Test Report: SpectrumImageAnalysisPy

Isobel Bicket

December 18, 2017

1 Revision History

Date	Version	Notes
December 18, 2017	1.0	Initial draft

2 Symbols, Abbreviations and Acronyms

See SRS document for further definitions.

symbol	**description**
RMS	Root Mean Square
SNR	Signal-to-Noise Ratio
T	Test

Contents

List of Tables

List of Figures

This document details the results of the testing detailed in the Test Plan, for the code SpectrumImageAnalysisPy, which can be found, along with the other documentation, on https://github.com/icbicket/SpectrumImageAnalysisPy/tree/SpectrumImageAnalysisPy_dev. Data and results from testing can be found under /home/isobel/Documents/McMaster/PythonCodes/DataAnalysis/src/TestCase.

3 Functional Requirements Evaluation

In this section, the testing for each of the functional requirements is addressed. Some testing is still in progress and will be completed at a later date.

3.1 R1: SI Inputs

Some unit testing has been performed for error checking on initializing a spectrum image. More remains to be done.

3.2 R2: Spectrum Inputs

Testing remains to be done.

3.3 R3: Input Verification

Some unit testing has been performed for error checking on initializing a spectrum image. More remains to be done.

3.4 R4: Slice (1D) SI and extract Image

Some unit testing has been performed on the 1D slice module in Spectrum.

3.5 R5: Mask (2D) SI and extract Spectrum

Testing remains to be done.

3.6 R6: Richardson-Lucy Deconvolution

The test data for testing the RMS error and SNR of the deconvolution algorithm were obtained from Dr. E.P.Bellido and are the same datasets used in [1]. The inputs to the deconvolution function are shown in Figure 1. The original spectrum is simulated, with a narrow Gaussian representing the zero loss peak. An experimental point spread function was convolved with it to represent the blurring of the 'ideal' spectrum with the point spread function of the CCD. Poisson noise was added to the result of this convolution to imitate noise from the detector.

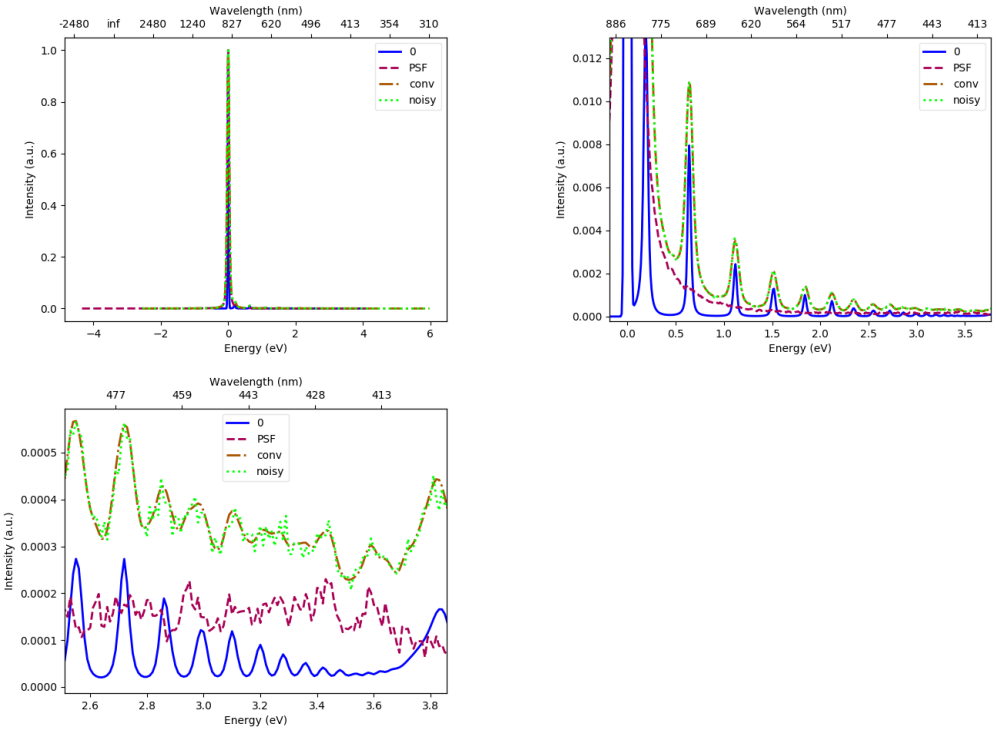

Figure 1: Original spectrum (0), Point spread function (PSF), the original spectrum convolved with the PSF (conv), and the original spectrum convolved with the PSF with Poisson noise added (noisy). Three images show three different regions of the spectrum.

The goal of deconvolution is to retrieve the original spectrum from the spectrum convolved with the PSF. Up to 2000 iterations of the Richardson-Lucy deconvolution algorithm have been performed on this dataset, using

the experimental point spread function, and the noisy convolved spectrum as inputs. The results at several chosen values are shown in Figure 2. As the number of deconvolutions increases, the spectra quickly approach the 'ideal' spectrum (RL1, RL50). Further increments represent small benefits in the approach to the 'ideal' spectrum (RL107-RL350). Using even more iterations results in periodic artifacts starting to appear in the spectrum (RL1000, RL2000), where small peaks appearing in between the main peaks can be seen and grow larger with further iterations.

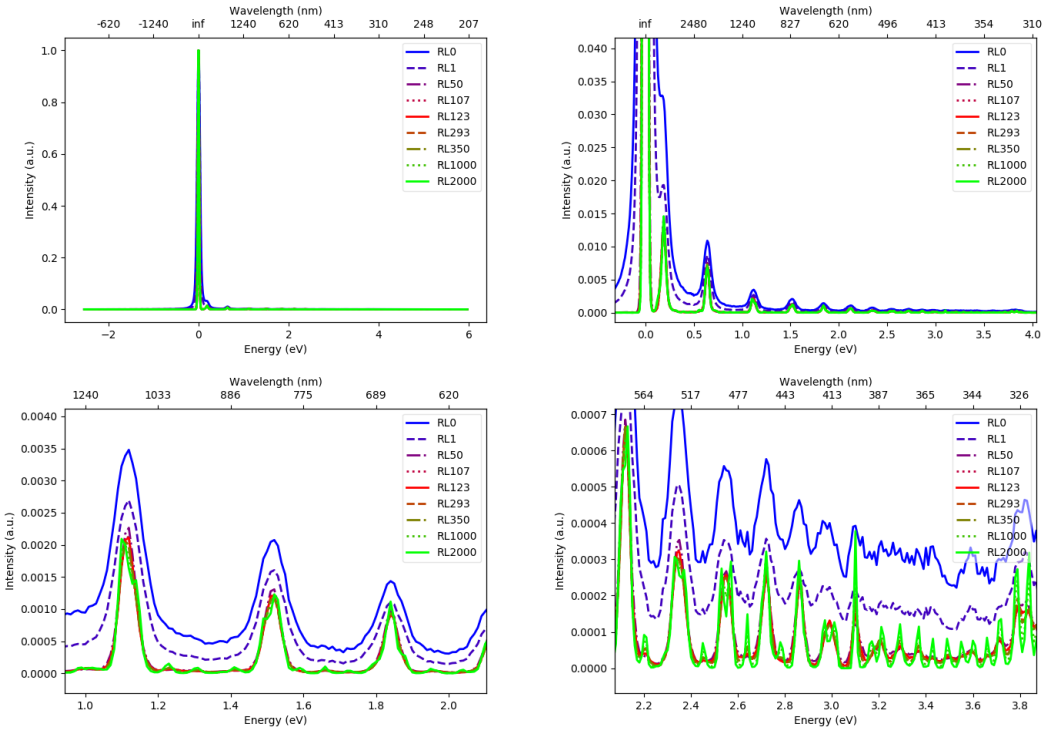

Figure 2: Spectra deconvolved using the Richardson-Lucy algorithm. Different numbers of iterations were used, as shown in the legend (RL2000 = 2000 iterations). Four images show different regions of the same plot.

To quantify the difference, the RMS error is plotted against the number of iterations in 3. There is at first a rapid decrease in the RMS error, ending in a local minimum at around 100-120 iterations (the exact value of the minimum depends on the noise added). Following this, the error increases as the periodic artifacts begin to appear and grow larger. The minimum RMS

error at around 100-120 iterations represents about 0.12% error.

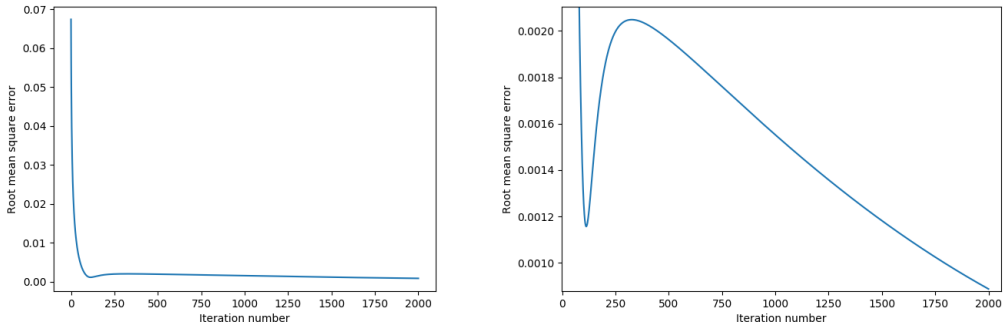

Figure 3: Root mean square error against number of RL deconvolution iterations performed.

To compare more directly with the data from Bellido *et al.* [1], the signal-to-noise ratio is plotted against the number of iterations in Figure 4. As was done by Bellido *et al.*, the calculation for the SNR was split into two sections in the spectrum. The SNR-ZLP section represents the SNR calculated within 0.1 eV of the zero loss peak (at 0 eV), and the SNR-plasmon section represents the spectrum from 0.1 eV to 4.0 eV (the 'plasmon region' of the spectrum). The SNR increases rapidly with the first few iterations, followed by a peak at a similar value to the minimum in the RMS error. Interestingly, the peak in SNR does not coincide exactly with the RMS minimum, but is slightly offset in the number of iterations. There is also a small difference between the SNR-ZLP and the SNR-plasmon peaks. After the peak, the SNR-ZLP begins to increase. The ZLP is easily the strongest feature in the spectrum: adding noise has very little effect on it and the SNR increases with the number of iterations (after a local minimum). The SNR-plasmon, on the other hand, has very little signal and the Poisson noise can easily dominate or replace some of the smaller peaks. After the local maximum, the SNR-plasmon begins to decrease and continues to decrease as the number of iterations increases. Inspecting the results of the deconvolution in Figure 2, it is apparent that this is caused by the periodic artifacts appearing between the peaks and the way that some of the peaks have split into two peaks (*e.g.*, near 2.6 eV).

These results are quite similar to those of Bellido *et al.*, though not exactly the same (likely due to the different Poisson noise used, since the Poisson noise was generated independently for this test). Although it appears that

there is a further decrease in error and an increase in the ZLP-SNR with large numbers of iterations, the results on the SNR-plasmon suggest that there is an optimum number of iterations (dependent on the amount of noise) to balance the RMS error and SNR for the region of interest, the plasmon region of the spectrum; care must be taken with the number of iterations used, as higher numbers can introduce extra spectrum peaks or emphasize the noise present. It should be noted that this optimum number of iterations will be difficult to determine in the experimental case, where the 'ideal' spectrum is not known and the noise may be higher.

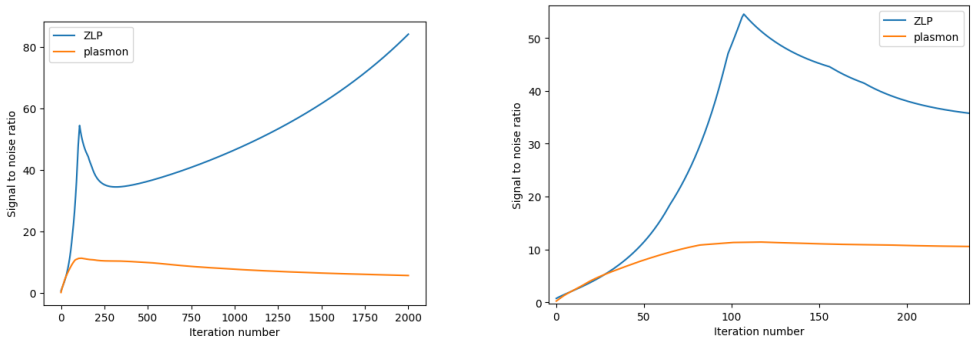

Figure 4: Signal to noise ratio against number of iterations

3.7 R7: Normalization

Testing has not yet been performed for this functional requirement.

3.8 R8: Background subtraction

Testing has not yet been performed for this functional requirement.

3.9 R9: Gain Correction

Testing has not yet been performed for this functional requirement.

4 Nonfunctional Requirements Evaluation

4.1 Usability

Testing has not yet been performed for this functional requirement. Tests for the functional requirements and unit tests will be completed before asking users to complete a survey on their experience.

4.2 Performance

Performance testing will be performed once unit tests and functional requirement tests have been completed.

5 Comparison to Existing Implementation

A comparison to the implementation in the HyperSpy python toolbox deconvolution function will be performed in the future.

6 Unit Testing

To date, partial unit testing has been performed on the following files:

- Filenamer_test.py, used to name the files to be exported and avoid conflicts with already existing files).
- Spectrum_test.py, used to implement the Data 1D Spectrum module.
- SpectrumImage_test.py, used to implement the Data 3D Spectrum module
- ImagePlotter_test.py, implementing the Display 2D Image module.

The inputs, outputs, and procedure for each unit test can be found in each unit test file. Coverage statistics for these files can be found in Statement Coverage and Branch Coverage. As of the latest update, all unit tests were passing:

```
Ran 50 tests in 0.008s

OK
```

7 Changes Due to Testing

- A zero in the denominator during RL deconvolution now raises an exception
- Minor bug fixes
- Rewrite of the file naming program for exporting files

8 Automated Testing

A Git Hook is used to automatically run the unit tests with every commit, providing automatic continuous integration testing.

9 Trace to Requirements

Table 1 shows the connections between the tests and testing files mentioned above and the functional requirements.

	R1	R2	R3	R4	R5	R6	R7	R8	R9
Section 3.6						X			
Filenamer_test.py				X	X				
Spectrum_test.py				X		X	X		
SpectrumImage_test.py	X								
ImagePlotter_test.py				X					

Table 1: Traceability Matrix Showing the Connections Between Functional Requirements and Tests

10 Trace to Modules

The relationship between the tests listed here and the different modules are listed in Tables 2, 3, and 4.

	M1	M2	M3	M4	M5	M6	M7	M8	M9
Section 3.6									
Filenamer_test.py						X	X	X	X
Spectrum_test.py									
SpectrumImage_test.py									
ImagePlotter_test.py									

Table 2: Traceability Matrix Showing the Connections Between Modules and Tests (1)

	M10	M11	M12	M13	M14	M15	M16	M17
Section 3.6	X							
Filenamer_test.py								
Spectrum_test.py	X	X			X			
SpectrumImage_test.py								
ImagePlotter_test.py								

Table 3: Traceability Matrix Showing the Connections Between Modules and Tests (2)

	M18	M19	M20	M21	M22	M23	M24	M25
Section 3.6								
Filenamer_test.py								
Spectrum_test.py								
SpectrumImage_test.py					X			
ImagePlotter_test.py	X							

Table 4: Traceability Matrix Showing the Connections Between Modules and Tests (3)

11 Code Coverage Metrics

Code coverage metrics were obtained using the Python coverage library (coverage.py-4.4.2) and represent the statement and branch coverage of the unit tests, to date.

11.1 Statement Coverage

Statement coverage provides a value for the percentage of lines which were run during the unit tests. The statistics can be found in Table 5.

Name	Statements	Missed	Coverage %
FileNamer.py	40	3	92%
ImagePlotter.py	136	97	29%
SpectrumImage.py	247	155	37%
Spectrum.py	136	70	49%

Table 5: Statement coverage

11.2 Branch Coverage

Branch coverage combines the number of statements run during the test with the number of branches covered during the test. The statistics for the branch coverage of the unit tests can be found in Table 6.

Branch Name	Statements	Missed	Branches	Partial Branches	Coverage %
FileNamer.py	40	3	14	1	93%
ImagePlotter.py	136	97	56	0	23%
SpectrumImage.py	247	155	60	1	37%
Spectrum.py	136	70	32	1	48%

Table 6: Branch coverage

References

[1] E. P. Bellido, D. Rossouw, and G. A. Botton, "Toward 10 meV Electron Energy-Loss Spectroscopy Resolution for Plasmonics," *Microscopy and Microanalysis*, vol. 20, pp. 767–778, June 2014.