Journal of
open research software

Hughes, A et al 2015 Scikit-spectra: Explorative Spectroscopy in Python. *Journal of Open Research Software,* 3: e6, DOI: http://dx.doi.org/10.5334/jors.bs

SOFTWARE METAPAPER

Scikit-spectra: Explorative Spectroscopy in Python

Adam Hughes[1], Zhaowen Liu[1] and M. E. Reeves[1]

[1] Dept. of Physics, The George Washington University, USA
hugadams@gwmail.gwu.edu, evelynzwliu@gmail.com, reevesme@gwu.edu

Scikit-spectra is an intuitive framework for explorative spectroscopy in Python. Scikit-spectra leverages the Pandas library for powerful data processing to provide datastructures and an API designed for spectroscopy. Utilizing the new IPython Notebook widget system, scikit-spectra is headed towards a *GUI when you want it, API when you need it* approach to spectral analysis. As an application, analysis is presented of the surface-plasmon resonance shift in a solution of gold nanoparticles induced by proteins binding to the gold's surface. Please refer to the scikit-spectra website for full documentation and support: http://hugadams.github.io/scikit-spectra/

Keywords: correlation spectroscopy; GUI; Python; IPython; Pandas; SciPy; scikits; spectroscopy; timeseries
Funding statement: Supported in part by the George Gamow Research Fellowship, Luther Rice Collaborative Research fellowship programs, the George Washington University (GWU) Knox fellowship and the GWU Presidential Merit Fellowship.

(1) Overview

Introduction

Spectroscopy, the study of the interaction of light and matter, is utilized as an experimental technique in chemistry (IR/Raman), physics (NMR/Xray), biology and nanotechnology (UVVis/circular dichroism), and many other scientific fields. Despite widespread interest, spectroscopy software development is not often a research focus; researchers traditionally rely on commercial software bundled with instrumentation, such as a benchtop spectrometer, or a Raman microscope. Such software is expensive and usually tailored to a particular research domain or application. Open-source solutions are less abundant, and also tend to be specialized.

Python has emerged as a swiss-army knife for scientific research, due in large part to a core group of scientific libraries known as SciPy[1], perhaps the most prominent of which are NumPy[2], Pandas[3] and IPython[4]. To integrate with the SciPy ecosystem, new libraries must be NumPy-compatible. For example, the scikit-image[1][5] library stores images as pure NumPy arrays, while Pandas' primary datastructures are directly subclassed from NumPy arrays. In regard to spectroscopy in Python, a handful of domain-specific, SciPy-compatible libraries are available; for example, NMRGlue[6] and PySpecKit[7] are great resources for nuclear magnetic resonance and astronomy applications, respectively.

Interoperability between Python's spectroscopy libraries is challenging, even when they are NumPy-compatible. The primary difficulty arises in storing metadata. Spectral data is tabular: a matrix of **n** spectra measured at **m** timepoints, or more generally, **m** variational points, with labeled rows (e.g. wavelength) and columns (e.g. time). Most would recognize this datastructure in an Excel Spreadsheet. Fortunately, tabular data is already well-supported in Python by the widely-used Pandas library. Pandas provides an intuitive, NumPy-friendly API for IO, plotting, statistical analysis and data manipulation. Unfortunately, it's not straightforward to simply repurpose Pandas for spectroscopy, since Pandas datastructures don't preserve arbitrary metadata, nor do they support conventional Python subclassing. Such obstacles reduce Pandas' applicability to spectral analysis.

Herein, scikit-spectra is presented, a Python library that provides generalized datastructures and APIs for explorative[2] spectroscop. Scikit-spectra overcomes the aforementioned Pandas metadata and subclassing obstacles to provide spectroscopy datastructures that behave identically to Pandas objects, leading to a much more intuitive framework for IO, manipulation and plotting of spectral data. The ways that scikit-spectra extends Pandas to suit the needs of spectroscopists include:

1. Nearest-neighbor slicing to index data based on an approximate range of values.
2. 2D and 3D contour, waterfall, auto-correlation, and other spectral plots seamlessly integrated into pandas' pre-existing plotting API.
3. Unit-aware indexing objects for easy unit conversions and integration with the plotting API.
4. IPython Notebook graphical user interfaces (GUIs) to expedite many common tasks such as resampling,

normalization, and plotting without ever leaving the notebook environment.

5. **Spectra** and **Spectrum** classes to replace the Pandas DataFrame and Series, respectively. These objects retain all of the functionality of their Pandas counterparts, and add capabilities such as persistence of metadata, the notion of baseline and reference spectra, and reversible spectral normalization: for example converting raw data into a transmission(T), percent transmission(%T), or absorbance(A) spectra.

Implementation and architecture

Scikit-spectra's core datastructures are the Spectrum and Spectra, which behave as if they were directly subclassed from the Pandas Series and DataFrame, respectively. Spectrum and Spectra are actually composite classes: pure Python classes that store both a Pandas object and metadata attributes; however, to the end user, operate identically to Pandas objects. In the future, scikit-spectra may be refactored to truly subclass from Pandas objects, as libraries like GeoPandas[8] and Xray[9] have recently shown how to do this properly.

In addition, scikit-spectra provides a **SpecStack** class for operating on multiple Spectra, analogous to the Pandas Panel class. SpecStack provides basic functionality for operations on multiple datasets, but does not try to emulate the API of the Panel.

Scikit-spectra defines custom Pandas Index objects, such as the **SpecIndex**, **TimeIndex** and **TempIndex** to support common spectral labels. For example, a TimeIndex supports timestamped labels and can convert to interval representations, (e.g. seconds elapsed), and the SpecIndex can transform spectral units, for example nanometers to inverse centimeters to electron volts. Arbitrary unit systems can also be defined through the **Unit** class. For example, a custom unit to denote polarization would be defined as follows:

```
from skspec.units import Unit

polarization = Unit(
       short = 'polar',
       full = 'Polarization',
       symbol = r'$\theta$', # Latex optional
       )
```

The *short*, *full* and *symbol* attributes ensure new units will automatically interface to the indexing and plotting systems.

Scikit-spectra includes graphical applications developed fully with IPython's widget API, and is one of the first libraries to do so. The documentation is built with Sphinx[10], using the Bootstrap theme[11], and sphinx gallery extensions[12]; and is heavily inspired by the scikit-image and scikit-learn[13] docs.

Examples of use[3]

To illustrate some basic functionality of scikit-spectra, data are analyzed from a system of gold nanoparticles (AuNPs) in a cuvette of water before and after protein has been added to the solution. Binding between the protein and the nanoparticles yields a characteristic shift towards long wavelengths in the absorbance spectrum of the gold, known as the localized surface plasmon resonance[14,15]. For brevity, a bundled dataset, `aunps_water()`, is used; however, reading data from a CSV file is quite easy, as scikit-spectra wraps Pandas' powerful `read_csv()` parser.

```
from skspec.data import aunps_water

ts = aunps_water()
ts.iloc[0:5, 0:5]
```

The *iloc* indexer was used to display only the first five rows and columns, as seen in **Table 1**. If working in the IPython Notebook, Spectrum and Spectra objects will automatically render as HTML tables, with metadata attributes such as name, units, shape, baseline; and reference and normalization states, shown in the header. The ability to store arbitrary metadata is crucial to repurposing Pandas for specific applications.

In this dataset, the baseline comes preset, but is not subtracted. The dataset (baseline and reference spectra are plotted as dashed lines for clarity) is shown in **Fig. 1**.

```
ts.reference = 0
ts.varunit = 's'
ax = ts.plot()
ts.baseline.T.plot(color='k', ls='--', ax=ax)
ts.reference.T.plot(color='magenta', ls = '--',
ax=ax);
```

Spectral data in their raw form are typically not very useful; it is better to work with the absorbance spectrum, which is defined by the transformation:

Gold Nanoparticles in Water (5 X 5) **[timestamp X nanometers]** **lunit: Counts (photons)**

Baseline: Found (no sub) **Reference: Found** **Normalization: None**

	2013-02-04 15:40:48	2013-02-04 15:41:34	2013-02-04 15:42:20	2013-02-04 15:43:06	2013-02-04 15:43:52
200	305.53	306.31	305.23	305.17	307.22
201	311.67	313.43	312.75	314.12	314.88
202	318.09	319.56	319.12	319.51	319.13
203	323.47	324.90	322.34	324.48	325.61
204	340.97	340.09	340.86	342.99	341.78

Table 1: HTML output of first five rows and columns of gold nanoparticles in water. This built-in dataset is preset with a stored baseline and reference spectra, and has column labels of timestamps.

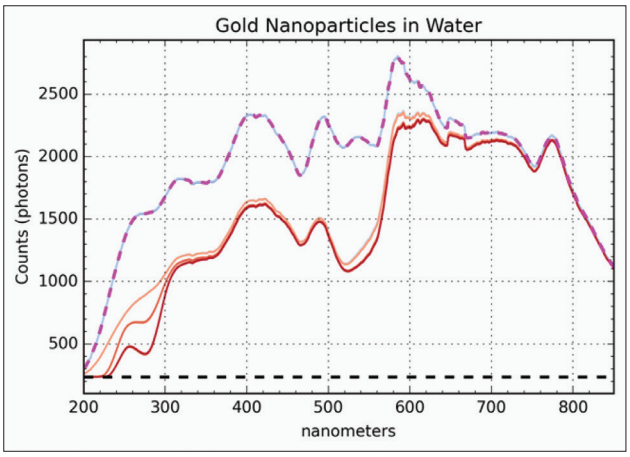

Fig. 1: Gold nanoparticles in water visualized: the data is stored in a TimeSpectra class. Baseline and reference are overlaid as dashed lines in black and magenta, respectively. Reference is user-selected, in this case set to the first curve in the dataset.

$$A_n(\lambda) = -\log_{10}\left(S_n(\lambda)\ B_n(\lambda) / R_n(\lambda)\ B_n(\lambda)\right)$$

where $A_n(\lambda)$ is the absorbance of the nth curve, $S_n(\lambda)$ is the spectrum, $R_n(\lambda)$ is the reference spectrum, and $B_n(\lambda)$ is the baseline. The absorbance data tend to be very noisy in the short wavelength region, due to the small signal in the raw data, so the usual procedure is to crop the values between 400–700nm. The `nearby()` method invokes nearest-neighbor slicing to index over an approximate range; for example, `nearby[400:700]` will automatically find and slice between the closest spectral index values, in this case 400.36–699.58nm.

```
from skspec.data import aunps_water

ts.sub_base() # subtract baseline inplace
ts.norm = 'a' # absorbance
ts = ts.nearby[400:700]
ts.plot(cbar=True)
```

The curves in **Fig. 2** show the surface plasmon resonance around 525nm and its clear shift to the right after proteins are added.

The plasmon resonance refers to the wavelength at which the nanoparticles maximally absorb, $A_n(\lambda_{max})$. Prior to analysis, the first few curves must be eliminated. These correspond to the timepoints taken prior to the addition of nanoparticles to the cuvette; that is, $A_n(\lambda_{max}) = 0$. This is most easily done through boolean masking, which nicely exemplifies the notion of NumPy-compatibility .

The blue curves in **Fig. 2** correspond to timeseries taken before the addition of nanoparticles. To remove these and retain only the subset of curves with significant absorbance, a mask is defined with a lower threshold of 0.10 absorbance units.

```
mask = ts.max() > 0.10
ts_cut = ts[ts.columns[mask]] #Index by the mask
```

The new TimeSpectra, `ts_cut`, retains only curves after the addition of AuNPs, the timepoint when the absorbance maximum rises above the threshold value.

Next, the plasmon resonance shift vs. time is analyzed. Pandas already has a method that returns the index corresponding to the maximum value for every curve in the dataset: `idxmax()`. Since scikit-spectra objects inherit all Pandas methods, `idxmax()` is also a TimeSpectra method.

```
ts_cut.idxmax().plot(title='SPR shift vs. time')
plt.ylabel('Peak Wavelength (nm)')
```

The method, `ts_cut.idxmax()`, returns a Spectrum class whose `plot()` method is then called. The result, shown in **Fig. 3**, indicates that the plasmon resonance hovers between 524–525nm, then shifts about 4 nm to 528–529nm. The jagged nature of the trend merely reflects the 1nm resolution of our benchtop spectrometer.

Most of the analysis so far could have been performed in one of scikit-spectra's Notebook GUIs. At any point in the workflow, one could have opened the GUI, manipulated the data, exported it back into the Notebook namespace, and resumed working through the API, exemplifying the philosophy of the *GUI when you want it, the API when you need it*. The code needed to run the GUI and a screenshot

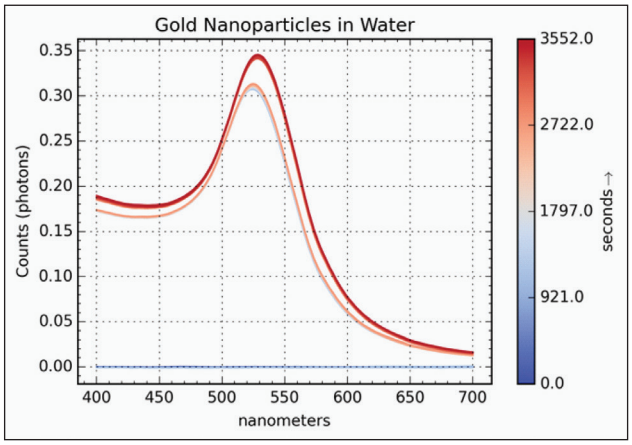

Fig. 2: Absorbance of AuNPs increases after the addition of bovine serum albumin protein, due to binding between the protein and gold surface. The amount of protein bound to the surface is related to the size of the shift in peak position.

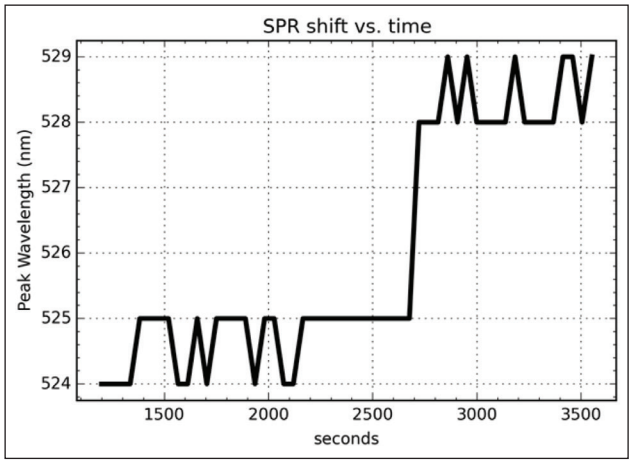

Fig. 3: Absorbance spectra maximum vs. time shows the movement of the plasmon resonance position after protein binds to the AuNPs.

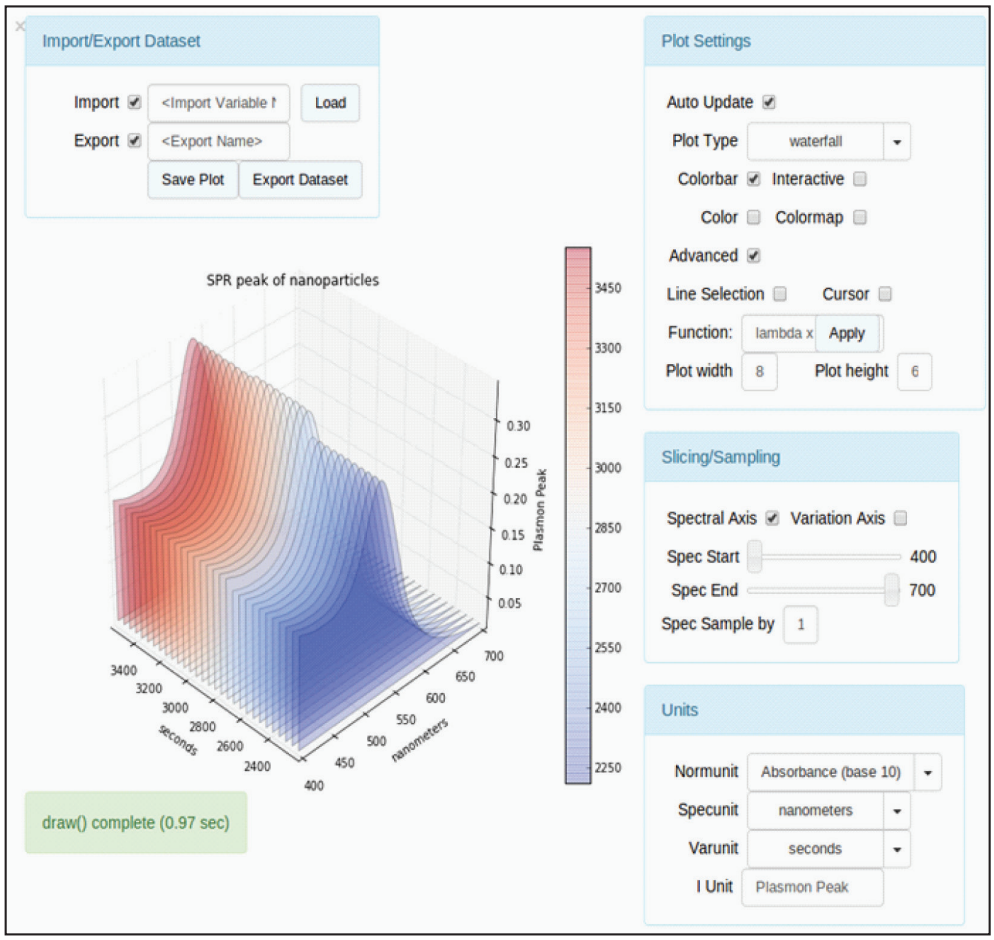

Fig. 4: IPython Notebook GUI for plotting, slicing, resampling, changing units and IO.

of the result are shown below in **Fig. 4**. The GUI supports nine plot types, as well as interactive plots through mpld3[16]. A video tutorial of the GUI is available on the scikit-spectra website[17].

```
from skspec.interact import SpectraModel,
SpectraGui

specmodel = SpectraModel(spec=aunps)  # Traitlets
                                        model
gui = SpectraGui(model=specmodel)     # GUI
gui.tight_layout()
gui
```

Scikit-spectra provides a natural framework for generalized spectroscopy applications, such as two-dimensional correlation spectroscopy(2DCS)[18], factor analysis[19] and chemometrics[20]. An API for 2DCS has been implemented, with the intention of turning scikit-spectra into the de-facto toolkit for correlation spectroscopy. While there are a few graphical applications for 2DCS, like 2dShiege[21] and MIDAS[22] in MATLAB™, there is no corresponding API-level toolkit available. To demonstrate scikit-spectra's 2DCS API, the asynchronous[4] correlation spectrum[23] of the AuNPs dataset will be analyzed.

Asynchronicity, $\Psi(\lambda,\lambda)$, measures the spectral distribution of uncorrelated events over a time[5] interval. For example, if a peak at λ=a forms early in an experiment, and then later a second peak at λ=b appears, then there is asynchronicity at Ψ(a,b) because these events occurred at different times: they are uncorrelated and likely due to different underlying processes in the system. If the peaks had formed at the same time, then they would be regarded as highly *synchronous*. Together, synchronicity and asynchronicity encompass all of the variance in the dataset. 2DCS applications are discussed much more extensively in the scikit-spectra documentation. For datasets with many spectral peaks, 2DCS can often resolve otherwise intractable information about the order and nature of events in the system.

```
from skspec.correlation import Corr2d

cspec = Corr2d(ts.nearby[:600])
cspec.async.plot(contours=128, cbar=True)
```

Fig. 5 illustrates the asynchronicity at wavelengths ranging from 200 to 600nm in the absorbance spectra. The time-span, spectral unit, spectral symbol and other metadata appear in the default plot labels, demonstrating the connectedness of scikit-spectra's units and plotting APIs. Strong cross-peaks between the ultraviolet and the plasmon resonance regions indicate that at some point in the experiment these peaks change asynchronously. This is verified by looking back at the data and observing that the initial protein binding leaves the nanoparticles saturated, with no binding-sites for a second addition of proteins. However, the additional protein does increase the absorbance in UV region. In other words, adding protein late in the experiment increases short-wavelength absorption, but has no effect on the plasmon resonance shift, resulting in asynchronicity between the 250nm and 530nm regions.

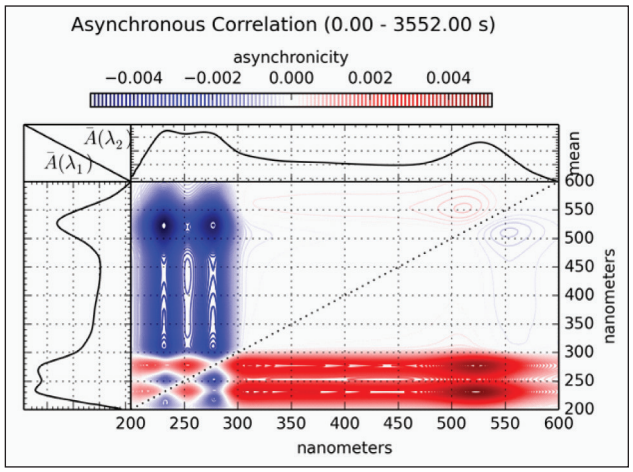

Fig. 5: Asynchronous correlation spectrum of gold nanoparticle absorbance at wavelengths ranging from 200–600nm. The strong cross peaks between the UV and plasmon resonance regions (roughly 525–550nm) corresponds to the time after nanoparticle-protein binding is saturated. Addition of more protein causes a UV response, but no response in the plasmon resonance region, resulting in this asynchronicity. Sideplots show the mean-centered average spectrum from the full set.

Quality control
Scikit-spectra includes a preliminary nose[24] test framework, inspired by the excellent Pandas test suite. A collection of tutorials and website examples are batch-run to catch breaking-changes that are not covered by the nose tests.

(2) Availability

Operating system
Scikit-spectra has been tested on Ubuntu 10.04, 12.04, 13.10; Mac OS 10.8, and Windows 7.

Programming language
Python 2.7

Dependencies

For core functionality
Pandas v0.14, SciPy, NumPy, matplotlib

For full functionality
IPython v2.0 or higher, mpld3, Traits

List of contributers
Adam Hughes, Zhaowen Liu

Software location
Archive

Name
scikit-spectra

Identifier
DOI 10.5281/zenodo.13965

Licence
BSD 3-Clause License

Publisher
Zenodo

Date published
1/15/15

Code repository
Name
Github

Identifier
https://github.com/hugadams/scikit-spectra

Licence
BSD 3-Clause License

Current version
0.3.2

Date published
Fall 2012 (formerly “pyuvvis”)

Documentation
Url
hugadams.github.io/scikit-spectra/

Mailing list
scikit-spectra.39571.n7.nabble.com

(3) Reuse potential

Scikit-spectra is built for generalized applications, and built on the already successfully Pandas library. Scikit-spectra’s core design is adaptable to many branches of spectroscopy, such as NMR, IR and Raman. Ideally, each branch will eventually be supported as a distinct subset of scikit-spectra, built on the same core framework. This is a long-term goal and will require contributions from many scientists and developers. The vision for scikit-spectra is to adapt and interface with other spectroscopy libraries, not supplant them. Unifying Python’s spectroscopy libraries, whether or not it ultimately involves scikit-spectra, is critical to bringing open-source solutions to the research community, much in the same way that ImageJ[25] has brought open-source image processing into the mainstream. Interested developers are welcome to contact the authors with suggestions or ideas.

Acknowledgements
We’d like to acknowledge the following developers for helpful discussions, without which scikit-spectra would not have been realized:

- Nicholas Bollweg (IPython)
- Jeff Reback and Stephan Hoyer (Pandas)
- Jonathan March and Robert Kern (Traits)

Notes

[1] Scikit stands for SciPy Toolkit, which are SciPy-based libraries deemed too specialized to live in the core SciPy distribution.

[2] The term "explorative" refers to an API compatible with SciPy libraries to streamline customized analysis visualization.

[3] These examples are available in a single notebook at: http://nbviewer.ipython.org/github/hugadams/scikit-spectra/blob/master/examples/Notebooks/grad_presentation.ipynb

[4] The synchronous and asynchronous correlation spectra are fundamental to 2DCS. Some important new developments include generalized scaling of correlation spectra[X] and the derivation of the so-called codistribution spectra[X], both of which are built into scikit-spectra's 2DCS API.

[5] 2DCS is also applicable to non-temporal datasets, for example spectra changing as a function of pressure, temperature or any other "perturbation variable".

References

1. **Bohren, H** 1983 Absorption and Scattering of Light by Small Particles, John Wiley & Sons, INC.
2. **Ginsburg, A** and **Mirocha, J** 2011 'PySpecKit: Python Spectroscopic Toolkit', Astrophysics Source Code Library, record ascl:1109.001 URL: http://adsabs.harvard.edu/abs/2011ascl.soft09001G
3. **Gorsuch, R L** 1983 Factor Analysis, 2nd edn, Lawrence Erlbaum Associats, Inc., Hillsdale, NJ.
4. **Helmus, J** and **Jaroniec, C** 2013 'Nmrglue: An open source Python package for the analysis of multidimensional NMR data', Journal of Biomolecular NMR 55(4), 355–367. URL: http://link.springer.com/article/10.1007/s10858-013-9718-x
5. **Hoyer, S** 2015 'xray: N-D labeled arrays and datasets in Python'. URL: https://github.com/xray/xray
6. **Hughes, A** and **Liu, Z** 2014 'scikit-spectra: Tools for explorative spectroscopy'. URL: http://hugadams.github.io/scikit-spectra/
7. **Jones, E, Oliphant, T** and **Peterson, P** 2001 'Scipy: Open source scientific tools for Python'. URL: http://www.scipy.org
8. **Jordahl, K** 2014 'GeoPandas: Python tools for geographic data'. URL: https://github.com/geopandas/geopandas
9. **McKinney, W** 2010 Data Structures for Statistical Computing in Python, in S. van der Walt & J. Millman, eds, 'Proceedings of the 9th Python in Science Conference', pp. 51–56. URL: http://pandas.pydata.org/
10. **Morita, S** 2002 '2D Shige'. URL: https://sites.google.com/site/shigemorita/home/2dshige
11. **Nájera, O** 2014 'Sphinx-Gallery'. URL: https://github.com/sphinx-gallery/sphinx-gallery
12. **Noda, I** 2007 'Scaling techniques to enhance two-dimensional correlation spectra', Journal of Molecular Structure 883–884, 216–227. URL: http://linkinghub.elsevier.com/retrieve/pii/S0022286007008411
13. **Noda, I** and **Ozaki, Y** 2004 Two-Dimensional Correlation Spectroscopy, Wiley.
14. **Normand, E** 2011 MIDAS 2010: Mid-Infrared Data Analysis Software 2010 A Matlab Package for 2D IR Spec-troscopy Analysis. URL: http://www.mathworks.com/matlabcentral/fileexchange/32384-midas-2010
15. **Oliphant, T E** 2007 'Python for Scientific Computing', Computing in Science & Engineering 9(90), 2007. URL: http://www.numpy.org/
16. **Pedregosa, F** et. al. 2011 'Scikit-learn: Machine Learning in Python', Journal of Machine Learning Research 12, 2825–2830. URL: http://scikit-learn.org/stable/
17. **Pellerin, J** 2009 'nose'. URL: https://nose.readthedocs.org/en/latest/
18. **Perez, F** and **Granger, B** 2007 'IPython: a Stystem for Interactive Scientific Computing', Computing in Science and Engineering 9(3), 21–29.
19. **Roemer, R** 2014 'Sphinx Bootstrap Theme'. URL: http://ryan-roemer.github.io/sphinx-bootstrap-theme/
20. **Schneider, C A, Rasband, W S,** and **Eliceiri, K** 2012 'NIH Image to ImageJ: 25 years of image analysis', Nature Methods: Focus on Bioimage Informatics 9(7), 671–675. URL: http://www.nature.com/nmeth/journal/v9/n7/full/nmeth.2089.html
21. **Sphinx: Python Documentation Generator** 2014 URL: http://sphinx-doc.org/index.html
22. **Vanderplas, J** 2014 'mpld3: Brining Matplotlib to the Browser'. URL: http://mpld3.github.io/
23. **van der Walt, S, Nunez-Iglesias, J, Boulogne, F, Warner, J** and **Al, E** 2014 'scikit-image: image processing in Python', PeerJ (453). URL: http://scikit-image.org/
24. **Willets, K A** and **Van Duyne, R P** 2007 'Localized surface plasmon resonance spectroscopy and sensing.', Annual review of physical chemistry 58, 267–97. URL: http://www.ncbi.nlm.nih.gov/pubmed/17067281
25. **Wu, Y, Tsenkova, R** and **Ozaki, Y** 2000 'Methods, Chemometrics, and Two - Dimensional Correlation Spectroscopy in the Analysis of Near - Infrared', 54(7).

How to cite this article: Hughes, A, Liu, Z and Reeves, M E 2015 Scikit-spectra: Explorative Spectroscopy in Python. *Journal of Open Research Software* 3:e6, DOI: http://dx.doi.org/10.5334/jors.bs

Published: 05 June 2015

]u[*Journal of Open Research Software* is a peer-reviewed open access journal published by Ubiquity Press

OPEN ACCESS