Magni: A Python Package for Compressive Sampling and Reconstruction of Atomic Force Microscopy Images

Oxvig, Christian Schou; Pedersen, Patrick Steffen; Arildsen, Thomas; Østergaard, Jan; Larsen, Torben

Published in:
Journal of Open Research Software

DOI (link to publication from Publisher):
10.5334/jors.bk

Publication date:
2014

Document Version
Accepted author manuscript, peer reviewed version

Journal of
open research software

Oxvig, C S, Pedersen , P S, Arildsen, T, Østergaard, J and Larsen, T 2014 Magni: A Python Package for Compressive Sampling and Reconstruction of Atomic Force Microscopy Images. *Journal of Open Research Software,* 2: e29, DOI: http://dx.doi.org/10.5334/jors.bk

SOFTWARE METAPAPER

Magni: A Python Package for Compressive Sampling and Reconstruction of Atomic Force Microscopy Images

Christian Schou Oxvig[1], Patrick Steffen Pedersen[1], Thomas Arildsen[1], Jan Østergaard[1] and Torben Larsen[1]

[1] Signal and Information Processing Section, Department of Electronic Systems, Faculty of Engineering of Science, Aalborg University, Aalborg, Denmark

Magni is an open source Python package that embraces compressed sensing and Atomic Force Microscopy (AFM) imaging techniques. It provides AFM-specific functionality for undersampling and reconstructing images from AFM equipment and thereby accelerating the acquisition of AFM images. Magni also provides researchers in compressed sensing with a selection of algorithms for reconstructing undersampled general images, and offers a consistent and rigorous way to efficiently evaluate the researchers own developed reconstruction algorithms in terms of phase transitions. The package also serves as a convenient platform for researchers in compressed sensing aiming at obtaining a high degree of reproducibility of their research.

Keywords: Atomic Force Microscopy; Compressive Sensing; Python; Image Reconstruction; Reproducible Research

Funding Statement: TThis project has been supported by 1) The Danish Council for Independent Research (DFF) via funding from DFF/Technology and Production (FTP), grant DFF–1335–00278, for the project "Enabling Fast Image Acquisition for Atomic Force Microscopy using Compressed Sensing", and 2) by the Danish e-Infrastructure Cooperation (DeIC) via a grant for a high performance computing system for the project "High Performance Computing SMP Server for Signal Processing".

(1) Overview

Introduction

In our research group at Aalborg University (AAU) we have recently launched a new research project, FastAFM[1], seeking to utilise compressed sensing in accelerating the acquisition of atomic force microscopy images. This is a relatively unexplored application area where results have only just started to appear [1], [2]. With the present paper, we present the general software package `magni`, which we have developed to combine compressed sensing and AFM imaging techniques.

Compressed sensing is a theory which has attracted a great deal of attention recently. In brief, the theory states that a wide range of possible signal types can be accurately represented from a greatly reduced number of acquired samples [3], [4]. That is, these signal types can be accurately reconstructed from samples taken significantly below the Shannon-Nyquist rate which is normally seen as the ultimate limit.

Atomic Force Microscopy (AFM) is one of the most advanced tools for high-resolution imaging and manipulation of nanoscale matter [5]. When used for imaging, it is able to generate a 3D surface map with sub nanometer resolution of an object [6]. To generate this map, a sharp probe is brought close to the surface of the object, and the probe tip and the object are then moved relative to each other. The mechanical probe tip is affected by the force on the surface and, loosely speaking, "feels" the surface [6], [7]. Unfortunately, standard AFM imaging requires a timescale on the order of minutes to hours to acquire an image [7].

In the course of our work with compressed sensing and AFM, we have identified three shortcomings. We find that these are not adequately met by available free and open source research software in this area:

1. Software for reconstruction of compressed sensing signals.
2. Software for consistent and rigorous testing of reconstruction algorithms, particularly of their reconstruction capabilities in terms of phase transition.
3. Software for acquisition and processing of AFM images in relation to compressed sensing.

While free and open source software for compressed sensing signal reconstruction is available as such [8], [9], most of this software relies on Matlab from MathWorks

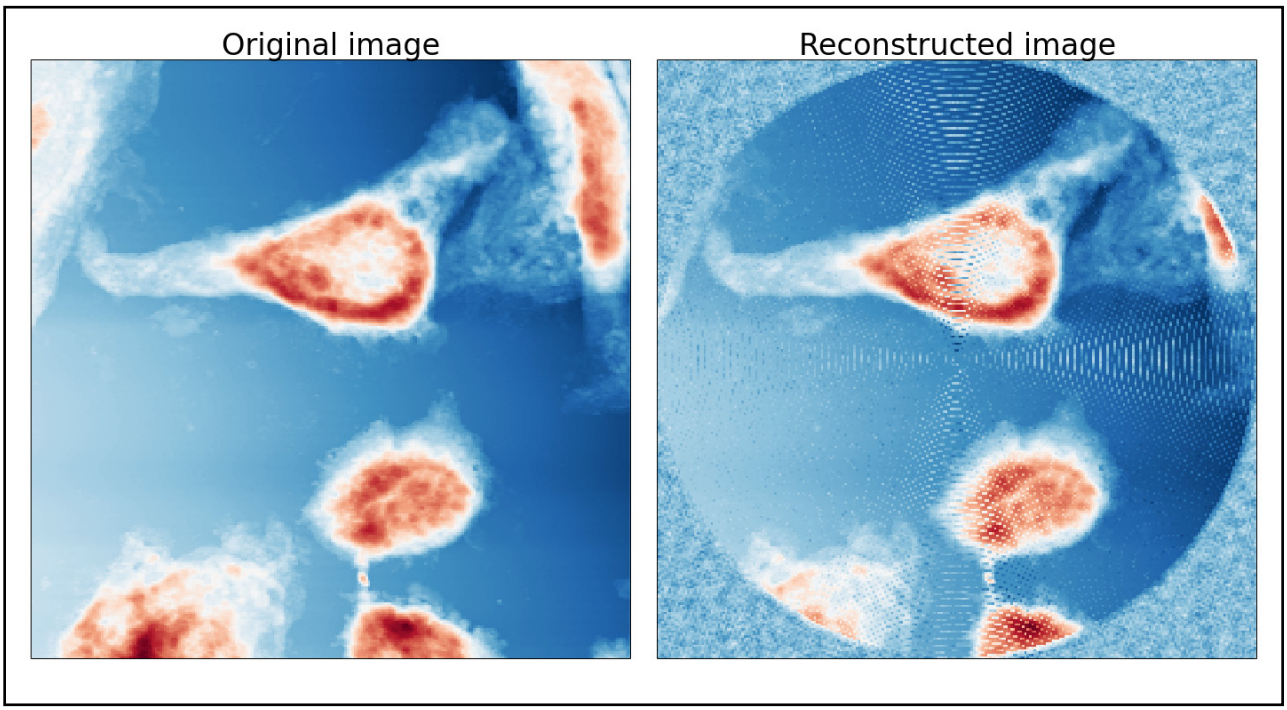

Figure 1: An example compressive sensing reconstruction of an AFM image.

which limits the reproducibility. Also, the available free and open source software for AFM image post-processing and visualisation [10] does not include compressed sensing related functionality. Instead, to mitigate these shortcomings, we have built the `magni` software package to ensure the highest degree of reproducibility defined for signal processing [11]. This has been done by relying on the free and open source programming language Python[2] and by making all examples, figures, etc. easily reproducible. An example of the reconstruction of an AFM image is shown in **Figure 1**. Using `magni`, the original image was loaded, preprocessed, sampled, reconstructed and displayed in less than 25 lines with intuitive calls to `magni` such as:

```
f>>> magni.imaging.measurements.spiral_sample_
image(h, w, scan_length, num_points)
>>> magni.imaging.measurements.construct_meas
urement_matrix(img_coords, h, w)
>>> magni.imaging.dictionaries.get_DCT((h, w))
>>> magni.afm.reconstruction.reconstruct(domain.
measurements, Phi, Psi)
```

We have designed the `magni` software package to address the above three needs: it contains a selection of compressed sensing reconstruction algorithms, a framework for evaluating reconstruction algorithms through Monte Carlo Simulations, and more AFM-specific functionality for sampling and reconstructing images from AFM equipment. Further development of the package is planned through our ongoing FastAFM research project as this progresses over the coming years. This further development aims to extend the functionality of the package both in terms of directly interfacing the AFM equipment and in terms of adding more post-processing and reconstruction algorithms.

Implementation and architecture

The `magni` package is written in the Python programming language[2]. Python combined with a set of third-party libraries is an excellent tool for scientific and engineering applications [12]. The `magni` package uses the following third-party libraries to exploit code reuse, to ease the quality control process, and to enhance the end user experience:

- The `numpy` and `scipy` libraries are used for handling data (using the efficient `ndarray` data container class [13]) and for performing numerical computations. These are two of the core libraries for scientific computing using Python [12].
- The `pytables` library [14] is used for storing data through a high-abstraction HDF5 database interface.
- The `matplotlib` library [15] is used for visualising data.
- The easy-to-use `IPython` [16] Notebook is used for presenting a number of examples showing the capabilities of the `magni` package.

The `magni` package is itself a library, i.e. it is a collection of Python sub-packages and modules and as such does not provide any (graphical) user interface. The functionality provided by `magni` may be grouped into five categories with a sub-package assigned to each category, as illustrated in **Figure 2**. Furthermore, each sub-package has a number of modules or nested sub-packages to group related functionality.

As for coding style, procedural programming is preferred over object-oriented programming, to avoid unnecessary overhead [17]. Also, the developers found procedural programming more transparent for implementing the

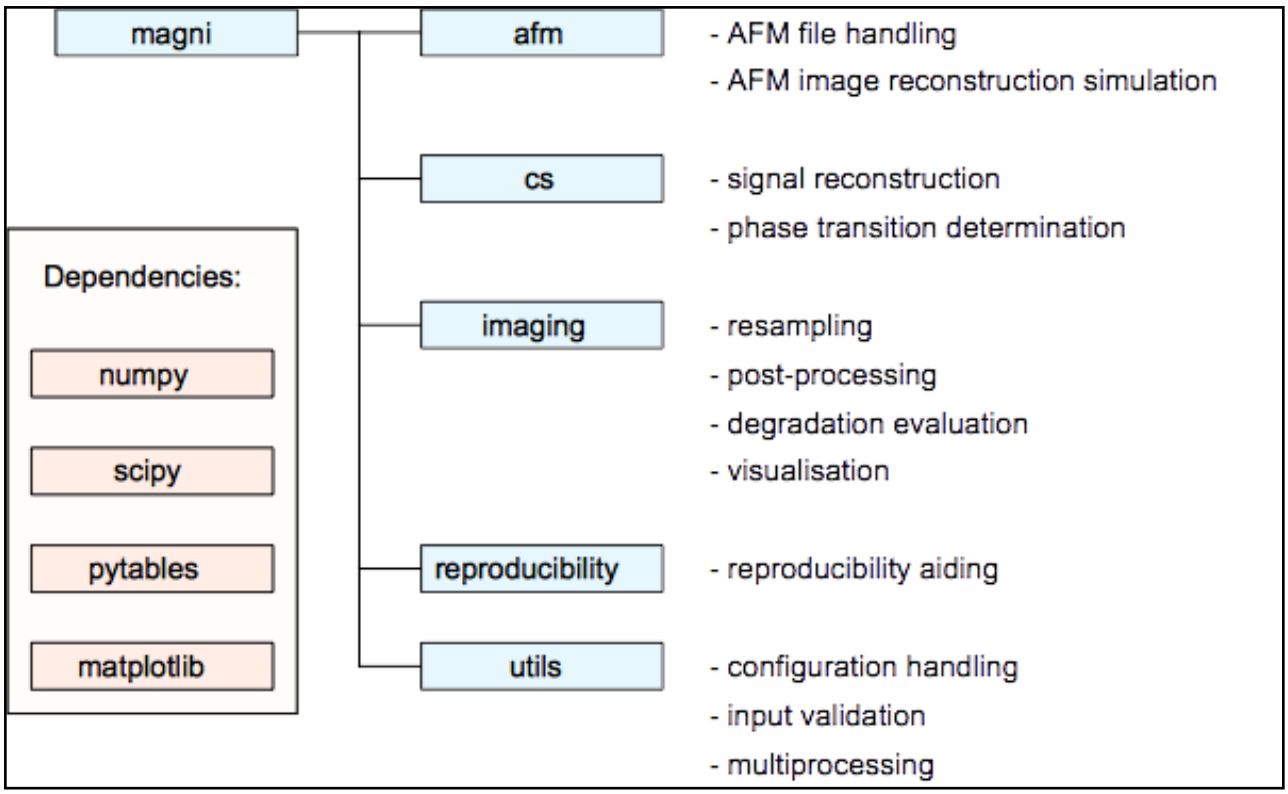

Figure 2: The functionality of the 5 sub-packages of magni along with the dependencies of `magni`.

desired functionality. Only in a few cases where the use of classes leads to significantly cleaner code, object-oriented programming is applied. Thus each module has its functionality encapsulated in a number of functions and classes, for which a distinction is made between public, internal, and private accessibility [18]. These accessibility levels are reflected in the code by use of the weak "internal use" indicator underscore convention as suggested by PEP8[3]:

- Private functionality is used only by the module itself. An underscore precedes the name of such functions or classes.
- Internal functionality is used by modules in the same or a nested sub-package. No underscore precedes the name of such functions or classes, but an underscore precedes the name of the module.
- Public functionality is available to the end-users and used by the package itself. No underscore precedes the name of such functions or classes, and no underscore precedes the name of the module.

Both functions and methods are implemented as to ensure readability in addition to efficiency by limiting the number of logical tasks per routine, the cyclomatic complexity [19], [20], and the number of physical code lines[4]. The cyclomatic complexity, i.e. the number of independent paths through the function, is kept below 10 for core functionality, consistent with observations on the level which programmers can usually handle flawlessly. This has been validated via the static code analyser `radon`[5]. The number of physical code lines is kept below 50 which is consistent with recommendations used at IBM and TRW [19] and general experiences in this field [21], [22].

The `magni` package complies with the PEP8 recommendation for Python coding conventions. This ensures that all Python code conforms to a number of recommendations with the aim of making the code user-friendly and thus easier and more robust from a maintenance point-of-view. The recommendations cover e.g. line width, variable naming conventions, package importing, indentations, and source encoding. Furthermore, `magni` is extensively documented using numpydoc[6] formatted doc-strings which describe the objective of the code, specify inputs and outputs of functions, elaborate on the functionality of the code, mention relevant references, and present examples of the use of the package. Finally, the input of every public (i.e. user-accessible) function and class is validated according to the known requirements with appropriate Python exceptions raised for invalid input. This is done to avoid runtime errors with hard-to-debug messages and stack traces.

Quality control

The code development procedure was built on what was found to be the best choice of methods from: 1) Well defined stage-based methods such as the structured waterfall approach [23] and the spiral approach [24] allowing backward interaction between different development phases; and 2) The test and adaptive centred Agile procedure [25] including e.g. Scrum [26], [27] and extreme programming [28], [29] with parts such as code reviews, code iteration, simplicity of design, frequent refactoring and collective ownership. All code modules were first

developed with tight links to the algorithm and refactoring was then performed to ensure maintainability, readability, robustness and sufficient performance. Multiple smaller and one large code review were held by 2-6 researchers including the main developers. Throughout the development process, Git was used for version control and issue tracking [30], and multiple branches were used to ensure that only tested code entered the master branch.

Testing and code validation has been handled by different instruments:

- 15 carefully designed end-to-end examples have been implemented in `IPython` Notebook (the .ipynb files). United, these examples exercise all critical code segments and serve the purpose of integration and regression testing.
- Doc-strings for all public functions include examples that are used in automated doctests[7]. This helps with the regression testing and ensures that the docstrings are kept up-to-date.
- `pyflakes` and `pylint` static source code analysers for Python have been used in the code development process to catch bugs and bad coding quality.

As always, no software package is better than its documentation and examples provided along with the package. The examples and part of the documentation have already been mentioned. Some of the examples use an AFM image, which is provided with the package. Furthermore, a full documentation in html is automatically generated from the doc-strings. A pdf version of this is shipped with the code.

(2) Availability

Operating system

Tested on Ubuntu 12.04 LTS Linux, Apple Mac OS X 10.9, and Microsoft Windows 7. Since `magni` is written in pure Python, it should run on any system on which Python and the `magni` dependencies run.

Programming language

The `magni` package is written in pure Python. Python 2 (>=2.7) or Python 3 (>=3.3) is required to use the package. The package has been tested with the Anaconda[8] Python distribution by Continuum Analytics.

Additional system requirements

`magni` is designed to process data sets of all sizes. Hardware requirements in terms of processor power, memory capacity, etc. depend primarily on the size of the data sets that are processed.

Dependencies

`magni` depends on numpy, `scipy`, `pytables`, and `matplotlib`. The package has been tested with:

- `numpy` version 1.8
- `scipy` version 0.13
- `pytables` version 3.1
- `matplotlib` version 1.3

The following libraries are optional requirements for `magni`:

- `IPython` Notebook >= 1.1 (for running examples)
- Math Kernel Library (`mkl`) >= 11.1 (for accelerated vector operations)
- `sphinx` >= 1.2 (for building the documentation from source)
- `napoleon` >= 0.2.6 (for building the documentation from source)

List of contributors

- Christian Schou Oxvig (Aalborg University) - Development
- Patrick Steffen Pedersen (Aalborg University) - Development
- Jan Østergaard (Aalborg University) - Testing and code review
- Thomas Arildsen (Aalborg University) - Testing and code review
- Tobias L. Jensen (Aalborg University) - Testing and code review
- Torben Larsen (Aalborg University) - Testing and code review

Archive

Name

Videnbasen (VBN), Aalborg University

Persistent identifier

DOI: http://doi.org/10.5278/VBN/MISC/Magni

License

BSD 2-Clause

Publisher

Christian Schou Oxvig

Date published

23/05/14

Code Repository

Name

GitHub

Identifier

https://github.com/SIP-AAU/Magni/

License

BSD 2-Clause

Date published

23/05/14

Language

English

(3) Reuse potential

The `magni` package has been designed to facilitate reuse through extensive documentation of functionality and interfaces. The code has been implemented with focus on readability. And the package is accompanied by a number of examples to demonstrate its use in various use-cases.

We expect the `magni` package to have significant reuse potential for researchers in the area of AFM, particularly in relation to compressed sensing acquisition and reconstruction of AFM images. This applies to both users interested in developing and testing new sampling patterns for use in conjunction with compressed sensing techniques and users developing new algorithms for compressed sensing in the context of AFM.

Furthermore, the `magni` package is applicable to compressed sensing in general and can be particularly useful to those looking for compressed sensing reconstruction algorithms for use in Python, which have so far been scarce. In addition to reconstruction algorithms, the package provides a consistent framework which can be used to empirically estimate the reconstruction capabilities of the users' own reconstruction algorithms in terms of reconstruction phase transitions.

Due to the `magni` package being based on well established Python libraries, it fits naturally into the Python ecosystem [31] of high-quality tools for scientific computing. The software complies with the reproducible research paradigm as used in the field of signal processing [11]. The intent of reproducible research is to create an open and transparent approach to the software related to some specific conducted research – see e.g. [32], [33], [34], [35]. We thus provide full open access to all source code and full reuse rights via the generous BSD 2-Clause license, making it easy for others to use the code base. While it is the plan of the developers to continuously expand the functionality of the software, others are free to use it in separate branches. The `reproducibility` subpackage goes one step further by providing functionality for reading and writing the version and complete configuration of `magni`. Furthermore, information about conda, git revision, and the system platform is included if available. This information can be automatically shipped alongside the results, by letting `magni` use the same HDF5 database for storing the two. With these features, the developers hope to inspire others to make their results reproducible.

Notes

[1] See http://dx.doi.org/10.5278/vbn/projects/FastAFM/.
[2] See https://www.python.org/.
[3] Python Enhancement Proposal, see http://legacy.python.org/dev/peps/pep-0008/.
[4] See https://docs.python.org/2/reference/lexical_analysis.html.
[5] See https://github.com/rubik/radon.
[6] See https://github.com/numpy/numpy/blob/master/doc/HOWTO_DOCUMENT.rst.txt/.
[7] See https://docs.python.org/2/library/doctest.html/.
[8] See http://www.continuum.io/anacondace.html/.

References

1. **Song, B, Xi, N, Yang, R, Lai, K W C** and **Qu, C** 2011 Video Rate Atomic Force Microscopy (AFM) Imaging using Compressive Sensing. *11th IEEE International Conference on Nanotechnology*. 15-18 August 2011, Portland, Oregon, USA: 1056–1059, DOI: http://dx.doi.org/10.1109/NANO.2011.6144587
2. **Andersson, S B** and **Pao, L Y** 2012 Non-Raster Sampling in Atomic Force Microscopy: A Compressed Sensing Approach. *American Control Conference (ACC)*. 27-29 June 2012, Montréal, Canada: 2485–2490.
3. **Baraniuk, R G** 2007 Compressive sensing [lecture notes]. *IEEE Signal Processing Magazine* 24(4): 118–121, DOI: http://dx.doi.org/10.1109/MSP.2007.4286571
4. **Candès, E J** and **Wakin, M B** 2007 An Introduction To Compressive Sampling. *IEEE Signal Processing Magazine* 25(2): 21–30, DOI: http://dx.doi.org/10.1109/MSP.2007.914731
5. **Abramovitch, D Y, Andersson, S B, Pao, L Y** and **Schitter, G** 2007 A Tutorial on the Mechanisms, Dynamics, and Control of Atomic Force Microscopes. *American Control Conference*. 11-13 July 2007, New York City, USA: 3488–3502, DOI: http://dx.doi.org/10.1109/ACC.2007.4282300
6. **Bhushan, B** and **Marti, O** 2010 Scanning Probe Microscopy – Principle of Operation, Instrumentation, and Probes In: *Springer Handbook of Nanotechnology*. Berlin Heidelberg: Springer, DOI: http://dx.doi.org/10.1007/978-3-642-02525-9_21
7. **Hansma, P K, Schitter, G, Fantner, G E** and **Prater, C** 2006 High-Speed Atomic Force Microscopy. *Science* 314(5799): 601–602, DOI: http://dx.doi.org/10.1126/science.1133497
8. **van den Berg, E** and **Friedlander, M P** 2008 Probing the Pareto Frontier for Basis Pursuit Solutions. *SIAM Journal on Scientific Computing* 31(2): 890–912, DOI: http://dx.doi.org/10.1137/080714488
9. **Yang, J** and **Zhang, Y** 2008 Alternating Direction Algorithms for l1-Problems in Compressive Sensing. *SIAM Journal on Scientific Computing* 33(1): 250–278, DOI: http://dx.doi.org/10.1137/090777761
10. **Klapetek, P** 2013 *Quantitative Data Processing in Scanning Probe Microscopy: SPM Applications for Nanometrology*. 1st ed. Elsevier.
11. **Vandewalle, P, Kovačević,** J and **Vetterli, M** 2009 Reproducible Research in Signal Processing [What, why, and how]. *IEEE Signal Processing Magazine* 26(3): 37–47, DOI: http://dx.doi.org/10.1109/MSP.2009.932122
12. **Oliphant, T E** 2007 Python for Scientific Computing. *Computing in Science & Engineering* 9(3): 10–20, DOI: http://dx.doi.org/10.1109/MCSE.2007.58
13. **van der Walt, S, Colbert, S C** and **Varoquaux, G** 2011 The NumPy Array: A Structure for Efficient Numerical Computation. *Computing in Science & Engineering* 13(2): 22–30, DOI: http://dx.doi.org/10.1109/MCSE.2011.37
14. **Alted, F** and **Fernández-Alonso, M** 2003 PyTables : Processing And Analyzing Extremely Large Amounts Of Data In Python. *PyCon2003*. April 2003, Washington, D.C., USA: 1–9.

15. **Hunter, J D** 2007 Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering* 9(3): 90–95, DOI: http://dx.doi.org/10.1109/MCSE.2007.55
16. **Pérez, F** and **Granger, B E** 2007 IPython: A System for Interactive Scientific Computing. *Computing in Science & Engineering* 9(3): 21–29, DOI: http://dx.doi.org/10.1109/MCSE.2007.53
17. **Jürgens, D** 2009 *Survey on Software Engineering for Scientific Applications: Reuseable Software, Grid Computing and Applications*. Germany: Institute of Scientific Computing, Carl-Friedrich-Gauss-Fakultät, Technische Universität Braunschweig.
18. **Moock, C** 2007 Essential ActionScript 3.0. 1st ed. O'Reilly Media / Adobe Dev Library.
19. **McCabe, T J** 1976 A Complexity Measure. *IEEE Transactions on Software Engineering* SE-2(4): 308–320, DOI: http://dx.doi.org/10.1109/TSE.1976.233837
20. **Watson, A H** and **McCabe, T J** 1996 Structured Testing: *A Testing Methodology Using the Cyclomatic Complexity Metric*. National Institute of Standards and Technology (NIST): 500-235.
21. **Shen, V Y, Yu, T J, Thebaut, S M** and **Paulsen, L R** 1985 Identifying Error-Prone Software -- An Empirical Study. *IEEE Transactions on Software Engineering* 11(4): 317–324, DOI: http://dx.doi.org/10.1109/TSE.1985.232222
22. **Kelly, D, Hook, D** and **Sanders, R** 2009 Five Recommended Practices for Computational Scientists Who Write Software. *Computing in Science & Engineering* 11(5): 48–53, DOI: http://dx.doi.org/10.1109/MCSE.2009.139
23. **Royce, W W** 1970 Managing the Development of Large Software Systems. *IEEE WESCON*. August 1970, : 328–338.
24. **Boehm, B W** 1988 A Spiral Model of Software Development and Enhancement. *Computer* 21(5): 61–72, DOI: http://dx.doi.org/10.1109/2.59
25. **Sletholt, M T, Hannay, J E, Pfahl, D** and **Langtangen, H P** 2012 What Do We Know about Scientific Software Development's Agile Practices?. *Computing in Science & Engineering* 14(2): 24–37, DOI: http://dx.doi.org/10.1109/MCSE.2011.113
26. **Mahnic, V** 2012 A Capstone Course on Agile Software Development Using Scrum. *IEEE Transactions on Education* 55(1): 99–106, DOI: http://dx.doi.org/10.1109/TE.2011.2142311
27. **Rising, L** and **Janoff, N S** 2000 The Scrum Software Development Process for Small Teams. *IEEE Software* 17(4): 26–32, DOI: http://dx.doi.org/10.1109/52.854065
28. **Beck, K** 1999 Embracing Change with Extreme Programming. *Computer* 32(10): 70–77, DOI: http://dx.doi.org/10.1109/2.796139
29. **Maurer, F** and **Martel, S** 2002 Extreme Programming: Rapid Development for Web-Based Applications. *IEEE Internet Computing* 6(1): 86–90, DOI: http://dx.doi.org/10.1109/4236.989006
30. **Loeliger, J** and **McCullough, M** 2012 *Version Control with Git: Powerful tools and techniques for collaborative software development*. 2nd ed. O'Reilly Media.
31. **Pérez, F, Granger, B E** and **Hunter, J D** 2011 Python: An Ecosystem for Scientific Computing. *Computing in Science & Engineering* 13(2): 13–21, DOI: http://dx.doi.org/10.1109/MCSE.2010.119
32. **Schwab, M, Karrenbach, M** and **Claerbout, J** 2011 Making Scientific Computations Reproducible. *Computing in Science & Engineering* 2(6): 61–67, DOI: http://dx.doi.org/10.1109/5992.881708
33. **LeVeque, R J, Mitchell, I M** and **Stodden, V** 2012 Reproducible Research for Scientific Computing: Tools and Strategies for Changing the Culture. *Computing in Science & Engineering* 14(4): 13–17, DOI: http://dx.doi.org/10.1109/MCSE.2012.38
34. **Barni, M** and **Perez-Gonzalez, F** 2005 Pushing Science into Signal Processing. *IEEE Signal Processing Magazine* 22(4): 120–119, DOI: http://dx.doi.org/10.1109/MSP.2005.1458324
35. **Fomel, S** and **Claerbout, J F** 2009 Guest Editors' Introduction: Reproducible Research. *Computing in Science & Engineering* 11(1): 5–7, DOI: http://dx.doi.org/10.1109/MCSE.2009.14

How to cite this article: Oxvig, C S, Pedersen , P S, Arildsen, T, Østergaard, J and Larsen, T 2014 Magni: A Python Package for Compressive Sampling and Reconstruction of Atomic Force Microscopy Images. *Journal of Open Research Software,* 2: e29, DOI: http://dx.doi.org/10.5334/jors.bk

Published: 07 October 2014

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

OPEN ACCESS