>>> from joblib import Parallel, delayed
>>> filtered_chunks = Parallel(n_jobs=4)(↩
    delayed(filters.gaussian)(chunks[i, j]) for↩
     i in range(4) for j in range(4))
```

`scikit-image` also offers experimental support for a more integrated parallel processing pipeline, thanks to the `dask` [43] module:

```
filtered_im = util.apply_parallel(filters.↩
    gaussian, im, depth=8)
```

The size of chunks is determined automatically from the number of available cpus, or can be specified by the user.

Caching provides another tool to speed up data analysis. A situation that often arises is that, while prototyping a workflow, scripts (containing the image processing pipeline) are run several times to experiment with parameters. `joblib` provides a caching mechanism that avoids the repetition of function calls, if their arguments have not changed:

```
>>> from skimage import filters, data
>>> from joblib import Memory
>>> mem = Memory(cachedir='/tmp/joblib')
>>> median = mem.cache(filters.median)
>>> im = data.camera()
>>> filtered_im = median(im, np.ones((3, 3)))
________________________________
[Memory] Calling skimage.filters.rank.generic.↩
    median...
median(array([[156, ..., 152],
       ...,
       [121, ..., 111]], dtype=uint8), array([[↩
            1.,  1.,  1.],
       [ 1.,  1.,  1.],
       [ 1.,  1.,  1.]]))
_____________________________median - 0.0s, 0.0min
>>> filtered_again = median(im, np.ones((3, 3))↩
    )
>>> # the above call did not trigger another ↩
    evaluation
```

Finally, we note that the biggest performance improvements often come from the improvement of algorithms (as opposed to computing architectures only). For example, non-local means denoising [10] is a costly operation, since it requires several nested loops, on all pixels and on neighboring patches to be compared with the pixel-centered patch. Thanks to the implementation of a more recent algorithm [16] that modifies the internal organization of loops, it was possible to improve the execution time by a factor of roughly ten times. The large size of the `scikit-image` community makes it likely for algorithmic improvements to be discussed regularly. During the code review process, a close watch is also kept on memory consumption, since for large image sizes, transfers between computer memory (RAM) and CPU cache are often a serious performance bottleneck.

Documentation

The quality of software documentation is (perhaps especially) important in software aimed at scientists. `scikit-image` users have access to several kinds of documentation. All functions are documented using the NumPy documentation standard [35], which is universal across all major Scientific Python packages. The standard includes a description of all input and output variables and their data types, together with explanations of what each function does and how to use it. Function documentation is accessible online or within the development environment itself (IPython, Spyder, Jupyter Notebook...).

In addition, a graphical gallery of examples (http://scikit-image.org/docs/dev/auto_examples/), part of which is displayed in Fig. 6, showcases graphical examples of common image processing operations. The examples are organized as an array of thumbnails with a short title (see Fig. 6 left). These thumbnails link to the webpage of the corresponding example, which features a mini-tutorial on the image processing method, the code needed to run the example, and the figure generated by the example. Since the graphical gallery is an efficient way to inform users about the features of `scikit-image`, every new feature integrated in the package must include an example for the gallery. Longer tutorials and a more narrative documentation is available as well in the online User Guide of `scikit-image`. The User Guide explains in particular "big picture," foundational aspects of `scikit-image`, such as its use of `NumPy` arrays as images, or how the package interacts with other parts of the scientific Python ecosystem.

Finally, tutorials on `scikit-image` are available in various places, either as YouTube videos, or in the SciPy Lecture Notes [55], a comprehensive online book of Scientific Python tutorials.

Development and use of scikit-image

Who uses scikit-image Estimating the number of active users of an open-source package is a difficult task. Download statistics, for example, largely overestimate the number of active users, all the more if the package is bundled with others in a software distribution, such as Anaconda or Canopy. A view closer to reality can be obtained by analyzing the statistics of visits of the online help, available on the project website. As of the first half of 2016, 20,000 unique visitors visited the `scikit-image` website every month at http://scikit-image.org/, from 138 countries.

The `scikit-image` paper of 2014 [54] has been cited by 120 research works (as of August 2016, according to Google Scholar), among which are studies that used X-ray imaging in fields such as medical imaging [6, 30, 49], materials science [7], or geoscience [47].

Development process `scikit-image` is developed by a diverse team of volunteers. More than 170 individuals have contributed to the package. The large number of developers and users is key to project's sustainability. The development process takes place on GitHub https://github.com/scikit-image/scikit-image, where users and developers propose and discuss new contributions, report bugs, or submit ideas for improvements. A release cycle of one or two releases every year ensures that new features are propagated to users on a regular basis.

Discussion—current limitations and challenges

While we emphasize the assets of `scikit-image` for processing X-ray images, one should be aware of current limitations.

Speed of execution Although `scikit-image` approaches the speed of execution of compiled (C++, Java) code, it cannot reach the performance of code optimized for the GPU, or the hand-tuned CPU-specific optimizations found in OpenCV [40]. At the moment, `scikit-image` is not the best tool for ultrafast computations where the workflow is known beforehand, simple, and stable. However, it is an excellent tool for exploring image data interactively and testing different algorithms—an important component of data processing in scientific work, and its speed of execution is sufficient for processing gigabyte-sized tomographic images in seconds to minutes. Moreover, the multiprocessing capability of `scikit-image` is likely to improve in the near future.

3D compatibility Currently, about two-thirds of `scikit-image` functions transparently handle 2D or 3D arrays, with the remainder limited to 2D analysis, often unnecessarily. Improved support for 3D and higher-dimensional volumes is on the project roadmap.

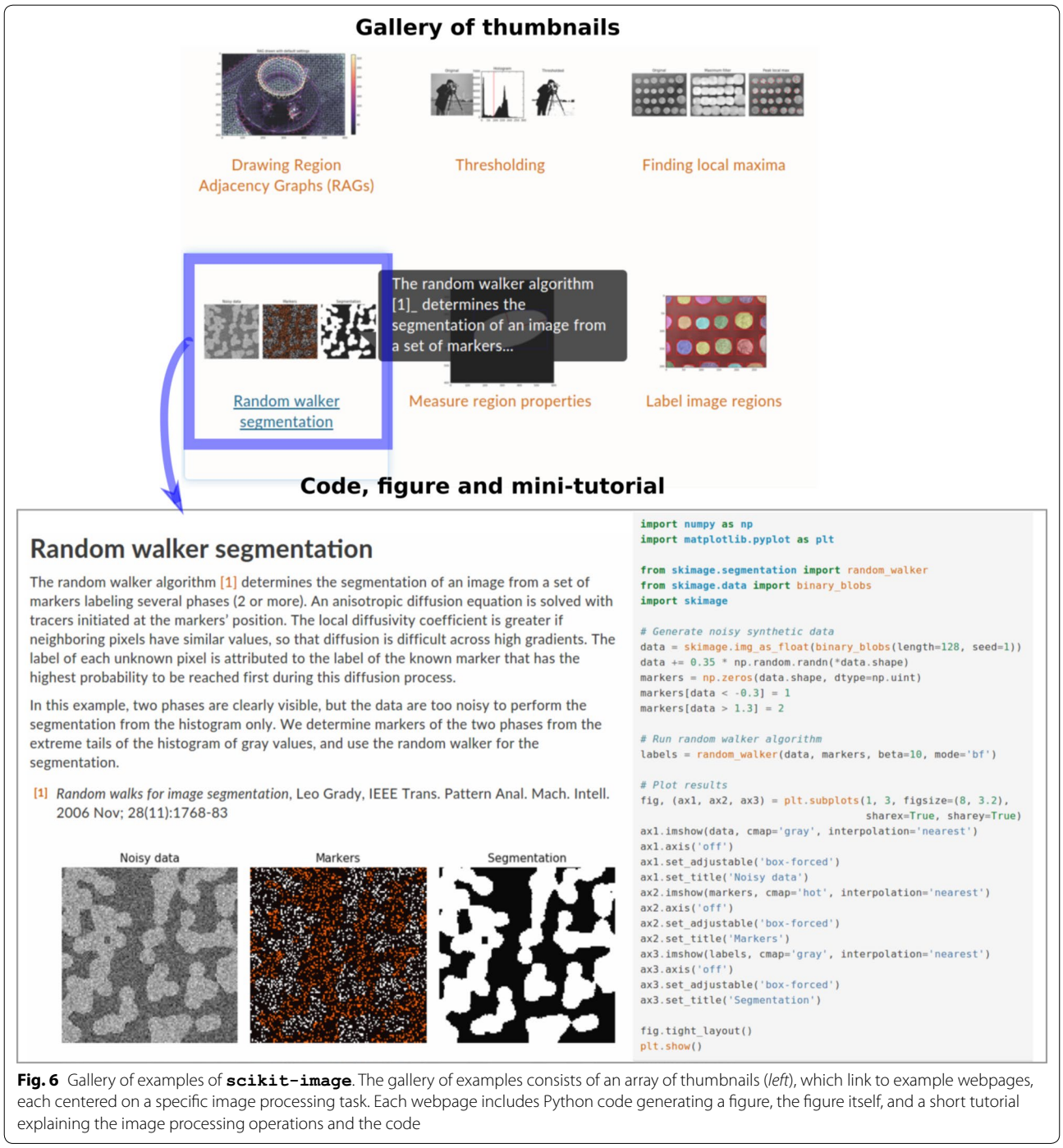


Fig. 6 Gallery of examples of **`scikit-image`**. The gallery of examples consists of an array of thumbnails (*left*), which link to example webpages, each centered on a specific image processing task. Each webpage includes Python code generating a figure, the figure itself, and a short tutorial explaining the image processing operations and the code

Documentation for domain-specific applications Some image processing libraries or applications address a specific scientific domain, such as CellProfiler [11, 28] for biological images. The documentation of such projects often showcases examples that are close to the experience of the targeted community. Since `scikit-image` is application-agnostic, applications such as tomographic imaging are not mentioned in detail in the documentation of `scikit-image`. A possible improvement would be to write comprehensive tutorials addressing specific communities, and to refer to these tutorials from the main `scikit-image` documentation.

Getting started

Scientists interested in experimenting with `scikit-image` are invited to read the installation instructions at http://scikit-image.org/docs/stable/install.html. `scikit-image` can be installed either bundled in a Scientific Python distribution, such as Anaconda (conda command line) or Canopy, or stand alone (along with its dependencies) using a installer/packager such as pip or Ubuntu's Aptitude.

The Getting Started section of the online User Guide (http://scikit-image.org/docs/dev/user_guide/getting_started.html) provides a good launch pad for beginners, and gently leads into other sections of the user guide. A gallery of examples (http://scikit-image.org/docs/dev/auto_examples/) lets users find applications close to their needs. Although most examples in the gallery use 2D images, many are applicable to 3D images as well.

Assistance on matters not covered by the documentation is provided on the dedicated mailing-list scikit-image@googlegroups.com or on Stack Overflow http://stackoverflow.com/questions/tagged/scikit-image.

Conclusions

`scikit-image` offers a wide variety of image processing algorithms, using a simple interface natively compatible with 2D and 3D images. It is well integrated into the Scientific Python ecosystem, so that it interfaces well with visualization libraries and other data processing packages. `scikit-image` has seen tremendous growth since its creation in 2009, both in terms of users and included features. In addition to the growing number of scientific teams that use `scikit-image` for processing images of various X-ray modalities, domain-specific tools are now using `scikit-image` as a dependency to build upon. Examples include `tomopy` [23] for tomographic reconstruction or `DIOPTAS` [39] for the reduction and exploration of X-ray diffraction data. It is likely that more application-specific software will benefit from depending on `scikit-image` in the future, since `scikit-image` strives to be domain-agnostic and to keep the function interface stable. On the end-user side, future work includes better integration of parallel processing capabilities, completion of full 3D compatibility, an enriched narrative documentation, speed enhancements, and expansion of the set of supported algorithms.

Authors' contributions

EG, JNI, and SvdW conducted the research and wrote the paper. All authors read and approved the final manuscript.

Author details

[1] Surface du Verre et Interfaces, UMR 125 CNRS/Saint-Gobain, 93303 Aubervilliers, France. [2] Victorian Life Sciences Computation Initiative, University of Melbourne, Carlton, VIC, Australia. [3] Division of Applied Mathematics, Stellenbosch University, Stellenbosch, South Africa.

Acknowledgements

The authors gratefully acknowledge the work of the contributors of `scikit-image`, and thank S. Deville and D. Vandembroucq for a careful reading of the paper and useful suggestions. E. Gouillart acknowledges the support of ANR project EDDAM ANR-11-BS09-027. CREMI (https://cremi.org/data/) is acknowledged for the membrane image of Fig. 5b.

Competing interests

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as potential competing interests.

Received: 26 August 2016 Accepted: 24 November 2016
Published online: 07 December 2016

References

1. Abràmoff, M.D., Magalhães, P.J., Ram, S.J.: Image processing with ImageJ. Biophotonics Int **11**(7), 36–42 (2004)
2. Achanta, R., Shaji, A., Smith, K., Lucchi, A., Fua, P., Süsstrunk, S.: SLIC superpixels compared to state-of-the-art superpixel methods. IEEE Trans Pattern Anal Mach Intell **34**(11), 2274–2282 (2012)
3. Adams, P.D., Afonine, P.V., Bunkóczi, G., Chen, V.B., Davis, I.W., Echols, N., Headd, J.J., Hung, L.-W., Kapral, G.J., Grosse-Kunstleve, R.W., et al.: PHENIX: a comprehensive python-based system for macromolecular structure solution. Acta Crystallogr D Biol Crystallogr **66**(2), 213–221 (2010)
4. Ashiotis, G., Deschildre, A., Nawaz, Z., Wright, J.P., Karkoulis, D., Picca, F.E., Kieffer, J.: The fast azimuthal integration Python library: pyFAI. J Appl Crystallogr **48**(2), 510–519 (2015)
5. Behnel, S., Bradshaw, R., Citro, C., Dalcin, L., Seljebotn, D.S., Smith, K.: Cython: the best of both worlds. Comput Sci Eng **13**(2), 31–39 (2011)
6. Blackledge, M.D., Collins, D.J., Koh, D.-M., Leach, M.O.: Rapid development of image analysis research tools: bridging the gap between researcher and clinician with pyOsiriX. Comput Biol Med **69**, 203–212 (2016)
7. Bouttes, D., Lambert, O., Claireaux, C., Woelffel, W., Dalmas, D., Gouillart, E., Lhuissier, P., Salvo, L., Boller, E., Vandembroucq, D.: Hydrodynamic coarsening in phase-separated silicate melts. Acta Mater **92**, 233–242 (2015)
8. Brookhaven National Lab: NSLS-II. http://nsls-ii.github.io/ (2016). Accessed Aug 2016
9. Brun, F., Mancini, L., Kasae, P., Favretto, S., Dreossi, D., Tromba, G.: Pore3D: a software library for quantitative analysis of porous media. Nucl Instrum Methods Phys Res Sect A Accel Spectrom Detect Assoc Equip **615**(3), 326–332 (2010)
10. Buades, A., Coll, B., Morel, J.M.: A non-local algorithm for image denoising. In: IEEE Computer Society Conference on computer vision and pattern recognition, 2005. CVPR 2005. vol. 2, pp. 60–65. IEEE (2005). doi:10.1109/CVPR.2005.38
11. Carpenter, A.E., Jones, T.R., Lamprecht, M.R., Clarke, C., Kang, I.H., Friman, O., Guertin, D.A., Chang, J.H., Lindquist, R.A., Moffat, J., et al.: CellProfiler: image analysis software for identifying and quantifying cell phenotypes. Genome Biol **7**(10), 100 (2006)
12. Chambolle, A.: An algorithm for total variation minimization and applications. J Math Imaging Vis **20**(1–2), 89–97 (2004)
13. ChumpusRex, Wikipedia: Typical screen layout of workstation software used for reviewing multi-detector CT studies. https://en.wikipedia.org/wiki/File:Ct-workstation-neck.jpg (2016). Accessed Aug 2016
14. Coelho, P.: Mahotas: open source software for scriptable computer vision. J Open Res Softw **1**, 1 (2013)
15. Coutinho, T.: PyTango. http://www.esrf.eu/computing/cs/tango/tango_doc/kernel_doc/pytango/latest/index.html (2016). Accessed Aug 2016
16. Darbon, J., Cunha, A., Chan, T.F., Osher, S., Jensen, G.J.: Fast nonlocal filtering applied to electron cryomicroscopy. In: 2008 5th IEEE International Symposium on biomedical imaging: from nano to macro, pp. 1331–1334. IEEE (2008). doi:10.1109/ISBI.2008.4541250
17. De Nolf, W., Vanmeert, F., Janssens, K.: XRDUA: crystalline phase distribution maps by two-dimensional scanning and tomographic (micro) x-ray powder diffraction. J Appl Crystallogr **47**(3), 1107–1117 (2014)

18. Felzenszwalb, P.F., Huttenlocher, D.P.: Efficient graph-based image segmentation. Int J Comput Vis **59**(2), 167–181 (2004)
19. Fraunhofer Institute for Industrial Mathematics ITWM: MAVI. http://www.itwm.fraunhofer.de/en/departments/image-processing/microstructure-analysis/mavi.html (2016). Accessed Aug 2016
20. Getreuer, P.: Rudin–Osher–Fatemi total variation denoising using split Bregman. Image Process Line **2**, 74–95 (2012)
21. Gouillart, E., Toplis, M.J., Grynberg, J., Chopinet, M.-H., Sondergard, E., Salvo, L., Suéry, M., Di Michiel, M., Varoquaux, G.: In situ synchrotron microtomography reveals multiple reaction pathways during soda-lime glass synthesis. J Am Ceram Soc **95**(5), 1504–1507 (2012)
22. Grady, L.: Random walks for image segmentation. IEEE Trans Pattern Anal Mach Intell **28**(11), 1768–1783 (2006). doi:10.1109/TPAMI.2006.233
23. Gürsoy, D., De Carlo, F., Xiao, X., Jacobsen, C.: TomoPy: a framework for the analysis of synchrotron tomographic data. J Synchrotron Radiat **21**(5), 1188–1193 (2014)
24. Hunter, J.D., et al.: Matplotlib: a 2D graphics environment. Comput Sci Eng **9**(3), 90–95 (2007)
25. Kass, M., Witkin, A., Terzopoulos, D.: Snakes: active contour models. Int J Comput Vis **1**(4), 321–331 (1988)
26. Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., et al.: Jupyter notebooks—a publishing format for reproducible computational workflows. In: Positioning and Power in Academic Publishing: Players, Agents and Agendas, p. 87 (2016). doi:10.3233/978-1-61499-649-1-87
27. Knudsen, E.B., Sørensen, H.O., Wright, J.P., Goret, G., Kieffer, J.: Fabio: easy access to two-dimensional x-ray detector images in python. J Appl Crystallogr **46**(2), 537–539 (2013)
28. Lamprecht, M.R., Sabatini, D.M., Carpenter, A.E., et al.: Cellprofiler^{TM}: free, versatile software for automated biological image analysis. Biotechniques **42**(1), 71 (2007)
29. Maire, E., Withers, P.: Quantitative x-ray tomography. Int Mater Rev **59**(1), 1–43 (2014)
30. Malan, D.F., van der Walt, S.J., Raidou, R.G., van den Berg, B., Stoel, B.C., Botha, C.P., Nelissen, R.G., Valstar, E.R.: A fluoroscopy-based planning and guidance software tool for minimally invasive hip refixation by cement injection. Int J Comput Assist Radiol Surg **11**(2), 281–296 (2016)
31. Marone, F., Münch, B., Stampanoni, M.: Fast reconstruction algorithm dealing with tomography artifacts. In: Proceedings of SPIE developments in X-Ray tomography VII, vol. 7804. International Society for Optics and Photonics, pp. 780410 (2010). doi:10.1117/12.859703
32. Mirone, A., Brun, E., Gouillart, E., Tafforeau, P., Kieffer, J.: The PyHST2 hybrid distributed code for high speed tomographic reconstruction with iterative reconstruction and a priori knowledge capabilities. Nucl Instrum Methods Phys Res Sect B Beam Interact Mater At **324**, 41–48 (2014)
33. Nunez-Iglesias, J., Beaumont, C., Robitaille, T.P.: Counting programming language mentions in astronomy papers (Late 2016 version). doi:10.5281/zenodo.163863. https://github.com/jni/programming-languages-in-astronomy/blob/1.1/programming-languages-in-ADS.ipynb (2016). Accessed Aug 2016
34. Oliphant, T.E.: Python for scientific computing. Comput Sci Eng **9**(3), 10–20 (2007)
35. Pawlik, A., Segal, J., Sharp, H., Petre, M.: Crowdsourcing scientific software documentation: a case study of the NumPy documentation project. Comput Sci Eng **17**(1), 28–36 (2015)
36. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al.: Scikit-learn: machine learning in Python. J Mach Learn Res **12**, 2825–2830 (2011)
37. Pérez, F., Granger, B.E.: IPython: a system for interactive scientific computing. Comput Sci Eng **9**(3), 21–29 (2007)
38. Perez, F., Granger, B.E., Hunter, J.D.: Python: an ecosystem for scientific computing. Comput Sci Eng **13**(2), 13–21 (2011)
39. Prescher, C., Prakapenka, V.B.: DIOPTAS: a program for reduction of two-dimensional x-ray diffraction data and data exploration. High Press Res **35**(3), 223–230 (2015)
40. Pulli, K., Baksheev, A., Kornyakov, K., Eruhimov, V.: Real-time computer vision with OpenCV. Commun ACM **55**(6), 61–69 (2012)
41. Rack, A., Scheel, M., Hardy, L., Curfs, C., Bonnin, A., Reichert, H.: Exploiting coherence for real-time studies by single-bunch imaging. J Synchrotron Radiat **21**(4), 815–818 (2014)
42. Ramachandran, P., Varoquaux, G.: Mayavi: 3d visualization of scientific data. Comput Sci Eng **13**(2), 40–51 (2011)
43. Rocklin, M.: Dask contributors: dask. http://dask.pydata.org/en/latest/ (2016). Accessed Aug 2016
44. Rossant, C.: Learning IPython for interactive computing and data visualization. Packt Publishing Ltd, Birmingham (2015)
45. Schindelin, J., Arganda-Carreras, I., Frise, E., Kaynig, V., Longair, M., Pietzsch, T., Preibisch, S., Rueden, C., Saalfeld, S., Schmid, B., et al.: Fiji: an open-source platform for biological-image analysis. Nat Methods **9**(7), 676–682 (2012)
46. Schindelin, J., Rueden, C.T., Hiner, M.C., Eliceiri, K.W.: The ImageJ ecosystem: an open platform for biomedical image analysis. Mol Reprod Dev **82**(7–8), 518–529 (2015)
47. Schlüter, S., Sheppard, A., Brown, K., Wildenschild, D.: Image processing of multiphase images obtained via x-ray microtomography: a review. Water Resour Res **50**(4), 3615–3639 (2014)
48. Schneider, C.A., Rasband, W.S., Eliceiri, K.W., et al.: NIH Image and ImageJ: 25 years of image analysis. Nat Methods **9**(7), 671–675 (2012)
49. Shen, W., Zhou, M., Yang, F., Yang, C., Tian, J.: Multi-scale convolutional neural networks for lung nodule classification. In: International Conference on Information Processing in Medical Imaging, pp. 588–599. Springer (2015). doi:10.1007/978-3-319-19992-4_46
50. Solé, V., Papillon, E., Cotte, M., Walter, P., Susini, J.: A multiplatform code for the analysis of energy-dispersive x-ray fluorescence spectra. Spectrochim Acta Part B At Spectrosc **62**(1), 63–68 (2007)
51. Sugandhi, R., Swamy, R., Khirwadkar, S.: Use of epics and python technology for the development of a computational toolkit for high heat flux testing of plasma facing components. Fusion Eng Des **112**, 783–787 (2016)
52. V. Armando Sole: PyMca. https://github.com/vasole/pymca (2016). Accessed Aug 2016
53. Van Der Walt, S., Colbert, S.C., Varoquaux, G.: The NumPy array: a structure for efficient numerical computation. Comput Sci Eng **13**(2), 22–30 (2011)
54. Van der Walt, S., Schönberger, J.L., Nunez-Iglesias, J., Boulogne, F., Warner, J.D., Yager, N., Gouillart, E., Yu, T.: scikit-image: image processing in python. PeerJ **2**, e453 (2014)
55. Varoquaux, G., Gouillart, E., Vahtras, O., scipy-lecture-notes contributors: Scipy lecture notes. http://www.scipy-lectures.org/ (2016). Accessed 10 Aug 2016

Submit your manuscript to a SpringerOpen journal and benefit from:

- ► Convenient online submission
- ► Rigorous peer review
- ► Immediate publication on acceptance
- ► Open access: articles freely available online
- ► High visibility within the field
- ► Retaining the copyright to your article

Submit your next manuscript at ► springeropen.com