SpectrumImageAnalysisPy

Isobel Bicket

December 7, 2017

1 Revision History

Date	Version	Notes
October 31, 2017	1.0	Initial draft

2 Symbols, Abbreviations and Acronyms

symbol	description
GUI	Graphical User Interface
RMS	Root Mean Square
T	Test

See the SRS Reference Material for further definitions.

Contents

List of Tables

This document outlines the different tests and assessment tools which will be used for assessing the performance of SpectrumImageAnalysisPy. The testing will be done to build confidence in the performance of the program and allow the end user to use it with more assurity that it is behaving correctly and any artifacts seen in the data are either a known result of the program, or independent of the use of SpectrumImageAnalysisPy to process the data.

3 General Information

The testing outlined in this document gives valuable contributions to the level of trust in the performance of the software. Not only will testing the software build confidence that it is performing correctly, but the results of the tests will also tell the user what to expect when running the software, for instance in terms of performance milestones. Testing not only ensures that the software will, within predictive power, fulfill its functional requirements, but gives the progress and effectiveness of the software in fulfulling the non-functional requirements. [Neat idea to have hyperrefs. Unfortunately they didn't work when I clicked them. —SS]

3.1 Purpose

The purpose of this document is to describe the tests which will be carried out on SpectrumImageAnalysisPy to verify that it meets the requirements described in the SRS.

[An explicit web-link to your GitHub repo would be nice. —SS]

3.2 Scope

This document will cover the verification plan for testing the software SpectrumImageAnalysisPy, including the system tests towards verifying that the software meets the functional and non-functional requirements; and the unit tests, of which a non-exhaustive list is given. The document does not cover the validation of the software algorithms, but will include an estimate of the combined error of the software and deconvolution algorithm used.

3.3 Overview of Document

It begins by outlining the structure of the document, followed by an overview of the testing arrangements. Specific details follow on the tests which will be performed to test whether or not SpectrumImageAnalysisPy satisfies the functional and non-functional requirements, as well as details of the unit tests which will be written. The unit tests themselves may satisfy some of the functional requirements without needing system testing, these will be detailed under Unit Testing Plan as appropriate.

4 Plan

This section details the plan to be followed when testing the software, including those involved in the testing, the testing approach, and the verification tools which will be used.

4.1 Software Description

The software, SpectrumImageAnalysisPy, is designed as a data analysis tool for electron energy loss spectroscopy and cathodoluminescence spectrum imaging data. It provides several data processing routines to the user and a graphical user interface to navigate the 3D data set and export desired spectra or image slices. The requirements and expectations for SpectrumImageAnalysisPy are detailed in the Software Requirements Specification.

4.2 Test Team

The test team has one member: Isobel Bicket.

4.3 Automated Testing Approach

Unit testing will be performed on the functions within the code, as an automated task to be done with every update of the code. The running of automated tests also provides regression testing and integration testing of new features and updates. The goal of testing is 100% code coverage.
The testing plan will not cover testing of the GUI, only the functions called by the GUI after user interaction. Proper user interfacing is expected to be

done by the library used to build the GUI.

4.4 Verification Tools

The following verification tools will be used:

- Python unittest library: for performing unit tests on the code;
- Python coverage library: to determine the coverage of the tests;
- Deconvo.m: a Richardson-Lucy deconvolution algorithm implementation in Matlab, written by Dr. E.P. Bellido [1];
- Python HyperSpy library: a Python-based library for spectrum processing for EELS and other techniques [2].

4.5 Non-Testing Based Verification

The following non-testing based verification methods will be used to assess the performance of SpectrumImageAnalysisPy

- Code review: a detailed code review will be performed by an external party and feedback provided;
- User survey: a group of qualified users will be asked to process a sample dataset using the software and will be polled on their user experience. The group should include a number of people with different levels of experience in data processing and programming, as well as different computer systems.

5 System Test Description

This section lists the system tests to be performed to verify whether or not the program fulfills the functional requirements and to test how well it meets the non-functional requirements. Note that some of the tests for the functional requirements are unit tests and can be found in the Unit Test section.

5.1 Tests for Functional Requirements

Here the tests to verify whether or not SpectrumImageAnalysisPy meets the functional requirements are listed. In addition, some of the tests validate the algorithms chosen for use in SpectrumImageAnalysisPy.

5.1.1 Deconvolution

In this subsection, the tests for the performance of the Richardson-Lucy deconvolution algorithm will be explained.

1. Deconvolution error estimation - blank PSF

   Type: Functional, Dynamic, Automatic

   Initial State: Spectrum loaded into SpectrumImageAnalysisPy.

   Input: A known spectrum (for instance, the data used in the paper by Bellido *et al* [1], if it is available from the author). [Don't wait to ask for the information from Bellido. You should contact him or her now. This sounds like a nice test case. —SS] Otherwise a spectrum will be simulated. A range of integers will be used as input for the number of deconvolution iterations. The input of a point spread function to the deconvolution will be an array of ones.

   Output: A series of spectra deconvolved using the range of iterations given in the input, and a plot of the RMS error vs the number of iterations used.

   How test will be performed: A series of deconvolutions will be run on the artificial spectrum, with varying numbers of iterations to produce a set of deconvolved spectra. The difference between the deconvolved spectra and the original simulated spectrum will be calculated and the RMS error plotted as a function of number of iterations. Ideally the deconvolved spectra will be identical to the input spectrum, because we are using a null function as the PSF reference.

2. Deconvolution error estimation

   Type: Functional, Dynamic, Automatic

Initial State: Spectrum loaded into SpectrumImageAnalysisPy.

Input: A known spectrum convolved with a broadening function (for instance, the data used in the paper by Bellido *et al* [1], if it is available from the author). Otherwise a spectrum will be simulated and convolved with a broadening function (such as a Lorentzian peak, a Gaussian peak, or an experimental point spread function). A range of integers will be used as input for the number of deconvolution iterations. The broadening function itself is also an input to the deconvolution.

Output: A series of spectra deconvolved using the range of iterations given in the input, and a plot of the RMS error vs the number of iterations used.

How test will be performed: A series of deconvolutions will be run on the artificially broadened spectrum, using the broadening function as the point spread function, with varying numbers of iterations to produce a set of deconvolved spectra. The difference between the deconvolved spectra and the original simulated spectrum (before convolution with a broadening function) will be calculated and the root mean square error plotted as a function of number of iterations. The same data will be deconvolved using the HyperSpy toolbox and the Matlab code listed in the Verification Tools section.

3. Deconvolution signal-to-noise ratio estimation

   Type: Functional, Dynamic, Automatic

   Initial State: Spectrum loaded into SpectrumImageAnalysisPy.

   Input: A known spectrum convolved with a broadening function (for instance, the data used in the paper by Bellido *et al* [1], if it is available from the author). Otherwise a spectrum will be simulated and convolved with a broadening function (such as a Lorentzian peak, a Gaussian peak, or an experimental point spread function). A range of integers will be used as input for the number of deconvolution iterations. The broadening function itself is also an input to the deconvolution.

   Output: A series of spectra deconvolved using the range of iterations given in the input, and a plot of the SNR vs the number of iterations used.

How test will be performed: A series of deconvolutions will be run on the artificially broadened spectrum, using the broadening function as the point spread function, with varying numbers of iterations to produce a set of deconvolved spectra. The difference between the deconvolved spectra and the original simulated spectrum (before convolution with a broadening function) will be calculated and the signal-to-noise ratio plotted as a function of number of iterations. The test will follow the SNR estimation used in [1], taking the variance of the signal every five data points.

5.1.2 File Export

In this subsection, the tests for checking that the program writes files correctly will be explained.

1. Image export

   Type: Functional, Dynamic, Manual

   Initial State: Spectrum image loaded into SpectrumImageAnalysisPy. The spectrum image to be used for this test may, for example, consist of float values from 0 to 1, reshaped into the size $(2, 2, 10)$, with an energy range from $0 - 9$ eV.

   Input: A selected range in the spectrum (eg, 1 to 2 eV).

   Output: An *Image.png* image written to the filesystem corresponding to the signal averaged over the selected energy range for each pixel.

   How test will be performed: Using a 3D dataset, the test will request the return value of an (x, y) image averaged over the chosen spectral range and export this to an *Image.png* file. No errors should be raised. The user must check that the range averaged over is the same as the range selected within the data limits.
   [Not sure if I can change these tests into automatic tests. I could read the file back into the program, but then I'm testing both the export and import functions and can't separate errors from one, so I made the test manual. —Author] [You could do this test the first time as a manual test, and then change it into an automated test. It could become part of your regression testing. The idea is that you would determine with confidence that Image.png is very likely correct. You

can then test any future executions to make sure that the generated Image.png matches the "golden" version. This kind of test doesn't prove that the future versions of your software are correct, but it does show whether something you have done has altered the output. This is what is done with VTK. They have nightly builds and verify that any changes have not altered their outputs. (If the output is altered, I'm sure they consider the possibility that the "golden" version had an error. After a quick google search, I found some code that vtk uses for their unit tests. This might be too much work for you to take on in the scope of this course, but it is worth thinking about. —SS]

2. Image export - GUI

   Type: Functional, Dynamic, Manual

   Initial State: Spectrum image loaded into SpectrumImageAnalysisPy. The spectrum image to be used for this test may, for example, consist of float values from 0 to 1, reshaped into the size $(10, 10, 25)$, with an energy range from $0 - 9$ eV.

   Input: The user must interact with the GUI to select an energy range in the spectrum from which to create and export an image.

   Output: An *Image.png* image written to the filesystem corresponding to the image averaged over the selected spectrum energy range.

   How test will be performed: Using a 3D dataset, the test will display the GUI and ask the user to select an energy range in the spectrum and run the command to export an image to file. The user must inspect the *Image.png* file to ensure that the correct spectrum was exported from the correct pixel.

3. Spectrum export

   Type: Functional, Dynamic, Manual

   Initial State: Spectrum image loaded into SpectrumImageAnalysisPy. The spectrum image to be used for this test may, for example, consist of float values from 0 to 1, reshaped into the size $(2, 2, 10)$, with an energy range from $0 - 9$ eV.

   Input: A selected range in the image (*e.g.*, pixel $(0, 0)$).

Output: A *Spectrum.csv* file written to the filesystem corresponding to the spectrum averaged over the selected area.

How test will be performed: Using a 3D dataset, the test will request the return value of a spectrum averaged over the chosen area. No errors should be raised. The user must inspect the *Spectrum.csv* file to ensure that the correct spectrum was exported from the correct pixel, with the correct energy range.

4. Spectrum export - GUI

   Type: Functional, Dynamic, Manual

   Initial State: Spectrum image loaded into SpectrumImageAnalysisPy. The spectrum image to be used for this test may, for example, consist of float values from 0 to 1, reshaped into the size $(10, 10, 25)$, with an energy range from $0 - 9$ eV.

   Input: The user must interact with the GUI to select an area on the image from which to create and export the spectrum.

   Output: A *Spectrum.csv* image written to the filesystem corresponding to the spectrum averaged over the selected area.

   How test will be performed: Using a 3D dataset, the test will display the GUI and ask the user to select an area in the image and run the command to export a spectrum to file. The user must inspect the *Spectrum.csv* file to ensure that the correct spectrum was exported from the correct pixel.

5.2 Tests for Nonfunctional Requirements

5.2.1 Program speed

1. Deconvolution calculation time - SI size

   Type: Performance, Dynamic, Automatic

   Initial State: Spectrum loaded into SpectrumImageAnalysisPy.

   Input: A known spectrum convolved with a broadening function (for instance, the data used in the paper by Bellido *et al* [1], if it is available from the author), and transformed into a spectrum image data format

(of 3D shape). Otherwise a spectrum will be simulated and convolved with a broadening function (such as a Lorentzian peak, a Gaussian peak, or an experimental point spread function). The broadening function itself is also an input to the deconvolution. The spectrum image size will be varied from (1 by 1 by K) to (1000 by 1000 by K) with a minimum of 10 intermediate values. 25 iterations will be performed for each spectrum image size. A typical value for K might be 2048 or 1024, depending on the dataset used.

Output: A plot of the calculation time required vs the size of the spectrum image used.

How test will be performed: The Richardson-Lucy deconvolution algorithm will be run on a series of spectrum images of varying size. 25 iterations will be run on each spectrum image and the time required to finish the calculations recorded and plotted against the number of (x, y) pixels in the spectrum image. The test will be repeated for the deconvolution algorithm written in the HyperSpy toolbox, for comparison of the plots.

2. Deconvolution calculation time - iterations

   Type: Performance, Dynamic, Automatic

   Initial State: Spectrum loaded into SpectrumImageAnalysisPy.

   Input: A known spectrum convolved with a broadening function (for instance, the data used in the paper by Bellido *et al* [1], if it is available from the author), and transformed into a spectrum image data format (of 3D shape). Otherwise a spectrum will be simulated and convolved with a broadening function (such as a Lorentzian peak, a Gaussian peak, or an experimental point spread function). The broadening function itself is also an input to the deconvolution. The spectrum image size will be set at (500 by 500 by E) and a the number of iterations will be changed from 1 to 500 with at least 10 intermediate steps. A typical value for E might be 2048 or 1024, depending on the dataset used.

   Output: A plot of the calculation time required vs the number of iterations used.

How test will be performed: The Richardson-Lucy deconvolution algorithm will be run repeatedly on a spectrum image with varying numbers of iterations. The time required to finish the calculations will be recorded and plotted against the number of (x, y) pixels in the spectrum image. The test will be repeated for the deconvolution algorithm written in the HyperSpy toolbox, for comparison of the plots.

3. Normalization calculation time

   Type: Performance, Dynamic, Automatic

   Initial State: Spectrum loaded into SpectrumImageAnalysisPy.

   Input: A known spectrum (for instance, an artificially created spectrum, a range of values from 0 to the number of pixels used in the spectrum image), transformed into a spectrum image data format (of 3D shape).The spectrum image size will be varied from (1 by 1 by K) to (1000 by 1000 by K) with a minimum of 10 intermediate values. A typical value for K might be 2048 or 1024, depending on the dataset used.

   Output: A plot of the calculation time required vs the size of the spectrum image used.

   How test will be performed: The normalization algorithm will be run on a series of spectrum images of varying size, normalizing to the integral of the full spectrum. The time required to finish the calculations will be recorded and plotted against the number of (x, y) pixels in the spectrum image.

4. Gain correction calculation time

   Type: Performance, Dynamic, Automatic

   Initial State: Spectrum loaded into SpectrumImageAnalysisPy.

   Input: A known spectrum (for instance, an artificially created spectrum, a range of values from 0 to the number of pixels used in the

spectrum image), transformed into a spectrum image data format (of 3D shape).The spectrum image size will be varied from (1 by 1 by K) to (1000 by 1000 by K) with a minimum of 10 intermediate values. A typical value for K might be 2048 or 1024, depending on the dataset used. The gain correction factor will be given as a range of integers from 0 to $K - 1$.

Output: A plot of the calculation time required vs the size of the spectrum image used.

How test will be performed: The gain correction function will be run on a series of spectrum images of varying size. The time required to finish the calculations will be recorded and plotted against the number of (x, y) pixels in the spectrum image.

5. Background subtraction calculation time

   Type: Performance, Dynamic, Automatic

   Initial State: Spectrum loaded into SpectrumImageAnalysisPy.

   Input: A known spectrum (for instance, the data used in the paper by Bellido *et al* [1], if it is available from the author), transformed into a spectrum image data format (of 3D shape).The spectrum image size will be varied from (1 by 1 by K) to (1000 by 1000 by K) with a minimum of 10 intermediate values. A typical value for K might be 2048 or 1024, depending on the dataset used. The background correction will be given as a an array of ones of size K.

   Output: A plot of the calculation time required vs the size of the spectrum image used.

   How test will be performed: The gain correction function will be run on a series of spectrum images of varying size. The time required to finish the calculations will be recorded and plotted against the number of (x, y) pixels in the spectrum image.

[I did not see the Usability NFR explicitly addressed. You have information on this throughout your document, but you really should have a specific

test plan related to it. You give a great user survey and some information on who will be recruited, but it would be nice to see the specific task that they will be asked to perform. It would also be nice to know what resources they will be given, in terms of documentation, assistance from you etc. —SS]

5.3 Traceability Between Test Cases and Requirements

This section provides a matrix with the relationships between the functional requirements in the SRS and the System Tests outlined above. The non-functional requirements are addressed in the section Tests for Nonfunctional Requirements. Note that testing for some of the non-functional requirements will be addressed with the user survey and code review.

	1	2	3	4	5	6	7	Unit Test
SI input								X
Spectrum input								X
Input dimensions								X
SI slicing				X	X			X
SI area						X	X	X
Deconvolution	X	X	X					X
Normalization								X
Background								X
Gain								X

Table 1: Traceability Matrix Showing the Connections Between Test Cases and Requirements

6 Unit Testing Plan

6.1 Inputs

This section details the unit tests concerning the import of spectrum images and spectrum datasets (R1, R2, R3) [Very nice to have explicit cross-references between documents. To keep the information up to date, consider using make to build your documentation. —SS]

Spectrum Image Input

1. SI Data array with energy range (EELS)

   Type: Functional, Dynamic, Unit

   Initial State: None

   Input: 3D data array, present in memory; use standard calibration variabilities
   For example, the following values might be used to create an SI:

   - $x = [0, 1]$
   - $y = [0, 1]$
   - Intensity = ones(size$(2, 2, 10)$)
   - x calibration = 1 nm/pixel
   - energy range = range$(0, 10)$ eV

   Output: Spectrum Image stored within SpectrumImageAnalysisPy; the first and second axes should correspond to spatial dimensions, while the third axis should correspond to a spectral dimension.

   How test will be performed: Create a 3D data array and attempt to initialize a spectrum image in SpectrumImageAnalysisPy. No errors should be raised. The spectrum image axes should be read in the correct order (x, y, E), and the calibrations should be applied to the correct axes. Addresses Functional Requirement.

2. SI Data array with dispersion (EELS)

   Type: Functional, Dynamic, Unit

   Initial State: None

Input: 3D data array, present in memory; use standard calibration variabilities
For example, the following values might be used to create an SI:

- $x = [0, 1]$
- $y = [0, 1]$
- Intensity = ones(size$(2, 2, 10)$)
- x calibration = 1 nm/pixel
- dispersion = 0.01 eV/pixel
- $E(0) = -2$ eV

Output: Spectrum Image stored within SpectrumImageAnalysisPy; the first and second axes should correspond to spatial dimensions, while the third axis should correspond to a spectral dimension.

How test will be performed: Create a 3D data array and attempt to initialize a spectrum image in SpectrumImageAnalysisPy. No errors should be raised. The spectrum image axes should be read in the correct order (x, y, E), and the calibrations should be applied to the correct axes.

3. SI Data array with wavelength range (CL)

   Type: Functional, Dynamic, Unit

   Initial State: None

   Input: 3D data array, present in memory; use standard calibration variabilities
   For example, the following values might be used to create an SI:

   - $x = [0, 1]$
   - $y = [0, 1]$
   - Intensity = ones(size$(2, 2, 10)$)
   - x calibration = 1 nm/pixel
   - wavelength range = range$(0, 10)$ nm

Output: Spectrum Image stored within SpectrumImageAnalysisPy; the first and second axes should correspond to spatial dimensions, while the third axis should correspond to a spectral dimension.

How test will be performed: Create a 3D data array and attempt to initialize a spectrum image in SpectrumImageAnalysisPy. No errors should be raised. The spectrum image axes should be read in the correct order (x, y, E), and the calibrations should be applied to the correct axes.

4. SI .dm3 file

   Type: Functional, Dynamic, Unit

   Initial State: None

   Input: A .dm3 file, containing a fabricated spectrum image. The .dm3 file will be created using Digital Micrograph (Gatan Microscopy Suite Software) software [3] with the following parameters:

   - $x = [0, 1]$
   - $y = [0, 1]$
   - Intensity = ones(size$(2, 2, 10)$)
   - x calibration = 1 nm/pixel
   - dispersion = 0.01 eV/pixel
   - $E(0) = -2$ eV

   Output: EELS Spectrum Image stored within SpectrumImageAnalysisPy, with metadata assigned to appropriate values

   How test will be performed: Read in the .dm3 file with SpectrumImageAnalysisPy. Check that no errors are raised. Display the spectrum image to manually check that the data was read in correctly. Check spatial and spectral calibration to verify metadata was assigned correctly.

   [There is a potential problem with the .dm3 file, since there are many version of Digital Micrograph out there and many different set-ups, all of which may format their .dm3 slightly differently. Not sure on the best way to test this, without asking several other labs for pieces of

data (which are not usually shared readily) and trying out all of them. —Author]

[Can you simplify your life and, at least for the time being, select one dm3 standard and make that the one you support and test? —SS]

5. SI .h5 file

   Type: Functional, Dynamic, Unit.

   Initial State: None

   Input: .h5 file, containing an acquired or fabricated spectrum image. The .h5 spectrum image file originates from Odemis software [4], it may be possible to fabricate a spectrum image using the scripting interface, using the following values for the calibration parameters, otherwise it may be necessary to use an acquired data file.

   - $x = [0, 1]$
   - $y = [0, 1]$
   - Intensity = ones(size$(2, 2, 10)$)
   - x calibration = 1 nm/pixel
   - wavelength range = range$(0, 10)$ nm

   Output: CL Spectrum Image stored within SpectrumImageAnalysisPy, with metadata assigned to appropriate values.

   How test will be performed: Read in the .h5 file with SpectrumImageAnalysisPy. Check that no errors are raised. Check spatial and spectral calibration to verify metadata was assigned correctly.

6. SI Data array with dispersion (EELS) - complex numbers

   Type: Functional, Dynamic, Unit

   Initial State: None

   Input: 3D data array, present in memory; use standard calibration variabilities
   For example, the following values might be used to create an SI:

   - $x = [0, 1]$
   - $y = [0, 1]$

- Intensity = ones(size(2, 2, 10)) + j * ones(size(2, 2, 10))
- x calibration = 1 nm/pixel
- dispersion = 0.01 eV/pixel
- $E(0) = -2$ eV

Output: Spectrum Image stored within SpectrumImageAnalysisPy; the first and second axes should correspond to spatial dimensions, while the third axis should correspond to a spectral dimension.

How test will be performed: Create a 3D data array and attempt to initialize a spectrum image in SpectrumImageAnalysisPy. No errors should be raised. The spectrum image axes should be read in the correct order (x, y, E), and the calibrations should be applied to the correct axes.

Spectrum Input

1. Spectrum Data array

   Type: Functional, Dynamic, Unit.

   Initial State:

   Input: 1D data array, present in memory. For example, the following data may be used to initialize a Spectrum:

   - intensity = ones(size(10)) counts
   - energy range = range(0, 10) eV

   Output: Spectrum stored within SpectrumImageAnalysisPy, with data assigned to appropriate values.

   How test will be performed: Create a 1D data array and attempt to read it with SpectrumImageAnalysisPy. No errors should be raised. The unit test will check that the intensity and energy range are assigned to the appropriate variables, as well as the spectral calibration.

2. Spectrum .csv file

   Type: Functional, Dynamic, Unit.

   Initial State:

Input: .csv file, containing a 1D spectrum. The following data may be used to create a spectrum csv file:

- Intensity = ones(size(10)) counts
- energy range = range$(0, 10)$ eV

Output: Spectrum stored within SpectrumImageAnalysisPy, with data assigned to appropriate values

How test will be performed: A csv file containing 1D spectrum data will be imported. No errors should be raised. The unit test will check that the intensity and energy range are assigned to the appropriate variables, as well as the spectral calibration.

3. Spectrum Data array - complex numbers

   Type: Functional, Dynamic, Unit.

   Initial State:

   Input: 1D data array, present in memory. For example, the following data may be used to initialize a Spectrum:

   - energy values = ones(size(10)) + $j *$ ones(size(10)) counts
   - energy range = range$(0, 10)$ eV

   Output: Error message - input must be real

   How test will be performed: Create a 1D data array and attempt to read it with SpectrumImageAnalysisPy. An error should be raised.

4. Spectrum Data array - 2D array

   Type: Functional, Dynamic, Unit.

   Initial State:

   Input: 2D data array, present in memory. For example, the following data may be used to initialize a Spectrum:

   - energy values = ones(size$(2, 2)$) counts
   - energy range = range$(0, 2)$ eV

   Output: Error message - input must be 1D

   How test will be performed: Create a 1D data array and attempt to read it with SpectrumImageAnalysisPy. An error should be raised.

6.2 Data Extraction

This section details the unit tests concerning the extraction of images or spectra from a spectrum image (R4, R5)

Energy range slicing The spectrum image to be used for these tests will be the same for all tests in this section and may, for example, consist of float values from 0 to 1, reshaped into the size $(2, 2, 10)$, with an energy range from $0 - 9$ eV.

1. Extract (x, y) slice from spectrum

   Type: Functional, Dynamic, Unit

   Initial State: Spectrum image exists in SpectrumImageAnalysisPy.

   Input: Two values within the spectrum range calibration values. For example, in a spectrum image with an energy range from $0 - 9$ eV, the input values might be 5 and 7. the existing spectrum image must have different values of intensity at each pixel and each channel along the energy axis.

   Output: An (x, y) image averaged over the spectral range specified by the input to the test.

   How test will be performed: Using a 3D dataset, such as those used in the Spectrum image 3D dataset unit test, the test will request the return value of an (x, y) image averaged over the chosen spectral range. No errors should be raised. Check that the range averaged over is the same as the range selected.

2. Extract (x, y) slice from spectrum, slice less than any value in the energy range

   Type: Functional, Dynamic, Unit

   Initial State: Spectrum image exists in SpectrumImageAnalysisPy.

   Input: Two values within the spectrum range calibration values. For example, in a spectrum image with an energy range from $0 - 9$ eV, the input values might be -2 and 5. the existing spectrum image must have different values of intensity at each pixel and each channel along the energy axis.

Output: An (x, y) image averaged over the spectral range specified by the input to the test and the minimum value of the energy range in the spectrum image.

How test will be performed: Using a 3D dataset, such as those used in the Spectrum image 3D dataset unit test, the test will request the return value of an (x, y) image averaged over the chosen spectral range. No errors should be raised. Check that the range averaged over is the same as the range selected within the data limits.

3. Extract (x, y) slice from spectrum, slice more than any value in the energy range

   Type: Functional, Dynamic, Unit

   Initial State: Spectrum image exists in SpectrumImageAnalysisPy.

   Input: Two values within the spectrum range calibration values. For example, in a spectrum image with an energy range from $0 - 9$ eV, the input values might be 5 and 12. the existing spectrum image must have different values of intensity at each pixel and each channel along the energy axis.

   Output: An (x, y) image averaged over the spectral range specified by the input to the test and the maximum value of the energy range in the spectrum image.

   How test will be performed: Using a 3D dataset, such as those used in the Spectrum image 3D dataset unit test, the test will request the return value of an (x, y) image averaged over the chosen spectral range. No errors should be raised. Check that the range averaged over is the same as the range selected within the data limits.

4. Extract (x, y) slice from spectrum, slice completely outside the energy range

   Type: Functional, Dynamic, Unit

   Initial State: Spectrum image exists in SpectrumImageAnalysisPy.

   Input: Two values within the spectrum range calibration values. For example, in a spectrum image with an energy range from $0 - 9$ eV, the input values might be -5 and -2. the existing spectrum image must have different values of intensity at each pixel and each channel along the energy axis.

Output: A warning should be raised that the range selected is outside the range for which there is data available.

How test will be performed: Using a 3D dataset, such as those used in the Spectrum image 3D dataset unit test, the test will request the return value of an (x, y) image averaged over the chosen spectral range. A warning should be raised and no image returned.

5. Extract (x, y) slice from spectrum, backwards slice

   Type: Functional, Dynamic, Unit

   Initial State: Spectrum image exists in SpectrumImageAnalysisPy.

   Input: Two values within the spectrum range calibration values. For example, in a spectrum image with an energy range from $0 - 9$ eV, the input values might be 7 and 5. the existing spectrum image must have different values of intensity at each pixel and each channel along the energy axis.

   Output: An (x, y) image averaged over the spectral range specified by the input to the test.

   How test will be performed: Using a 3D dataset, such as those used in the Spectrum image 3D dataset unit test, the test will request the return value of an (x, y) image averaged over the chosen spectral range (regardless of the order of selection, the program should behave exactly the same as if the input range was 5 to 7). No errors should be raised. Check that the range averaged over is the same as the range selected (ie, by knowing what data the spectrum image contains).

Area spectrum extraction The spectrum image to be used for these tests will be the same for all tests in this section and may, for example, consist of float values from 0 to 1, reshaped into the size (100, 100, 10), with an energy range from $0 - 9$ eV.

1. Extract area - square

   Type: Functional, Dynamic, Unit

   Initial State: Spectrum image exists in SpectrumImageAnalysisPy.

   Input: Four (x, y) values defining the vertices of a square (for instance, $(20, 20)$, $(20, 35)$, $(35, 35)$, $(35, 20)$). A spectrum image, as defined above for this section.

Output: An spectrum, averaged over the (x, y) area defined.

How test will be performed: Using a 3D dataset, the test will request the return value of a spectrum averaged over the chosen area. No errors should be raised. Check that the correct area was used in the average (correct solution is known by manual deduction from the spectrum image).

2. Extract area - triangle

   Type: Functional, Dynamic, Unit

   Initial State: Spectrum image exists in SpectrumImageAnalysisPy.

   Input: A range of (x, y) values defining a triangle (for instance, $(20, 20)$, $(35, 35)$, $(35, 20)$). A spectrum image, as defined above for this section.

   Output: A spectrum, averaged over the (x, y) values defined.

   How test will be performed: Using a 3D dataset, the test will request the return value of a spectrum averaged over the chosen area. No errors should be raised. Check that the correct area was used in the average (correct solution is known by manual deduction from the spectrum image).

3. Extract area - line

   Type: Functional, Dynamic, Unit

   Initial State: Spectrum image exists in SpectrumImageAnalysisPy.

   Input: A range of (x, y) values defining a line (for instance, $(20, 20)$, $(35, 35)$). A spectrum image, as defined above for this section.

   Output: A warning that a 2D area does not exist when only two points are used to define the area.

   How test will be performed: Using a 3D dataset, the test will request the return value of a spectrum averaged over the chosen area. A warning should be raised and nothing returned.

4. Extract area - outside image

   Type: Functional, Dynamic, Unit

   Initial State: Spectrum image exists in SpectrumImageAnalysisPy.

Input: A range of (x, y) values defining a line (for instance, $(-20, 20)$, $(35, 135)$). A spectrum image, as defined above for this section.

Output: A spectrum, averaged over the (x, y) values defined, within the image dimensions. Dimensions outside the image should be rounded to the closest value within the image range (0-99).

How test will be performed: Using a 3D dataset, the test will request the return value of a spectrum averaged over the chosen area. No errors should be raised. Check that the correct area was used in the average (correct solution is known by manual deduction from the spectrum image). The program should behave as though any image ranges outside the available range are at the closest border of the image (*e.g.* -20 rounds to 0, 135 rounds to 99).

6.3 Normalization

This section details the unit tests concerning the normalization of a spectrum (IM1, R7)

Spectrum normalization

1. Normalization - integral

   Type: Functional, Dynamic, Unit

   Initial State: None

   Input: 1D spectrum object, with an energy range from $0 - 9$ eV and intensity values of 1 at each spectrum channel.

   Output: Normalized spectrum, with an intensity value of 0.1 in each spectrum channel.

   How test will be performed: Feed the spectrum object into the normalization function and request normalization to the full integral. No errors should be raised and a normalized spectrum should be returned.

2. Normalization - select channel

   Type: Functional, Dynamic, Unit

   Initial State: None

Input: 1D spectrum object, with an energy range from $0 - 9$ eV and intensity values of 1 at each spectrum channel, except for the fifth channel, which will have a value of 10.

Output: Normalized spectrum, with an intensity value of 1 in the fifth channel and a value of 0.1 everywhere else.

How test will be performed: Feed the spectrum object into the normalization function and request normalization to the fifth channel. No errors should be raised and a normalized spectrum should be returned.

3. Normalization - integral is zero

   Type: Functional, Dynamic, Unit

   Initial State: None

   Input: 1D spectrum object, with an energy range from $0 - 9$ eV and intensity values of 0 at each spectrum channel.

   Output: Error message, stating that normalization cannot be performed on this spectrum because of a divide-by-zero error.

   How test will be performed: Feed the spectrum object into the normalization function and request normalization to the fifth channel. An error should be raised.

6.4 Background Correction

This section details the unit tests concerning the background correction of a spectrum (IM3, R8)

Spectrum-background correction

1. Background subtraction

   Type: Functional, Dynamic, Unit

   Initial State: None

   Input: 1D spectrum object, with an energy range from $0 - 9$ eV and intensity values of 1 at each spectrum channel. Background spectrum, with the same energy range and intensity values of 0.1 at each spectrum channel.

Output: Corrected spectrum, with an intensity value of 0.9 in each spectrum channel.

How test will be performed: Feed the spectrum object and background spectrum object into the background correction function. No errors should be raised and a background-corrected spectrum should be returned.

2. Background subtraction - different size

   Type: Functional, Dynamic, Unit

   Initial State: None

   Input: 1D spectrum object, with an energy range from $0 - 9$ eV and intensity values of 1 at each spectrum channel. Background spectrum, with an energy range of $0 - 8$ eV and intensity values of 0.1 at each spectrum channel (one channel shorter than the 1D spectrum object).

   Output: An error message, stating that the background correction spectrum is not the right size and the correction cannot be applied.

   How test will be performed: Feed the spectrum object and background spectrum object into the background correction function. An error should be raised.

6.5 Gain Correction

This section details the unit tests concerning the gain correction of a spectrum (IM4, R9)

Spectrum-gain correction

1. Gain correction

   Type: Functional, Dynamic, Unit

   Initial State: None

   Input: 1D spectrum object, with an energy range from $0 - 9$eV and intensity values of 1 at each spectrum channel. Gain correction spectrum, with the same energy range and intensity values of 0.1 at each spectrum channel.

Output: Corrected spectrum, with an intensity value of 0.1 in each spectrum channel.

How test will be performed: Feed the spectrum object and gain correction spectrum object into the gain correction function. No errors should be raised and a gain-corrected spectrum should be returned.

2. Gain correction - zeros

   Type: Functional, Dynamic, Unit

   Initial State: None

   Input: 1D spectrum object, with an energy range from $0 - 9$eV and intensity values of 1 at each spectrum channel. Gain correction spectrum, with the same energy range and intensity values of 0 at each spectrum channel.

   Output: Corrected spectrum, with an intensity value of 0 in each spectrum channel and a Warning that the gain correction is composed of zeros and so nullifies the whole spectrum.

   How test will be performed: Feed the spectrum object and gain correction spectrum object into the gain correction function. A warning should be raised and a gain-corrected spectrum should be returned.

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.

[2] F. de la Peña, T. Ostasevicius, V. T. Fauske, P. Burdet, P. Jokubauskas, M. Nord, E. Prestat, M. Sarahan, K. E. MacArthur, D. N. Johnstone, J. Taillon, J. Caron, T. Furnival, A. Eljarrat, S. Mazzucco, V. Migunov, T. Aarholt, M. Walls, F. Winkler, B. Martineau, G. Donval, E. R. Hoglund, I. Alxneit, I. Hjorth, L. F. Zagonel, A. Garmannslund, C. Gohlke, I. Iyengar, and H.-W. Chang, "hyperspy/hyperspy: Hyperspy 1.3," May 2017.

[3] Gatan, Inc., "Gatan Microscopy Suite Software."

[4] D. BV, "ODEMIS: Integrated software for microscopy solutions | DELMIC."

[5] Office of Safety and Mission Assurance, "Software Formal Inspections Guidebook," Aug. 1993.

[6] Karl E. Wiegers, "Generic Checklist for Code Reviews," 2001.

[7] Erik Dietrich, "Creating Your Code Review Checklist," Sept. 2015.

7 Appendix

This appendix contains questions for a Code Review and questions for a user survey.

7.1 Code Review Checklist

This section contains some questions to be covered during a Code Review process. An effort was made to keep the number of items low to allow more time for a thorough review of these items without requiring an overly lengthy time commitment from the reviewer. Questions were inspired by [5, 6, 7].

- Summarize in your own words what each function or unit of code does, based on inspection of the code.
- Is the documentation clear and readable for each function or unit of code?
- Does the described functionality match with your written description?
- Are all variables and functions named clearly and meaningfully?
- Are all cases covered in IF/ELSEIF loops?
- Do all branches in conditionals make sense (logic and resulting action is correct)?
- Are the specifications and requirements complete and correctly implemented?
- Is the user interface clear and easy to use?
- Does each unit of code have a single purpose, or can it be split into multiple functions?
- What is the expected memory usage for each function? Can it be reduced? (*e.g.* are unnecessary copies of arrays being created?)
- Are array slicing references in the correct order? (*e.g.*, check correct dimension is used)
- Does it appear to be easy to add new modules or functions?

7.2 User Survey

This section contains some sample questions which may be used to create a user survey of the software. Effort was made to keep the number of questions small and to address non-functional requirements.

- What operating system do you use?
- Did the software experience any noticeable lag while you were using the GUI interface?
- Where you able to use the interactive display easily, or did you find the commands difficult to remember and use?
- Could you easily read everything you needed to on the data display? (please consider font, font size, colour, plot lines, colour maps, etc.)
- What aspects of using the software did you find most intuitive? Least intuitive?
- Do you think you will be able to use this software given only the documentation available and no other training?
- On a scale of 1-10, did you find the software easy to use? (0: impossible to use; 10: very simple and intuitive)
- On a scale of 1-10, did you find the software enjoyable to use? (0: hated the software and resented the waste of time; 10: fully enjoyed the experience and I look forward to using this to process my data again)
- Do you think the software will be useful to you in your work?
- Do you have any comments or suggestions for further improvements? What would make this software more useful to you?

[Great work! —SS]