Mass Spect in Knime - Mass Spectrometry Analytics Extension for KNIME - Documentation

1) INTRODUCTION

This is the supporting information to the Knime software extension “Mass Spectrometry in Knime”. This software is built for Knime - the Konstanz Information Miner, and aims to bring methods for preprocessing, visualizing, and clustering Mass Spectrometry Imaging (MSI) datasets, specifically in the imzML data format.

1.1) Purpose

In this project, we aim to build upon the existing nodes in Knime, so that our new nodes can be used with current nodes in the node repository. We aim to make our tool extensible so that as new technologies appear in this field, we can add tools to our extension. Furthermore, we only have three months to focus on this tool and thus we aim to make it extensible as some necessary nodes may not have been implemented. The purposes of this node are to create an easy to use and open source tool for the analysis and visualization of Mass Spectrometry data, which brings us to why we chose to work in Knime. Knime is extremely easy to use with its drag and drop GUI, meaning users don’t have to write code. The typical use case of this software extension is in the fields of biology and chemistry, thus we wish to get rid of the difficulty of coding which is sometimes necessary for analytics to be done.

We wish to make our product accessible, thus we have chosen to work with the common data format imzML, which most MSI data formats have at least one method of converting to, either by their proprietary code, or by third party from the community. We also aim to follow the Knime noding guidelines V2.5, in order to allow our software to run as the user might expect.

1.2) Scope

The scope of this project is to explore and build basic pre-pcoessing methods typical to the field of MSI and signal processing. We will explore visualization techniques used in the pre existing software “Spectral Analysis” and “MSIQuant”. Both of these software applications have more

sophisticated visualization tools than we will have, due to the short time that we have, so for our purposes we will implement basic nodes. The scope of this project involves exploring clustering techniques in MSI. Our aim to explore k-means, DBSCAN, Normalized Cuts Image Segmentation, Self Organizing Maps and ToMaTo for Topological clustering.

1.3) System Description

Our system is a software extension in the platform Knime, which is a workflow management tool which is build in the domain of data analysis. Knime is a GUI based application which encapsulates complicated algorithms into a “node” which allows us to drag and drop them and connect them to our nodes and feed in data. Thus making very complex coding much easier, as all we have to do is choose the parameters for the algorithm in the nodes “dialog”, and then connect it to our nodes which inputs data.

1.4) Points of Contact

The major participants in this project are:

- Andrew P Talbot
    - Project Lead
    - APT467@student.bham.ac.uk

- Iain B Styles
    - Project supervisor
    - I.B.Styles@bham.ac.uk

1.7) Project Structure

The version control that we have used for our project is git, the url to our repository is here :

- https://git-teaching.cs.bham.ac.uk/mod-msc-proj-2017/apt467/

Here, we will explain the structure of the directories in out gir repository :

- week_x
    - Contains information about the project meeting we had with the project supervisor in the week “x” in the project. It contains any useful papers that we have encountered in that week, and a list of goals, and problems that we encounter during the project.
- presentation
    - Contains information used in the project demonstration which occurs in the final few weeks of the project.
- report

    - Contains information used in the project report, although not the final pdf.
- MassSpectInKnime2_1.0.1.jar
    - Is the final jar file that we will submit as our extension to Knime.
- Documentation
    - This will contain any relevant documentation used in the project, specifically a link to this Google Docs.

- knime_dev
    - Contains the two following sub-directories:
        - MassSpectInKnime - which stores source code and binary code for our Knime extension.
        - NormalizedCuts2 - which stores source code for our implementation of Normalized Cuts, we didn't implement it in the other project folder because for some reason on our machine, we could not get GUI's to appear in the other project, thus we worked outside this folder as we wanted to see visualizations of the clustering.

As we see above, to see the source code for our software extension of Mass Spectrometry Data Analytics in Knime, we must go to our version control, then follow the path "knime_dev/MassSpectInKnime/". Here, we will see the following directories.

- src
    - Contains source code for all nodes (including deprecated nodes). Also contains packages with logic used in the nodes.
- data
    - Contains small Mass Spectrometry datasets which have been used for testing purposes throughout the project.
- docs
    - Contains all documentation for this KNIME plugin.
- icons
    - Contains icons used for the KNIME nodes, typically for the icons displayed over the nodes.
- images
    - Contains images that have been used when testing the Normalized Cuts algorithm.
- lib
    - Contains third party libraries which this plugin is dependent on.
- META-INF
    - Contains MANIFEST.MF which describes how the plugin is built into a .jar file.
- test_workflows
    - Contains information of the test workflows used to test the KNIME nodes. To view the test workflows simply open KNIME with this directory as the root of the workspace.

2) ASSUMPTIONS AND CONSTRAINTS

2.1) Assumptions

Here are some assumptions that we have assumed for using and developing the software in this project.

2.1.1) Hardware

The hardware assumptions placed on this project are very minimal. This softare has been developed and used with the following hardware:

- 2.26GHz Intel Core 2 Duo processor with 3MB on-chip shared L2 cache running 1:1 with processor speed.
- NVIDIA GeForce 9400M 256 MB Graphics Card.
- 8 GB 1067 MHz DDR3.
- 250GB 5400-rpm Serial ATA hard disk drive.

So we assume hardware equal to or better than this, which should not be hard in 2018!

2.1.2) Software

This software was developed and used with the above hardware running Mac OS X El Capitan version 10.11.6 (15G1011). So we assume software as powerful as this both for development and usage.

The IDE used is Eclipse Neon 9.1, and the Java Runtime Environment used is 1.8.0_161. The Knime version used in testing was KNIME 3.5.3. Again, we assume versions of this or higher for each respective software.

2.2) Constraints

Time is a major constraint on this project, given that we have three months to research, build and test the software. A major constraint in the development of this project is that Knime data structures, which we are required to use, must be ran in an instance of Knime, or they will throw an exception will be thrown. Thus we must only test our code in Knime. We get around this by making our code modular, and only using Knime data structures where necessary.

3) CONTEXT AND DEVELOPMENT

During different phases of the project, we chose to alter our approach to software development. Before any software development was performed, we chose to spend some time researching

the Knime platform, and it's data structures. The initial phase of development involved an approach using rapid prototyping, aiming to get a few basic, working Knime nodes. The aim of this phase was to quickly learn in a practical sense how the Knime framework can be extended. This phase of rapid prototyping lasted for roughly 2 weeks, after which an agile approach was taken. We used the concept of sprints, by which we would make a list of tasks that we would aim to complete each week between meetings with my supervisor Iain Styles. For example, the first sprint was one week long, in which we aimed to implement nodes to be used for the reading of imzML data into Knime. Sometimes these sprints were completed on time, and others we had to dedicate more time to them, in particular the implementation of the Normalized Cuts algorithm. We aimed to complete sprints on nodes which had the most important functionality and were not dependent on other sprints first. For example, it was important that we completed reader nodes before building any nodes for processing or visualizations.

Overall, I believe that this approach to was successful and appropriate for the project, as we needed to spend some time learning the development cycle of a Knime node. In hindsight, I believe focusing only on an agile approach would have been superior. I believe that the rapid prototyping phase was in the grand scheme not as efficient as it relied on disposing of some source code and nodes.

4) SYSTEM REQUIREMENTS

Here, we outline some of the requirements that our software must follow as a whole, sort of global rules that must be followed by all nodes.

Functional Requirements :

1. The system must appear as a category in the Knime node repository.
2. The system must support the common Mass Spectrometry Imaging data format imzML.
3. The system must be built so that RowKey's are never altered from their respective row.
    3.1. Due to the spatial nature of MSI data, it is important that we maintain the spatial location associated with each spectrum, if we do not know what pixel the spectrum came from after processing, the results will not be as meaningful. As we store spectra in rows of the DataTable, we require a unique String key for each row. Our software uses a row key in the form ”(x,y)”, where x is the location in the width of the image of the spectrum, and y is the location in the height of the image. We have created an object KeyLogic to handle this and allow for easy access and creation of these keys. Some nodes existing in the repository concatenate numbers to the row keys, in order to ensure uniqueness, our class ignores this and handles it regardless.
4. The system must ensure the node invariant stated below at all nodes.
    4.1. All of our nodes aim to ensure this simple invariant, which is that the order in which rows are output from the node must be exactly the order that the rows were input to the node. This invariant can of course be broken during execution of the node, as long as it is repaired before finishing, as it is done when the

nodes are multithreaded. This is important once the processing has finished, and perhaps we may wish to visualize the data, this ensures that the image is correct.

5. The system must ensure all nodes have the correct colour and the correct node icon.
6. The system must ensure that each node has an informative usage guide explaining their capabilities and how to use them in the “node description” function in Knime.

Non-Functional Requirements :

1. The system must be developed such that Knime source code is separated from our java code.
2. The system must use Knime data structures sparingly.
3. The system should make use of multiple threads in individual nodes for concurrent programming.
4. The system must require every row to have a RowKey of the form “(x,y)” where x and y are integers which describe the spatial location of pixels in MSI.

5) NODE REQUIREMENTS

1) Reader Nodes

Here is some documentation on the reader nodes. There will be two main ways of reading information into KNIME data tables, spectrum by spectrum, and image by image. In order to parse the imzML and ibd files, we will be dependent on the imzMLConverter.jar provided by project supervisor Iain Styles.

1.1) imzML Reader (Row Spectrum)

This node is to read in an entire imzML dataset into a KNIME DataTable, such that each row contains a spectrum corresponding to a pixel in the image.

Functional Requirements:

1.1.a) The node must completely read imzML datasets into KNIME DataTables.

1.1.b) The node should have 0 input ports.

1.1.c) The node must have 2 output ports.

1.1.d) The node must output the m/z values for each spectrum in the DataTable output port 1, such that each spectrum is allocated one row in the DataTable.

1.1.e) The node must output the intensity values for each spectrum in the DataTable output port 2, such that each spectrum is allocated one row in the DataTable.

1.1.f) The node must output corresponding m/z and intensity pairs, such that the value output in row i, column j of output port 1 is the m/z value of the corresponding intensity found in output port 2’s row i and column j.

1.1.g) The node must have only one File Chooser, for the user to input the path to the .imzML file located.

1.1.h) The node must not have a File Chooser for the .ibd file.

1.1.i) The node must only accept the .imzML file as input, such that the .ibd file is in the same directory and has the same name as the .imzML file.

User input to the node:

1.1.i) The location of the imzML file to be read in the system.

1.2) imzML Reader (Column Spectrum)

This node is for reading a full imzML dataset into a KNIME DataTable such that each spectrum is allocated to a column in the outputs for a DataTable.

Functional Requirements:

1.2.a - 1.2.c) See 1.1.a - 1.1.c.

1.2.d) The node must output the m/z values for each spectrum in the DataTable output port 1, such that each spectrum is allocated one column in the DataTable.

1.2.e) The node must output the intensity values for each spectrum in the DataTable output port 2, such that each spectrum is allocated one column in the DataTable.

1.2.f - 1.2.i) See 1.1.f - 1.1.i.

User input to the node:

1.2.i) See 1.1.i.

1.3) imzML Reader Loop Start (Row Spectrum)

This node is to be used for reading in imzML datasets in chunks of one spectrum at a time, such that each spectrum is stored in a row in the KNIME DataTable, allowing spectra to be processed independently, and thus more memory efficient. It will do this be being a LoopStart node type in KNIME.

Functional Requirements:

1.3.a) The node must read in imzML datasets by spectrum.

1.3.b) The node must have 0 input ports.

1.3.c) The node must have 2 output ports.

1.3.d) The node should allow the user to specify how many spectra to output per chunk.

1.3.e) The node must output the m/z values for a given spectrum in a row in the DataTable in output port 1.

1.3.f) The node must output the intensity values for a given spectrum in a row in the DataTable in output port 2.

1.3.g) The node must pair the m/z values and the intensity values for a given spectrum. For example, output port 1 row i, column j should store the m/z value corresponding to the intensity value stored in port 2 row i, column j.

1.3.h) The node could have a view displaying meta information about the imzML dataset.

1.3.i - 1.3.k) See 1.1.g - 1.1.i.

User input to the node:

1.3.i) See 1.1.i.

1.4) imzML Reader Loop Start (Column Spectrum)

This node is to be used for reading in imzML datasets in chunks of one spectrum at a time, such that each spectrum is stored in a column in the KNIME DataTable, allowing spectra to be processed independently, and thus more memory efficient. It will do this be being a LoopStart node type in KNIME.

Functional Requirements:

1.4.a - 1.4.d) See 1.3.a - 1.3.d.

1.4.e) The node must output the m/z values for a given spectrum in a column in the DataTable in output port 1.

1.4.f) The node must output the intensity values for a given spectrum in a column in the DataTable in output port 2.

1.4.g - 1.3.j) See 1.3.g - 1.3.j.

User input to the node:

1.4.i) See 1.1.i

1.5) imzML Ion Image Reader

This node is to be used for reading in imzML datasets image by image. It loads in one image at a time, so that each image can be processed independently. This is a LoopStart node type with one out port.

Functional Requirements:

User input to the node:

1.5.i) See 1.1.i.

2) Manipulation Nodes

We are to implement nodes that can manipulate the DataTables provided which are read in by the imzML readers into KNIME.

2.1) Ion Image Region of Interest

This node is to be used for limiting the image dataset to a Region of Interest (ROI) in an imzML dataset. This can be done through a single pass through the images in the data.

Functional Requirements:

User input to the node:

2.1.i) Minimum x pixel value (integer), which must be greater than or equal to 1.

2.1.ii) Maximum x pixel value (integer), which must be greater than or equal to 1. If it is greater than the maximum size of the image, then we don't limit the max value at all.

2.1.iii) Minimum y pixel value (integer), which must be greater than or equal to 1.

2.1.iv) Maximum y pixel value (integer), which must be greater than or equal to 1. If it is greater than the maximum size of the image, then we don't limit the max value at all.

2.2) Limit Spectrum m/z

This node is to be used for limiting the minimum and maximum m/z values to be considered of an imzML dataset. This can be done through a single pass in the data.

Functional Requirements:

User input to the node:

2.5) Rebinner

This is to provide bins for the value of the m/z. Can be done through a single pass of the data. It can take unbinned datasets or binned datasets. It will create a new spectrum. It will have two input and two output ports.

Functional Requirements:

User input to the node:

2.4) Savitzky Golay Smoother

This node is to be used to smooth the data. This can be done through a single pass of the data. This node is to perform Savitzky-Golay smoothing on either the rows or columns.

Functional Requirements:

2.4.a) The node must have 2 input ports and 2 output ports.

2.4.b) The node must smooth data using a Savitzky-Golay filter.

2.4.c) The node must allow the smoothing of columns, as specified by the user in the dialog.

2.4.d) The node must allow the smoothing of rows, as specified by the user in the dialog.

2.4.e) The node must allow the user to specify the filter width to be used.

2.4.f) The node must output the exact table that was input to input port 0.

2.4.g) The node must output the smoothed data with the same DataTableSpec as input port 1 in output port 1.

2.5.h) The node could allow for the calculating of nth derivatives too.

User input to the node:

2.4.i) The system must be informed as to whether the smoothing is to be performed on the rows or the columns of the incoming DataTable.

2.4.ii) The system must be informed of the filter width to be used for the Savitzky-Golay filter.

2.5) BaseLine Correction

This node is to be used for correcting the baseline of the spectrum. This can be done through a single pass of the data.

Functional Requirements:

User input to the node:

2.7) Normalization Factor

This node takes an input of intensity, and then outputs a spectrum which is the spectra used to normalize the data, by one of the following normalization methods, as specified by the user in the dialog. This node will not actually normalize the data. We do this to avoid passing through the dataset more than twice, in the preprocessing phase. We then use this information to normalize the data in a pass of the spectra.

Methods of Normalization:

- Constant Shift (minus the mean spectra from every spectrum.)
- Standard Normal Variate (minus the mean and divide by the standard deviation)
- Median (minus the median spectra.)
- TIC (returns the median TIC)

Currently DONE :

- TIC
- Median
- Euclidean Norm

Functional Requirements:

User input to the node:

2.7.i) The normalization method chosen by the user.

2.8) Normalization (Loop Body)

These are normalization methods that can be done in a single pass of the data and can thus be part of the loop body of the preprocessing stage of the analysis.

Methods of Normalization:

- Scaling (dividing a spectra by its total energy).

Methods of Normalization

Functional Requirements:

User input to the node:

6) INTERFACE BETWEEN NODES

Node	Node Type	Has View?	Input ports	Output Ports
imzML Reader (Row Spectrum)	Source	No	n/a	0 : A DataTable where the rows are the m/z values of the spectra. 1 : A DataTable where the rows are the intensity values of the spectra.
imzML Reader (Col Spectrum)	Source	No	n/a	0 : A DataTable where the columns are the m/z values of the spectra. 1 : A DataTable where the columns are the intensity values of the spectra.
imzML Reader Loop Start(Row Spectrum)	LoopStart	No	n/a	0 : A DataTable where the rows are the m/z values of the spectra. 1 : A DataTable where the rows are the intensity values of the spectra.
imzML Reader Loop Start (Column Spectrum)	LoopStart	No	n/a	0 : A DataTable where the columns are the m/z values of the spectra. 1 : A DataTable where the columns are the intensity values of the spectra.
Savitzky-Golay Smoother	Manipulator	No	0 : A DataTable of the x-coordinates for the Savitzky-Golay Smoothing 1 : A DataTable of the y-coordinates for the Savitzky-Golay Smoothing	0 : Input table 0, exactly how it was at input into this node. 1 : Input table 1, but the data in this case has now been smoothed.

Rebinner	Manipulator	No	0 : the m/z of spectra DataTable to be rebinned 1 : the intensity of spectra DataTable to be rebinned.	0 : DataTable of m/z spectra values to which have been rebinned. 1 : DataTable of intensity values which have been rebinned.
Normalization	Manipulator	No	0 : the intensity values in a DataTable to be normalized.	0 : The original DataTable with normalized columns.

7) TEST PLAN

We follow two different testing strategies for the different parts of this project. In particular, we will unit test the core logic using JUnit. KNIME requires a running instance for their code to work, hence we cannot use junit for the KNIME nodes, instead we will follow a black box approach and create test KNIME workflows, and ensure valid outputs. The technologies used for testing will be JUnit 4, and the KNIME Testing Framework 3.5.2.v201802051355 (org.knime.features.testingapplication.feature.group) category found in the KNIME node repository, this must be installed as a KNIME extension, as it is not included in the base KNIME installation.

Next, we have tables for each node and its test workflow, we have deemed it appropriate to do this for the KNIME nodes, following the black box approach to testing, but not for the logic, as that is tested and documented in their own respective JUnit tests (one for each class), found in src.msk.util.test.junit.

In the tables below, we are often dealing with data in KNIME formats, thus we adopt the following notation :

- [1,2,3] denotes a row in a DataTable with 1 as the first element, 2 as the second element, 3 as the third element.
- [[1,2,3],[4,5,6]] denotes a DataTable with 2 rows, as described above.The first row in the datatable is the leftmost, the second is the one after, and so on.

6.1) Mass Spect In Knime (Category)

Index	Test	Expected	Actual	Result (Pass/Fail)
0	MSK category name in node repository	“Mass Spect In Knime”	“Mass Spect In Knime”	Pass
1	Subdirectories	6 subdirectories	6 subdirectories	Pass

		named “Reader (imzML)”, “Preprocessing”, “Views”, “Postprocessing”, “Utility”, “Deprecated”	named “Reader (imzML)”, “Preprocessing”, “Views”, “Postprocessing”, “Utility”, “Deprecated”	
2	Category icon	Main MSK icon	Main MSK Icon	Pass
3	Subdirectory icons	Orange folder icon for all subdirectories	Orange folder icon for all subdirectories	Pass

6.2) imzML Reader (Node)

Index	**Test**	**Expected**	**Actual**	**Result (Pass/Fail)**
0	Contained directory	Reader (imzML)	Reader (imzML)	Pass
1	Node name	“imzML Reader”	“imzML Reader”	Pass
2	Node icon	Main MSK icon	Main MSK icon	Pass
3	Node Colour	Orange (Source Node Colour)	Orange	Pass
3	Ports	0 input, 2 output	0 input, 2 output	Pass
4	Node Description	As in xml	As in xml	Pass
5	Has Dialog	True	True	Pass
6	Dialog display	One FileChooser to input the imzML File	One FileChooser to input the imzML file	Pass
7	Has View	True	True	Pass
8	The views JTable should be resizable	Resizeable	Resizeable	Pass

6.3) imzML Reader Loop Start (Node)

We abbreviate the following in the next tables in this subsection:

- d_1 = Example_Continuous.imzML as a test dataset.
- d_2 = Example_Processed.imzML as a test dataset.

- d_3 = Example_Continuous with no .ibd file in directory.
- N = the number of spectra per iteration.

Index	**Test**	**Expected**	**Actual**	**Result (Pass/Fail)**
0	Contained directory	Reader (imzML)	Reader (imzML)	Pass
1	Node name	“imzML Reader Loop Start”	“imzML Reader Loop Start”	Pass
2	Node icon	Main MSK icon	Main MSK icon	Pass
3	Ports	0 input, 2 output	0 input, 2 output	Pass
3	Node Colour	Orange (Source Node Colour)	Orange	Pass
4	Node Description	As in xml	As in xml	Pass
5	Has Dialog	True	True	Pass
6	Dialog FileChooser	Displays FileChooser to input imzML location	Displays FileChooser to input imzML	Pass
7	Dialog NumberEdit for number of spectra per loop	Appear below FileChooser	Below FileChooser	Pass
8	Has View	True	True	Pass
9	Row Key for pixel x,y	“(x,y)” OR “(x,y)#n”	“(x,y)” OR “(x,y)#n” depending on whether the unique row keys is chosen on Loop End node	Pass
9	Dataset d_1, N=1, number of output rows in both output ports	9	9	Pass
10	Dataset d_1, N=9, number of output rows in both output ports	9	9	Pass
11	Dataset d_1, N=15, number of output rows in both output ports	9	9	Pass

12	Dataset = “NotAFile.imzML”	Failure to execute, popup a frame with error message telling user that is not a file.	Execute fails, Popup appears	Pass
13	Dataset = “NotImzml.html”	Failure to execute, “Not ImzML”	Failure to execute, popup appears “Not imzML“	Pass

6.4) Normalizer (Node)

Index	**Test**	**Expected**	**Actual**	**Result (Pass/Fail)**
0	Contained directory	Preprocessing	Preprocessing	Pass
1	Node name	“Normalizer”	“Normalizer”	Pass
2	Node icon	Main MSK icon	Main MSK icon	Pass
3	Ports	1 input, 1 output	1 input, 1 output	Pass
3	Node Colour	Yellow (Manipulator Colour)	Yellow	Pass
4	Node Description	As in xml	As in xml	Pass
5	Has Dialog	True	True	Pass
6	Has View	False	False	Pass
7	Dialog has ButtonGroup	True	True	Pass
8	ButtonGroup has three options	“TIC”, “Median”, “Euclidean Norm”	“TIC”, “Median”, “Euclidean Norm”	Pass
	The following tests are all with respect to the TIC Normalization being chosen.			
9	Single zero row [0]	[0]	[0]	Pass
10	Multiple zero rows [[0,0,0],[0,0,0],[0,0,0]]	[[0,0,0],[0,0,0],[0,0,0]]	[[0,0,0],[0,0,0],[0,0,0]]	Pass

11	Single row [1,1,1,1]	[0.25, 0.25, 0.25, 0.25]	[0.25, 0.25, 0.25, 0.25]	Pass
12	Multiple non-trivial rows [[1, 1, 2],[1, 5, 4],[6, 2, 2]]	[[0.25, 0.25, 0.5],[0.1, 0.5, 0.4],[0.6, 0.2, 0.2]]	[[0.25, 0.25, 0.5],[0.1, 0.5, 0.4],[0.6, 0.2, 0.2]]	Pass
	The following tests are all with respect to the Median Normalization being chosen.			
13	Single empty row [0]	[0]	[0]	Pass
14	Multiple zero rows [[0,0,0],[0,0,0],[0,0,0]]	[[0,0,0],[0,0,0],[0,0,0]]	[[0,0,0],[0,0,0],[0,0,0]]	Pass
15	Single Row [1,1,1,1]	[1,1,1,1]	[1,1,1,1]	Pass
16	Multiple non-trivial odd rows [[1, 1, 2],[1, 5, 4],[6, 2, 2]]	[1,1,2],[0.25,1.25,1],[3,1,1]]]	[[1,1,2],[0.25,1.25,1],[3,1,1]]]	Pass
17	Multiple non-trivial even rows [[1,1,2,1],[1,2,3,4],[3,2,1,2]]	[[1,1,2,1],[0.4,0.8,1.2,1.8],[1.5,1,0.5,1]]	[[1,1,2,1],[0.4,0.8,1.2,1.8],[1.5,1,0.5,1]]	Pass
	The following tests are all with respect to the Euclidean Norm Normalization being chosen.			
18	Single empty row [0]	[0]	[0]	Pass
19	Multiple zero rows [[0,0,0],[0,0,0],[0,0,0]]	[[0,0,0],[0,0,0],[0,0,0]]	[[0,0,0],[0,0,0],[0,0,0]]	Pass
20	Single Row [1,1,1,1]	[0.5,0.5,0.5,0.5]	[0.5,0.5,0.5,0.5]	Pass
21	Multiple non-trivial even rows [[1,1,1,1],[1,2,3,4],[3,2,1,2]]	[[1,1,1,1],[0.4,0.8,1.2,1.8],[1.5,1,0.5,1]]	[[1,1,2,1],[0.4,0.8,1.2,1.8],[1.5,1,0.5,1]]	Pass

6.5) Limit m/z Range (Node)

Index	**Test**	**Expected**	**Actual**	**Result (Pass/Fail)**
0	Contained directory	Preprocessing	Preprocessing	Pass
1	Node name	“Limit m/z Range”	“Limit m/z Range”	Pass
2	Node icon	Main MSK icon	Main MSK icon	Pass
3	Node Colour	Yellow (Manipulator Colour)	Yellow	Pass
3	Ports	2 input, 2 output	2 input, 2 output	Pass
4	Node Description	As in xml	As in xml	Pass
5	Has Dialog	True	True	Pass
6	Has View	False	False	Pass
7	Min m/z = 0 Max m/z = 0 Input m/z = [0] Input intensity = [1]	Output m/z = [0] Output intensity = [1]	Output m/z = [0] Output intensity = [1]	Pass
8	Min m/z = 0 Max m/z = -1 Input m/z = [0] Input intensity = [1]	Output m/z = EMPTY Output intensity = EMPTY	Output m/z = EMPTY Output Intensity = EMPTY	Pass
9	Min m/z = 0 Max m/z = 4 Input m/z = [1,2,3,4] Input intensity = [10,11,12,13]	Output m/z = [1,2,3,4] Output intensity = [10,11,12,13]	Output m/z = [1,2,3,4] Output intensity = [10,11,12,13]	Pass
10	Min m/z = 0 Max m/z = 2 Input m/z = [[1,2,3,4],[1,2,3,4],[1,2,3,4]] Input intensity = [[10,11,12,13],[5,4,3,2],[1,-1,0,11]]	Output m/z = [[2,3],[2,3],[2,3]] Output intensity = [[11,12],[4,3],[-1,0]]	Output m/z = [[2,3],[2,3],[2,3]] Output intensity = [[11,12],[4,3],[-1,0]]	Pass

6.6) Baseline Subtraction (Node)

Index	**Test**	**Expected**	**Actual**	**Result (Pass/Fail)**
0	Contained directory	Preprocessing	Preprocessing	Pass
1	Node name	“Baseline Subtraction”	“Baseline Subtraction”	Pass
2	Node icon	Main MSK icon	Main MSK icon	Pass
3	Node Colour	Yellow (Manipulator Colour)	Yellow	Pass
3	Ports	1 input, 2 output	1 input, 2 output	Pass
4	Node Description	As in xml	As in xml	Pass
5	Has Dialog	True	True	Pass
6	Has View	False	False	Pass
	The following tests are all with Baseline Subtraction set to “Minimum”			
7	Size = 2 , input = [0]	Baseline = [0] Subtracted = [0]	Baseline = [0] Subtracted = [0]	Pass
8	Size = 2, input = [1,1,1]	Baseline = [1,1,1] Subtracted = [0,0,0]	Baseline = [1,1,1] Subtracted = [0,0,0]	Pass
9	Size = 2, input = [[1,2,3], [4,2,4]]	Baseline = [[1,1,1],[2,2,2]] Subtracted = [[0,1,2],[2,0,2]]	Baseline = [[1,1,1],[2,2,2]] Subtracted = [[0,1,2],[2,0,2]]	Pass
	The following tests are all with Baseline Subtraction set to “Minimum”			
10	Size = 2 , input = [0]	Baseline = [0] Subtracted = [0]	Baseline = [0] Subtracted = [0]	Pass
11	Size = 2, input = [1,1,1]	Baseline = [1,1,1] Subtracted = [0,0,0]	Baseline = [1,1,1] Subtracted = [0,0,0]	Pass
12	Size = 2, input =	Baseline =	Baseline =	Pass

	[[1,2,3], [4,2,4]]	[[1,1,1],[2,2,2]] Subtracted = [[0,1,2],[2,0,2]]	[[1,1,1],[2,2,2]] Subtracted = [[0,1,2],[2,0,2]]	

7) TYPICAL WORKFLOW

8) USAGE GUIDELINES

9) EXTENSIBILITY GUIDELINE