SpectrumImageAnalysisPy

Isobel Bicket

October 30, 2017

1 Revision History

Date	Version	Notes
October 30, 2017	1.0	Initial draft

2 Symbols, Abbreviations and Acronyms

symbol	description
RMS	Root Mean Square
T	Test

See the SRS document Table of Symbols and Table of Units for further definitions.

Contents

List of Tables

List of Figures

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.

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.

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.

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.

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

   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). 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

   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.

3. Deconvolution signal-to-noise ratio estimation Type: Functional, Dynamic

   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.

5.2 Tests for Nonfunctional Requirements

5.2.1 Program speed

1. Deconvolution calculation time - SI size

   Type: Performance, Dynamic

   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.

2. Deconvolution calculation time - iterations

   Type: Performance, Dynamic

   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.

3. Normalization calculation time

   Type: Performance, Dynamic

   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

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

   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.

5.2.2 Area of Testing2

...

5.3 Traceability Between Test Cases and Requirements

6 Unit Testing Plan

6.1 Inputs

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]

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.

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 $0 - 10$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).

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 Microscopy Suite Software."

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

7 Appendix

This is where you can place additional information.

7.1 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.

7.2 Code Review Checklist

- 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? (*eg* are unnecessary copies of arrays being created?)
- Are array slicing references in the correct order? (eg, check correct dimension is used)

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

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