Pathomx Documentation

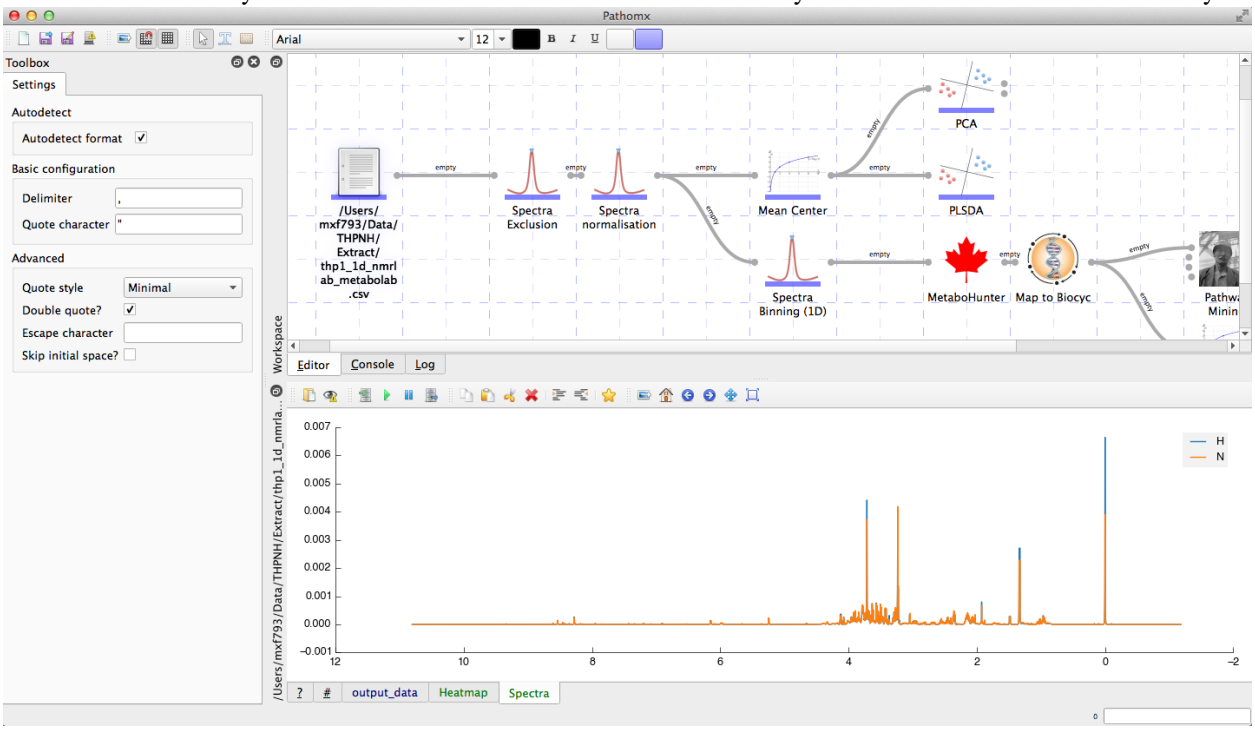
Release 3.0.0

Martin A. Fitzpatrick, Catherine M. McGrath, Stephen P. Young

August 06, 2015

Contents

Pathomx is a workflow-based tool for the analysis and visualisation of experimental data. Initially created as a tool for metabolomic data analysis is has been extended and can now be used for any scientific and non-scientific data analysis.



The software functions as a hybrid of workflow and script-based approaches to analysis. Using workflows it is possible to construct rapid, reproducible analysis constructs for experimental data. By combining this with custom inline scripting it is possible to perform any analysis imaginable. Workflows can be dynamically re-arranged to test different approaches and saved to track the development of your approach. Saved workflows can also be shared with other users or groups, allowing instant reproduction of results and methods. Tools can export images as publication-ready high resolution images in common formats.

This documentation contains useful information, demos and tips for the use of Pathomx by both users and developers.

CHAPTER 1

Users

1.1 Installation

We've created installation packages for Pathomx for both Windows and MacOS X. Follow the instructions below to install the software.

1.1.1 Windows (64bit)

Download the latest release as a Windows Installer (.exe). Double-click to start the installation process. Depending on security settings you may need to confirm the installation. Icons will be added to your Start menu and desktop. Launch as for any other application.

You may need to install a copy of the Visual C++ Redistributable Packages for Visual Studio 2013 if it isn't already installed on your system. Follow the above link, click *Download* and select *vcredist_x64.exe*. Download and install as for any other package.

- v3.0.2 (EXE installer) Windows Vista/7/8 (64bit)

To begin using Pathomx, take a look at *Getting Started*.

Previous releases are also available for download if required:

- v2.5.2 (MSI installer) Windows Vista/7/8 (64bit)
- v2.4.3 (MSI installer) Windows Vista/7/8 (64bit)
- v2.3.0 (MSI installer) Windows Vista/7/8 (64bit)

1.1.2 MacOS X

Download the latest release as a Mac Disk Image (.dmg). Double-click to open, and drag the Pathomx application into your Applications folder. Launch for any other app.

- v3.0.2 (Disk Image) MacOS X Mountain Lion

To begin using Pathomx, take a look at *Getting Started*.

Previous releases are also available for download if required:

- v2.5.2 (Disk Image) MacOS X Mountain Lion
- v2.4.0 (Disk Image) MacOS X Mountain Lion
- v2.3.0 (Disk Image) MacOS X Mountain Lion

1.1.3 Linux

Coming soon.

1.2 Getting Started

This is quick start-up guide for new users of Pathomx. In here you should find everything you need to know to start using Pathomx right away. Once you've been through the basics you might like to see some of the *Demos & Sample Workflows* to see what Pathomx is capable of.

Pathomx aims to offer a powerful, extensible analysis and processing platform while being as simple to use as possible to the casual user. It should be possible to pick up Pathomx, use the built-in - or bioinformatician provided - tools and perform a complete analysis in a matter of minutes. Saved workflows should be simple to use, reliable and reproducible. Outputs should be beautiful.

If Pathomx fails for you on any of those points, please do file a bug report and it'll be sorted out as soon as humanly possible.

1.2.1 First steps

Before you can start you'll need to install the software. There are a few different ways to install Pathomx but they make no difference to how you'll use it.

1.2.2 Nomenclature

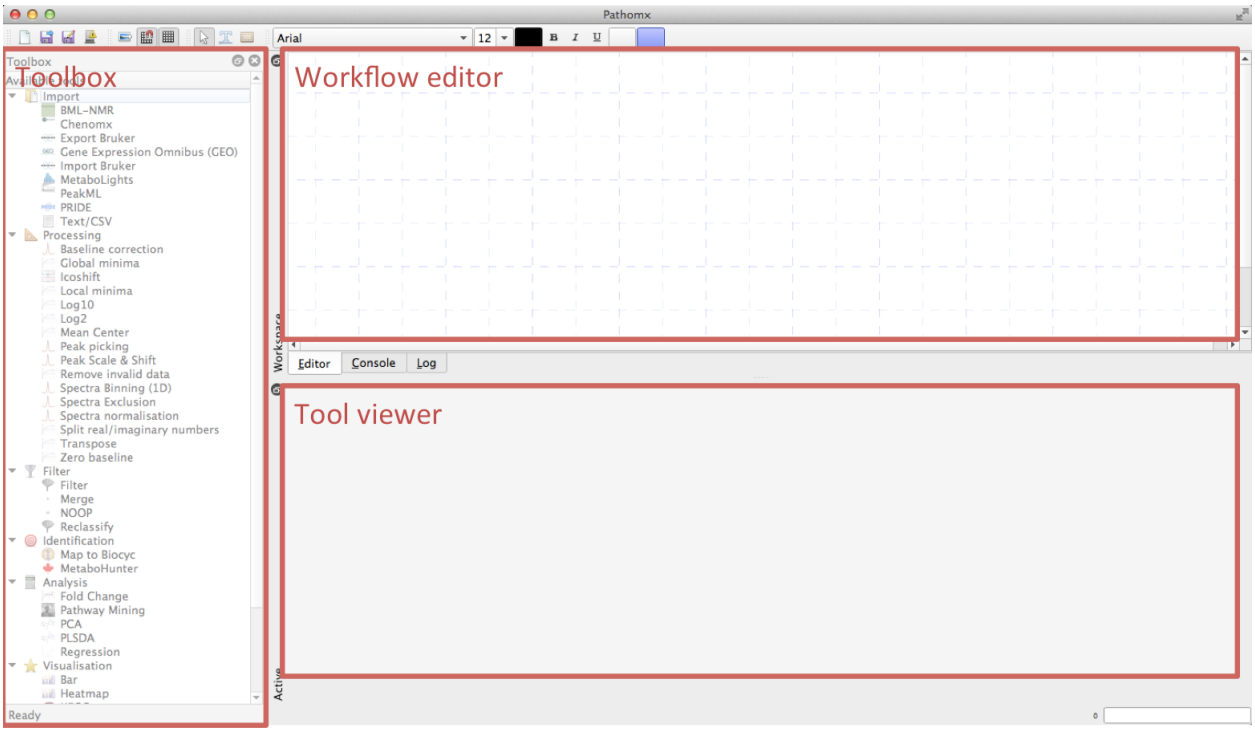
In Pathomx nomenclature *toolkits* provide *tools* with which you can construct *workflows*.

Your currently available tools are shown in the *Toolbox* within the application and can be dragged into the workspace to use. Once in the workflow tools can be dragged and rearranged as you like, the position of the tool has no effect on function.

Each tool has a number (0-infinity) of *ports* for *input* and *output*. Data is taken in via an input port, processed by the tool in some way, and passed out of the output port. The output of one tool can be connected to the input of another by *connectors* which can be created by dragging from the output to the input, represented by grey circles.

1.2.3 The interface

The Pathomx user interface (UI) is separated into 3 regions with specific purposes. These are dockable and re-arrangeable, but in their default configuration look like the following:



The *workflow editor* in the top right is where you arrange tools to construct workflows. Tools can be dragged-and-dropped from the *toolbox* then connected up. The are available in the workflow automatically extends to include all added tools and you can pan around the workflow as normal. If you find the workflow space too small you can un-dock it by clicking on the overlapping-window icon in the top left.

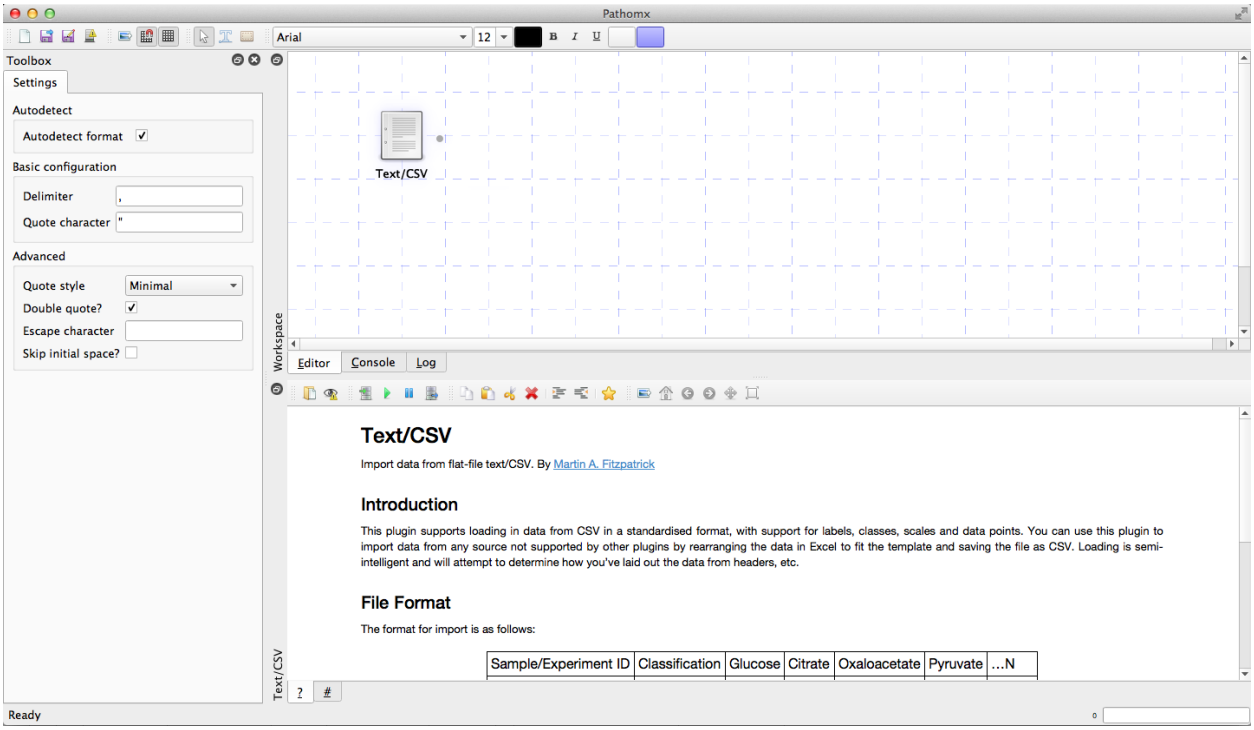
The *tool viewer* is a multi-purpose region that shows the info, code and current outputs for each tool presented in a tabbed interface. By default the tool information is displayed, but after running the tool will automatically show the first available output. Some outputs - such as figures - can also be displayed directly in the workflow editor.

When a tool is selected the *toolbox* will automatically change to show configuration options for that tool. In this way it is simple to rapidly reconfigure a processing workflow and see the resulting effects on the current and downstream tools.

1.2.4 Importing data

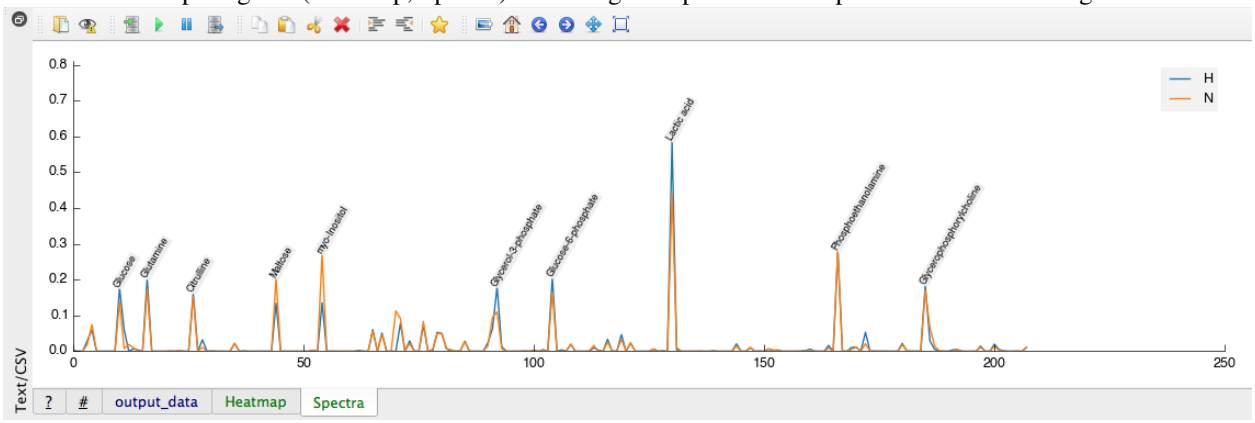
To demonstrate some key features of Pathomx we're going to perform a quick analysis using the standard toolkit and a demo data file. The downloadable file can be downloaded here.

Start up Pathomx and you'll be presented with an empty workflow editor. To get started we'll first add a tool that allows us to import this file: *Text/CSV*. Locate the tool in the toolbox and then drag and drop into the workflow (click-and-hold the tool, then move over the workflow and release). The new tool will be created in the location where you drop it. Next select the tool to activate it.



Selecting the tool will activate the configuration panel on the left where you can change tool settings. Any change to a setting will trigger the automatic re-calculation of the tools output. You can control this behaviour by using the *Pause* button on the tool run control toolbar. The *Play* button manually runs the current tool.

To load the data click the button next to the *filename* configuration box and an "Open file..." dialog box will appear. Locate the downloaded file and click OK. The Text/CSV tool will automatically run, loading the file and generating a set of default output figures (Heatmap, Spectra). Selecting the Spectra tool output tab will show the figure below:

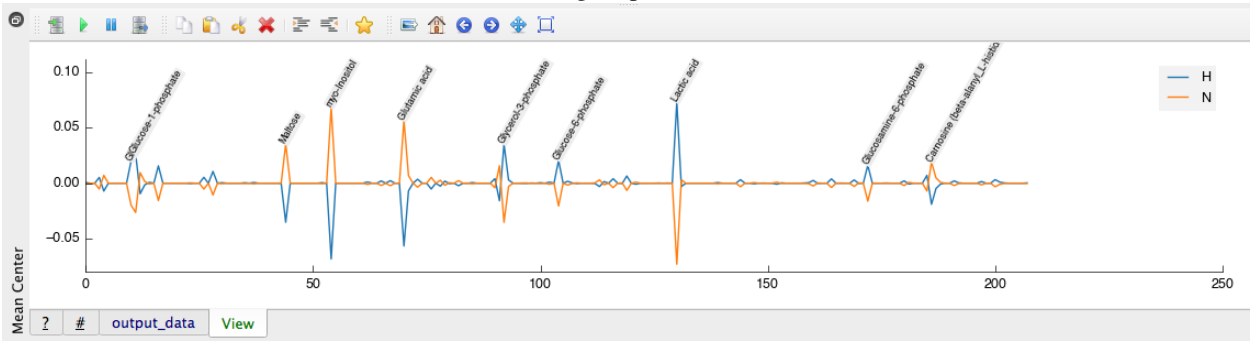


1.2.5 Processing

Performing further processing of the data is simply a case of adding more tools to the workflow. To return to the toolbox click any empty space in the workflow editor. Next, select the *Mean Center* tool and drag that into the workflow editor, somewhere to the right of the first tool. You will notice that the tools automatically connect, and the processing is automatically run (tool status bar turns blue). Any tools you add who's inputs are compatible with a

previous tool's outputs will automatically connect when added. This allows rapid construction of workflows.

Now select the Mean Center tool to show the following output:

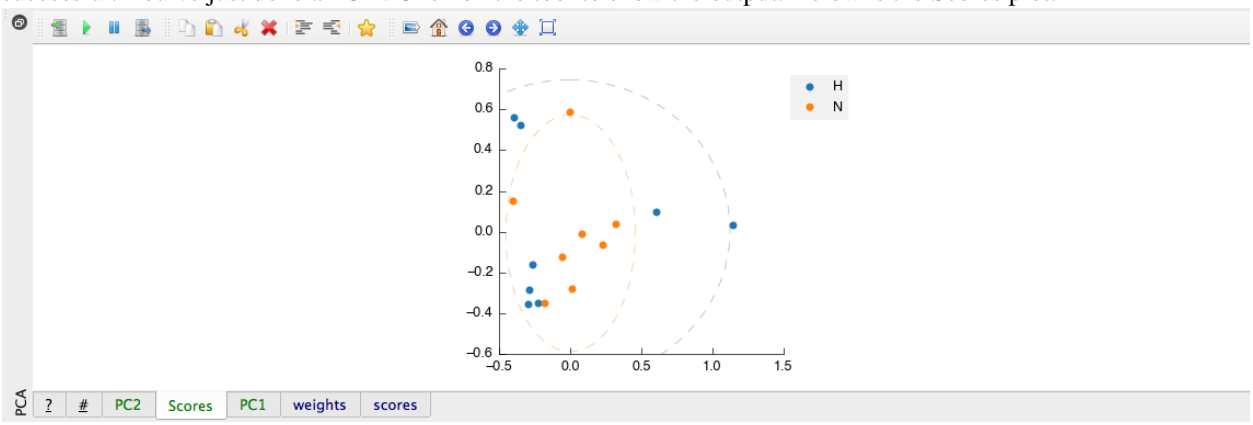


The imported data has been mean centered. Next we'll perform a quick multivariate analysis.

1.2.6 Analysis

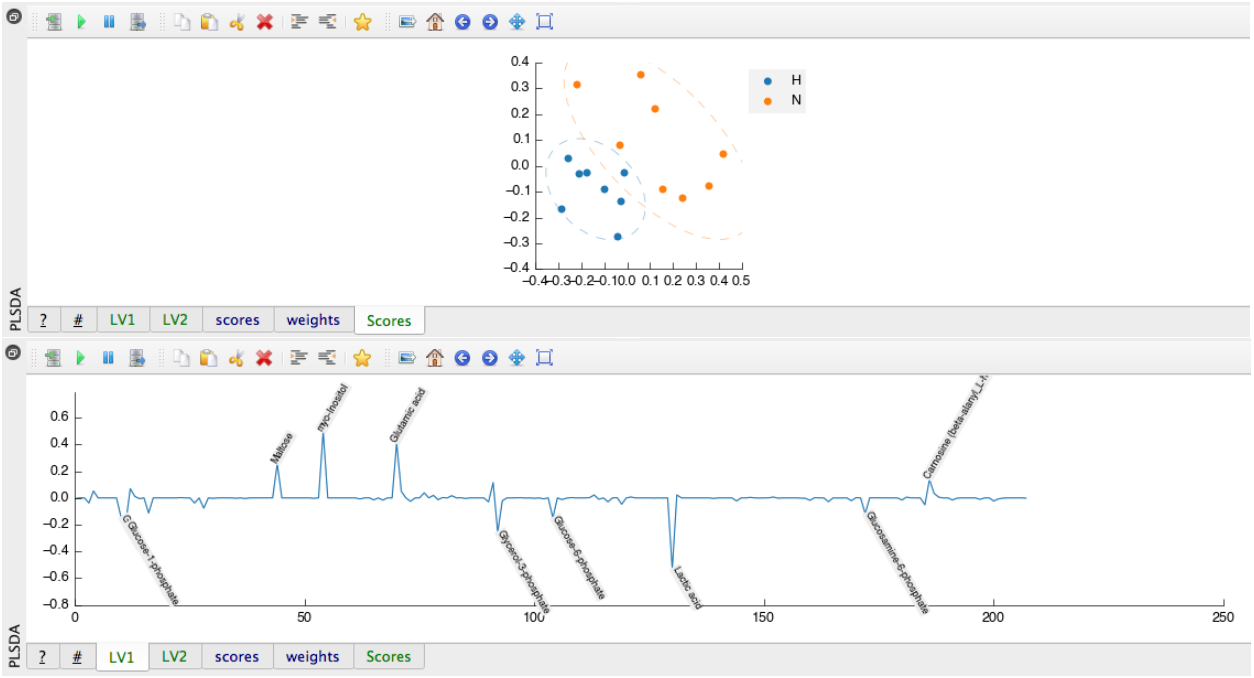
Multivariate analysis tools are provided in the default analysis toolbox provided with Pathomx: Principal Components Analysis (PCA) and Partial Least Squares Discriminant Analysis (PLS-DA). We'll quickly perform one of each to demonstrate how easy it is to do.

First, the PCA. Find the *PCA* tool in the Toolbox and drag and drop it into the workflow editor. Again you should see it automatically connect up with the previous (Mean Center) tool and turn blue to indicate that the analysis has been successful. You've just done a PCA! Click on the tool to show the output. Below is the Scores plot:



Next we'll perform the PLS-DA. Find the *PLS-DA* tool in the Toolbox and drag and drop it into the workflow editor, preferably below the PCA tool. Again it will automatically connect, but this time incorrectly to the output of the PCA. This is because the output of the PCA is a valid input for the PLS-DA, however on this occasion this is not what we want. So, to correct the connection simply drag the output from the Mean Center tool across to the input of the newly created PLS-DA tool.

The tool will recalculate, and you'll get the following outputs:

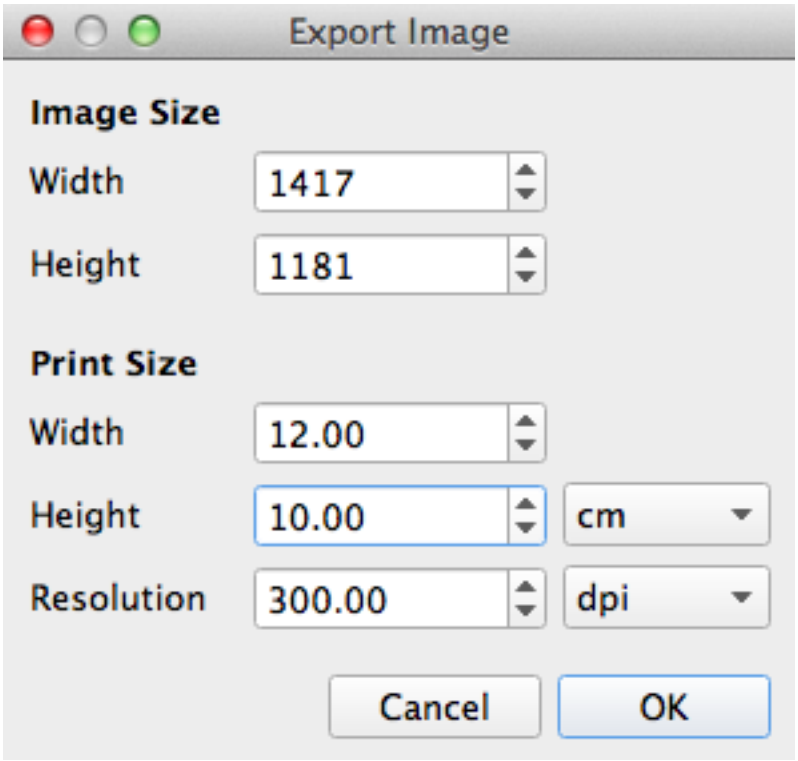


1.2.7 Figures

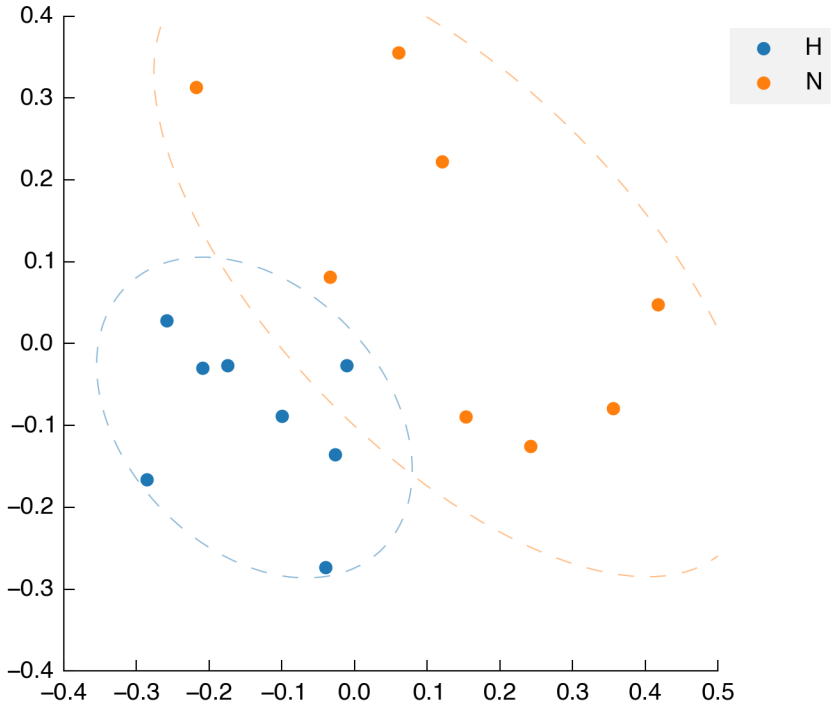
Figures generated by any Pathomx tool can be easily exported to high resolution formats (TIF) for publication. Selecting outputs (tabs) that support image export will activate the Figure toolbar. Select the PLS-DA Scores figure and then click on the image export icon (small picture with an arrow) will start the image export process.

The image export dialog will appear (below) that allows you change the settings for the exported image. For example, you can choose a higher/lower dpi setting and the dimensions of the final image. Resulting images will be automatically scaled to fit your chosen settings.

For the Scores plot the suggested size to export is 12x10cm and 300dpi for clarity.



Next an File Save dialog will be shown where you can choose the location, filename and file format. If you select TIF you will get a high-resolution image output at the specified dpi. If you've done everything correctly, it should look

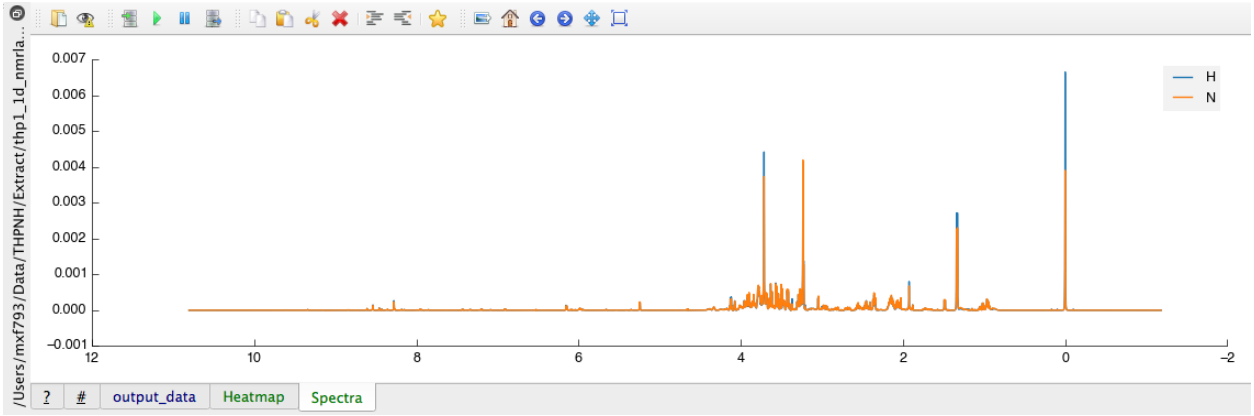


a lot like this: Colours and line styles can be managed for the entire workspace through the *Appearance > Line & Marker Styles* tool available via the main toolbar. Note that colours are applied based on experimental class groups, meaning that you can set a colour once and it will be used throughout for every output.

1.2.8 Re-using a workflow

While this has all been very nice, the real power of workflow analysis comes from the ability to re-use and re-apply the same series of steps to new data. There is a second dataset to download here that can be used to try this out.

To perform the analysis simply open up the Text/CSV tool you added first and select the new dataset via the configuration panel. The loaded data will look like this:



The analysis will run and the new figures will be generated. You can explore them by clicking through the tools in turn.

1.2.9 Next steps

This was a quick introduction to the use of Pathomx for analysis. To see more of what is possible have a look through some of the *Demos & Sample Workflows*.

1.3 Demos & Sample Workflows

This page lists a number of demo workflows that you can use to get acquainted with using Pathomx for analysis. Follow the links for both the completed workflow and a step-by-step guide to it's construction and then use.

If you have suggestions for new workflows or additions to the list open an feature request via Github.

1.3.1 1D Bruker NMR Analysis

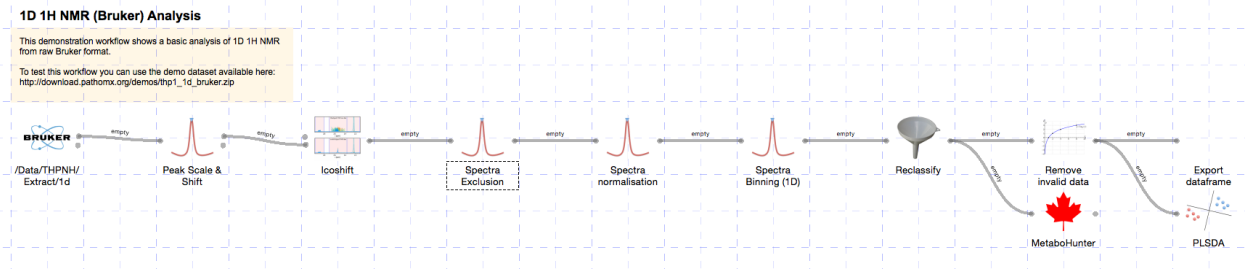
Analysis of 1D Bruker NMR data files including alignment, Icoshift segmental alignment, exclusion of regions, normalisation (PQN) and spectral binning. Data are subsequently assigned experimental classes, then annotated using the MetaboHunter online NMR peak identification service, analysed using a simple PLS-DA and also exported as a standard CSV format datafile for subsequent analysis.

You can download the completed workflow or follow the steps below to recreate it yourself. This workflow is also distributed with the latest versions of Pathomx and can be found within the software via *Help* > *Demos*.

The data used in this demonstration was derived from the culture of THP-1 macrophage cell line under hypoxic conditions for 24hrs. Metabolites were extracted using a methanol-chloroform protocol. The class groups used here represent (N)ormoxia and (H)ypoxia respectively.

Background

1D NMR is commonly used in metabolomics as a relatively quick and inexpensive method for profiling compounds in biological fluids. Processing of 1D NMR data involves a number of commonly applied steps. While the parameters applied to different NMR datasets may need to be altered, the series and order of applied steps tends to stay the same. This demo workflow provides a good standard base for customising your own analysis workflow and this tutorial will cover the process of extending the workflow further. Finally, a quick multivariate analysis will be performed.



To test the workflow as it's built you'll need to download the demo dataset and sample classification files. Unzip the dataset before use (the Bruker import tool requires a folder to open). The sample classification file is in CSV format and maps the NMR sample numbers to a specific class group.

Constructing the workflow

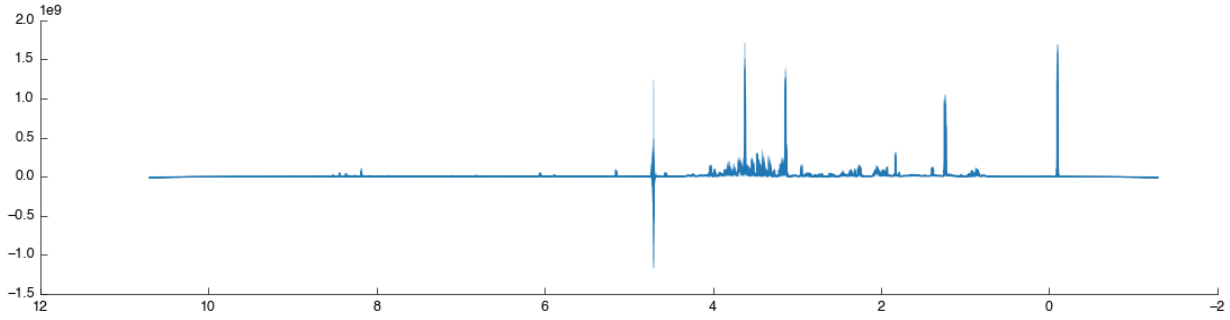
The workflow will be constructed step-by-step using the default toolkit supplied with Pathomx and a sample set of outputs shown along the way. If you find anything difficult to follow, let us know.

Importing data

Start up Pathomx and find the Bruker Import tool in the Toolbox panel on the left (the icon is the Bruker atomic logo). Drag and drop it into the workflow editor to create a new instance of the tool. Select it (turning it blue) to activate it and get access to the configuration panel.

The Bruker import tool supports a large number of configuration options. These are described in the help documentation with the tool (*?* tab) so won't be covered here. For this demo we can use the default values as-is. Simply click the open folder button and browse to the folder containing your spectra. If you're using the downloaded demo dataset unzip it first, then select the resulting folder.

The tool will now run, importing each spectra in turn and performing an automatic phase correction algorithm on the result. With the default settings a number of other functions will be performed including spectra reversal, deletion of the imaginary component, removal of the digital filter and zero filling. The resulting imported spectra are shown below:

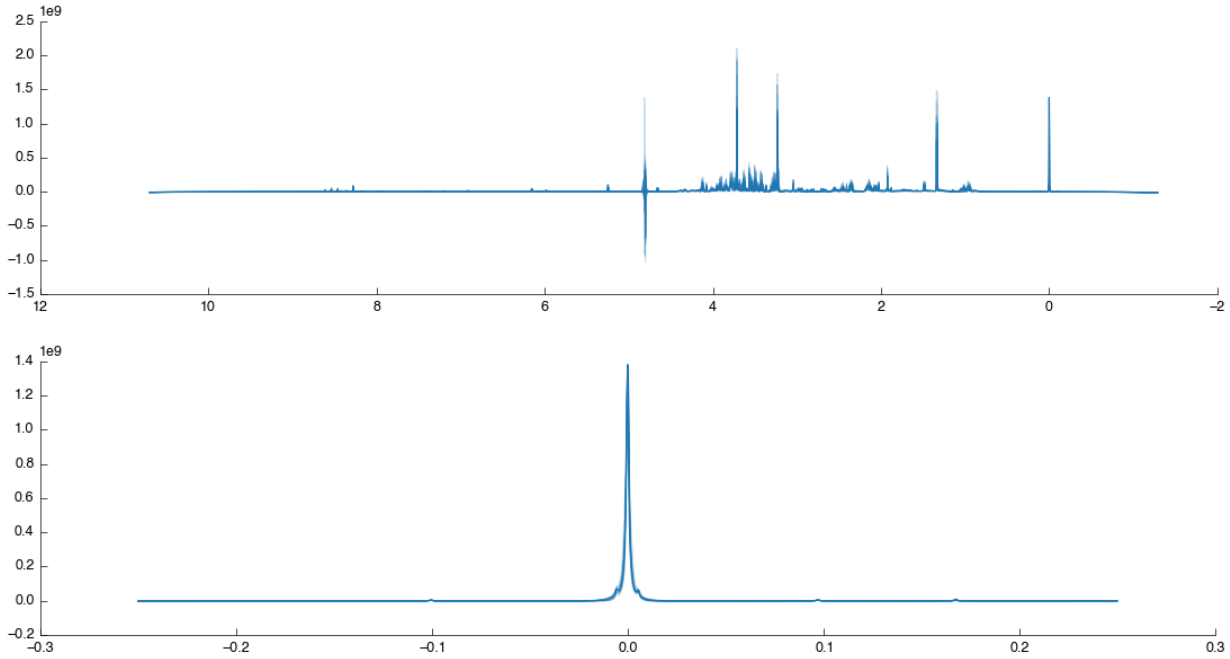


Spectral processing

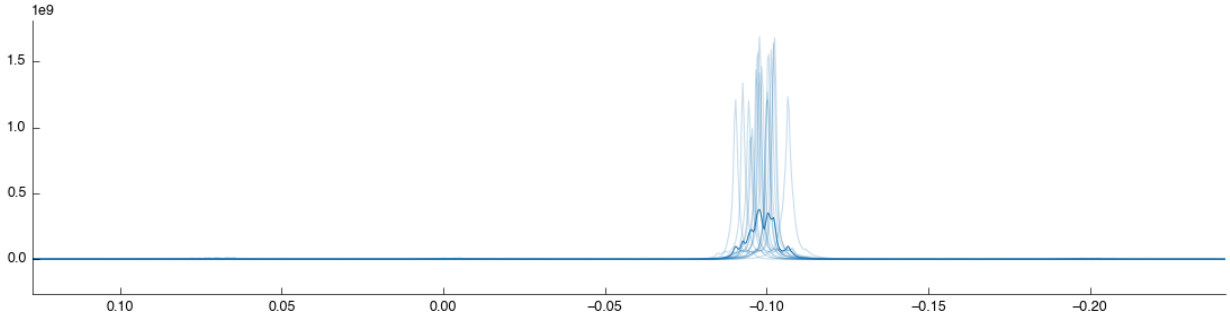
Next we'll perform a series of common NMR processing steps using the standard toolkit. As the data has already been loaded, these tools will autoconnect as they are added.

The first step is to scale and shift spectra by a reference peak (e.g. TMSP). TMSP has been added to all samples at the same concentration and so variation reflects differences in acquisition. By scaling and shifting on this peak we can eliminate that variation from our data.

Start by dragging the *Peak Scale & Shift* tool into the workflow editor to the right of *Bruker Import*. It will connect and automatically calculate using the default settings of alignment on the TMSP peak - exactly what we want for this dataset. The resulting data is output via *output_data* and two figures are generated *View* and *Region*. The former is the full spectra view (after scaling) while the second is a close-up of the region which we've shiftd and scaled on. The two outputs are shown below:

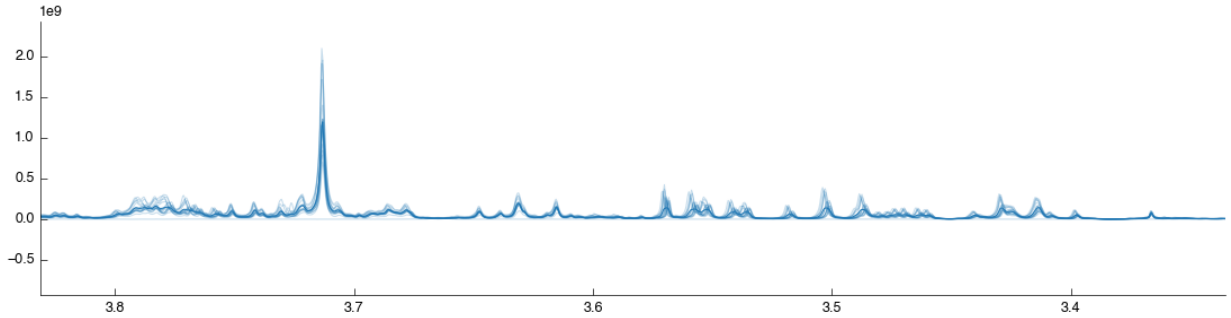


As shown the region around the TMSP peak has been well-shifted and scaled so that the various spectra overlap well. For comparison the original unshifted, unscaled TMSP region is shown below (as taken from the previous *Bruker Import* tool):

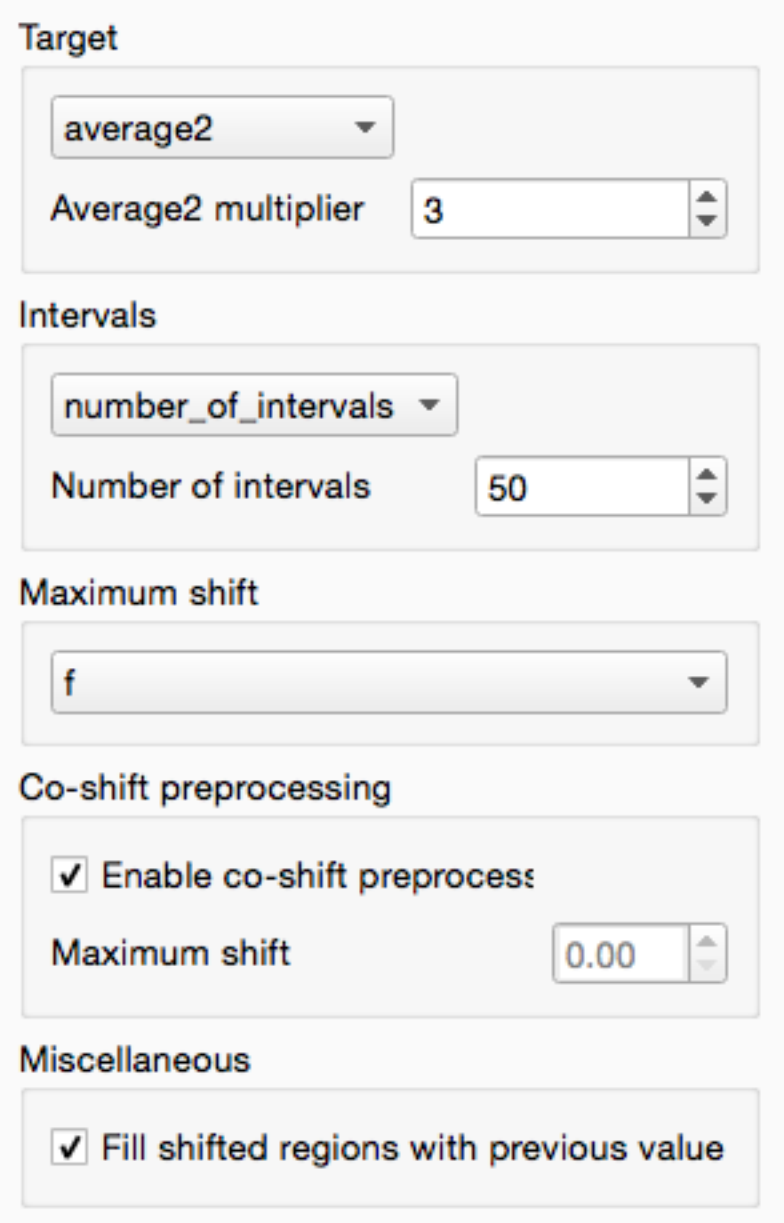


Although the spectra are now well aligned on the TMSP peak (accounting for shifting introduced by the spectrometer) the alignment may not be as good elsewhere. Variation in the pH of the analysed samples results in a shift in the compound peaks (± H+). This can have implications for later analysis as peak misalignment may make a compound appear to be reduced in a sample. As we're interested in real biological concentration differences we must try and limit

this effect as far as possible. In this demo we'll make use of the *Icoshift* tool, which uses a Python implementation of the *i*coshift segmental correlation shifting algorithm. Drag the tool into the workflow editor to autoconnect it. It will take a short while to run, and produce the following figures:



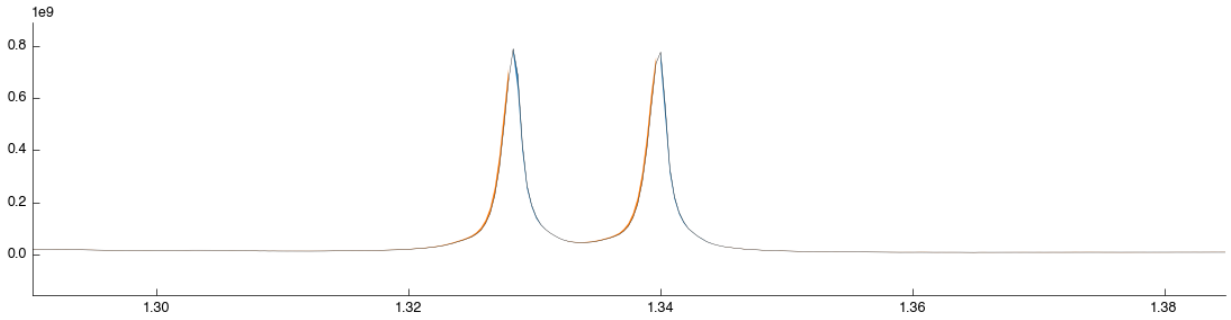
The default settings are not the most effective: by default it shifts the whole spectra towards a mean of the spectra. As we've already shifted the spectra this has limited effect. To optimise the shifting change the settings as follows:



These settings do the following:

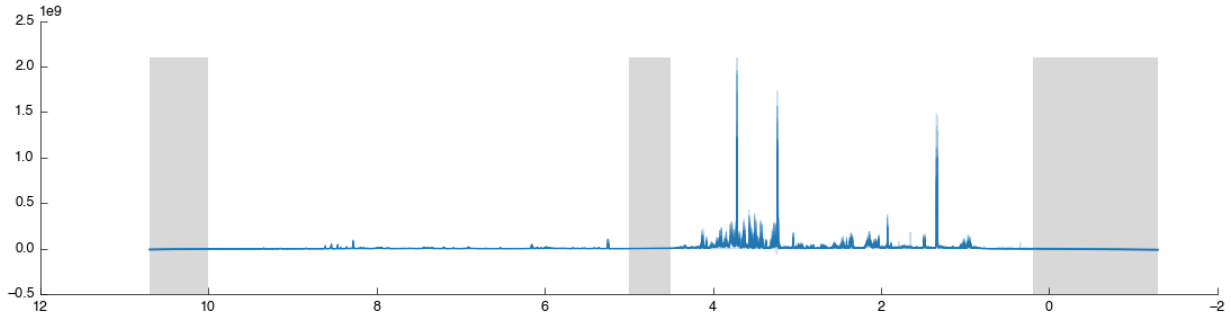
- Turn on co-shift preprocessing, a quick pre-shift to ensure alignment is maximised to begin with
- Change the target to average2, with weight of 3, for the most-average spectra to create a 'better' target
- Segmental shifting using 50 segments per spectra, allowing regions of the spectra to move independently to maximally align local peaks

The *Icoshift* tool also outputs a useful *Difference* plot that shows where the changes have occurred in the shifted spectra. You can use this to zoom in inspect whether the shift is beneficial to the clarity of the spectra.

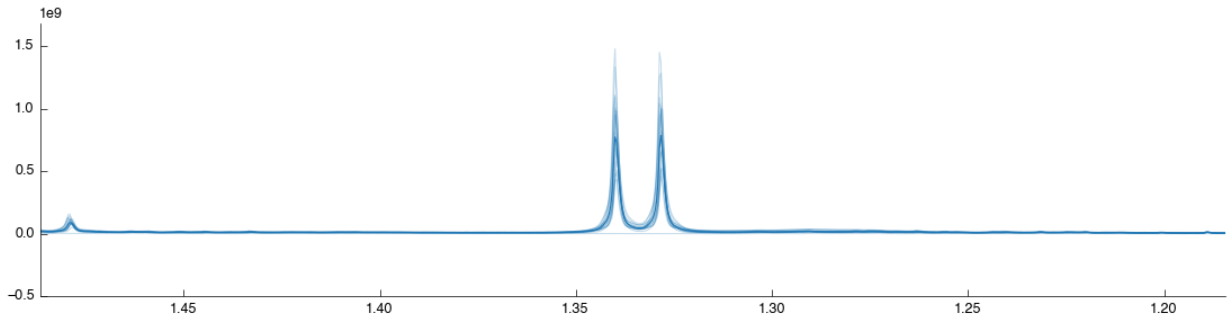


At this point we now have a optimally aligned set of spectra. The next step in NMR processing is to remove the uninformative parts of the spectra. These include the water region (noise) the TMSP reference peak (no longer required) and the far 10 ppm + region of the spectra (for 1H metabolomic NMR this contains no useful information).

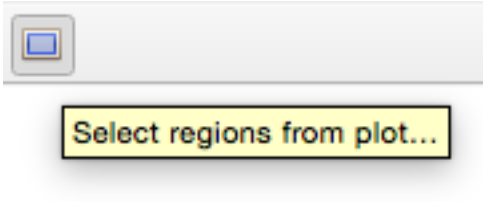
We can do all this in a single step using the *Spectra Exclusion* tool: simply drag and drop it into the workflow editor. This tool comes with the standard 1H regions pre-defined but you can add and remove any regions you wish. After the tool has completed processing you will see the following figure:



The excluded regions are shown in grey with flat-lines where data is missing. Let's now add another exclusion region just to see how it is done: we will remove lactate since it is easy to find. Zoom in on the lactate doublet peak at 1.30-1.35:

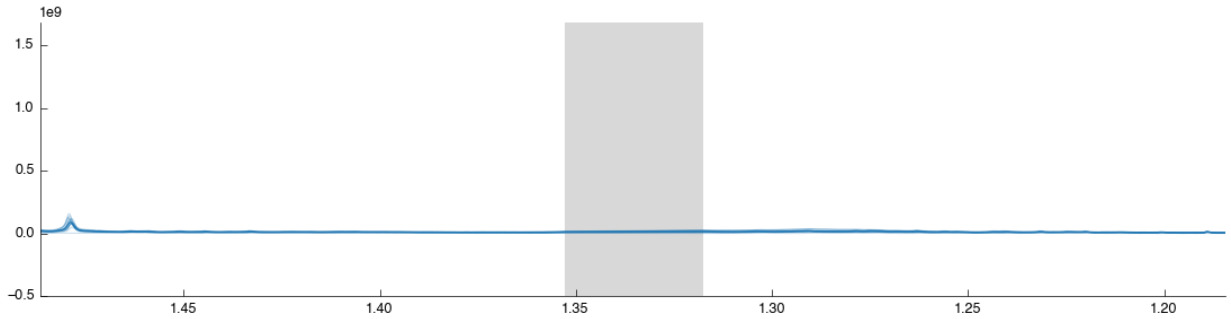


On the figure toolbar find the select region icon and click it to change to *Region* mode:

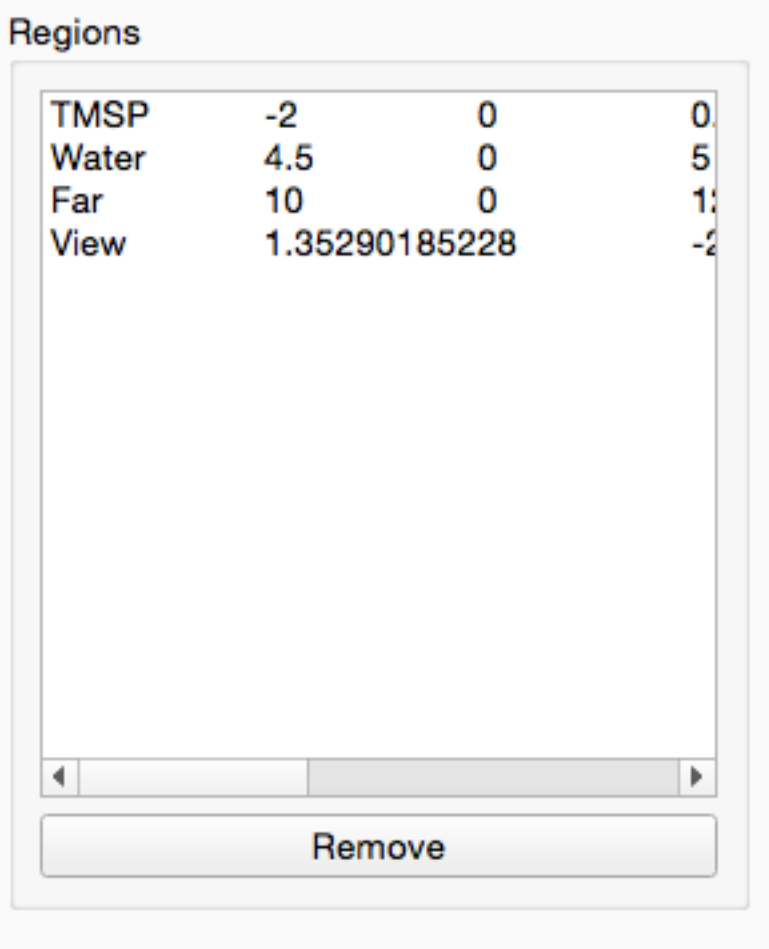


Drag a box over the lactate peak. Note that it doesn't matter if you contain the peak within the box, just that you cover the region on the X axis (this is a 1d plot). After you release the mouse the tool will auto-run with the new

configuration and you should see the following:



The region you created has also been added to the exclusion list in the configuration panel:



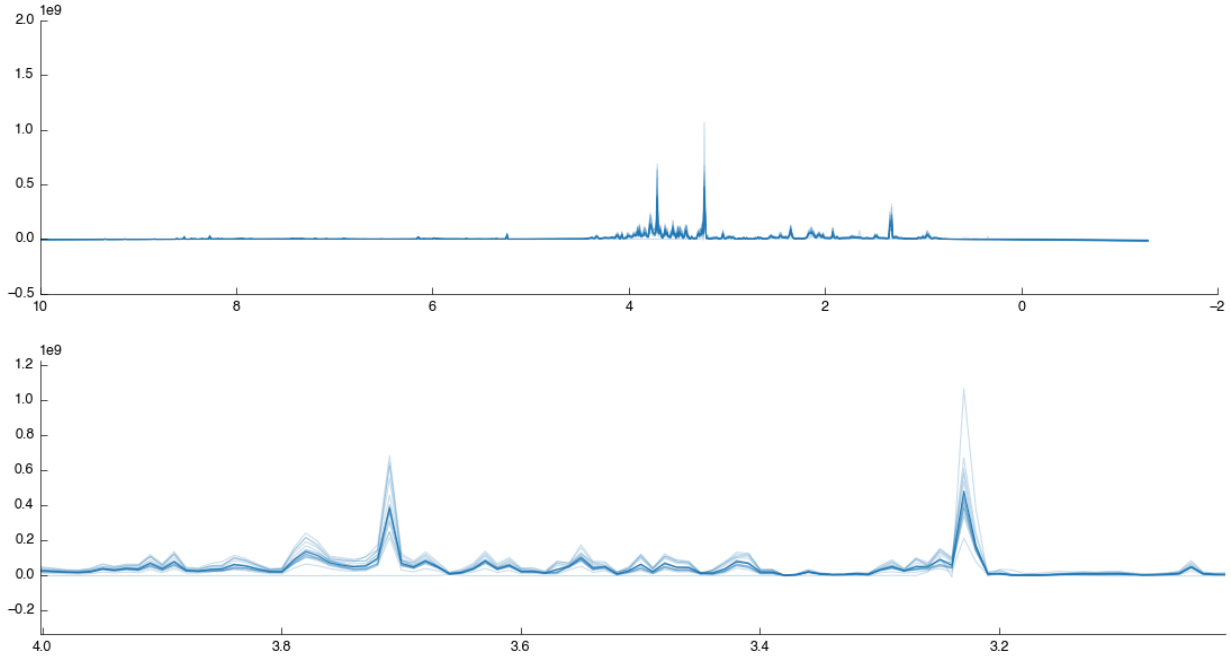
Select your region from the list and click "Remove" to remove it from the list and re-add the lactate region to the output spectra. Remember you can always re-use this tool later on to remove regions from the spectra that are causing issues in your downstream analysis.

Spectral binning

We've now got a set of spectra well-aligned and with all the useless data thrown away. However despite our best efforts there still exist tiny variations in peak positions. *Binning* (also known as *bucketing*) is the simplest method for the removal of this variation from tspectra. It splits the spectra up into multiple regions of equal size and then takes

the sum of the data within that region. This is a loss of resolution, but one that aids further downstream analysis by simplifying comparison between spectra.

In Pathomx this can be achieved using the *Spectral binning* tool from the toolkit. Just drag and drop it to the workflow editor to add it. It will run and produce the following figure: if you zoom in you'll see that the spectra is now more pointy.



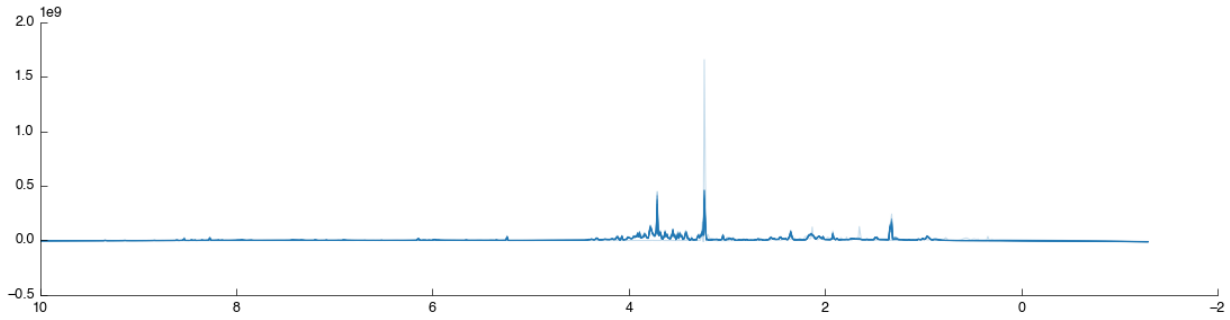
Spectral normalisation

We now have the spectra processed for analysis. However there is another (optional) step that can be used to help ensure variation observed in the spectra is indeed indicative of biology and not a side effect of the source material. One of the major sources of variation is dilution of the source material: particularly relevant in urinary metabolomics for example.

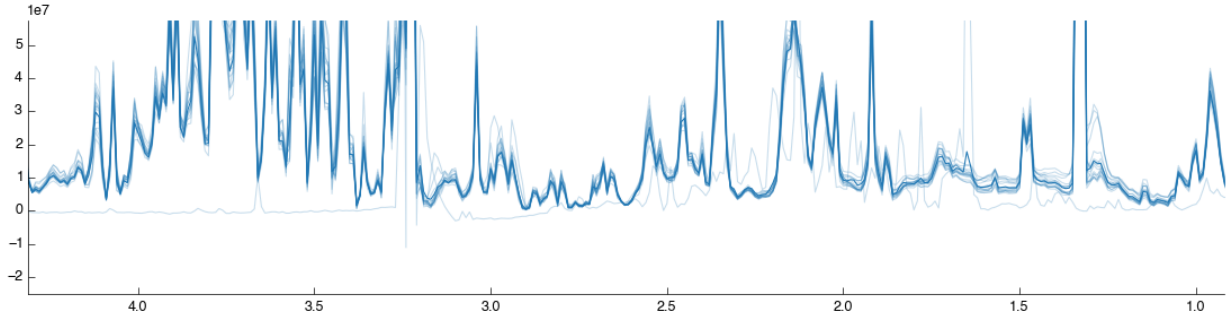
There are two common approaches for spectral normalisation used in metabolomics: Total Spectral Area (TSA) and Probabilistic Quotient Normalisation (PQN). Both function on the similar principal that most of the spectra will remain the same between samples in an experiment. TSA scales to a constant area under the curve (AUC) and is effective of urinary metabolomics assuming that the variation is small: a single large peak (contaminant) in a spectra will reduce all other peaks in the spectra and may incorrectly be interpreted as a reduction. PQN is a further improvement which uses TSA as a pre-step but then scales spectra to match their medians. This is less susceptible to the contaminant peak effect but relies on well-aligned spectra.

Our source data is from methanol-chloroform extracts from cell culture where cell number variations are a possibility. Here we'll use PQN to attempt to compensate (feel free to explore the analysis without this correction).

Drag and drop the *Spectra normalisation* tool into the workflow editor and it will automatically run. The default algorithm is PQN and will produce the following figure:



If you look closely you may notice that one of the spectra doesn't look right:



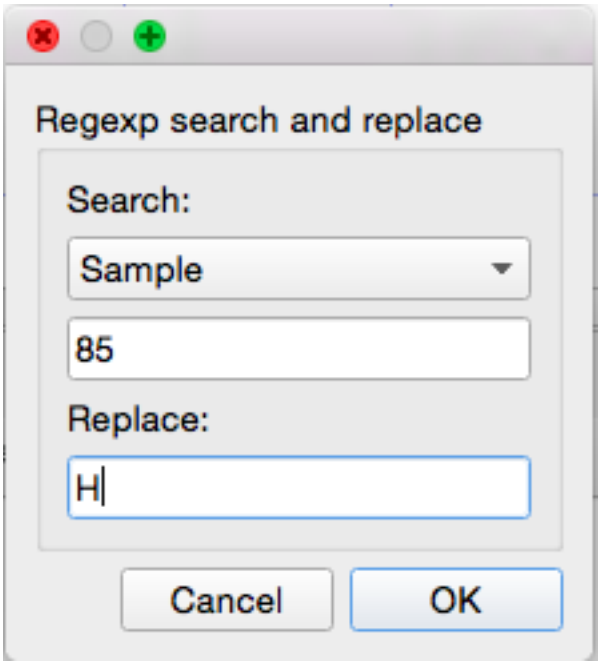
We'll look at how to filter spectra in a few minutes. For now, lets continue with the analysis.

Sample classification

The plot shows data for all the samples together with the mean (shown as a thicker line) as dataset doesn't currently contain any information on sample classifications. Let's add them now. Drag a *Reclassify* tool into the workflow editor. It will automatically take data from the *Spectral normalisation* tool.

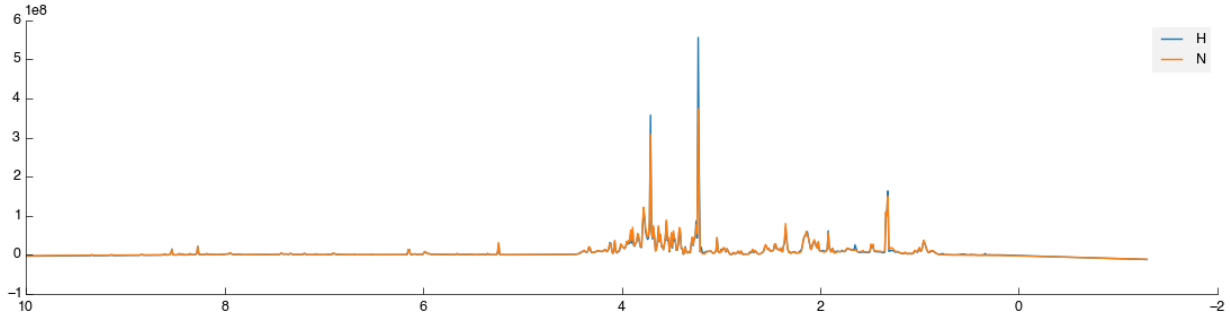
If you select the Reclassify tool and select the View output you will see exactly what you saw in the BML-NMR tool. That is because we haven't set up any reclassifications. You can do this in two ways: manual, or automatic from a CSV file import. We'll do the first one manually, then give up and do it quickly.

Select the Reclassify tool you just created. In the configuration panel on the left select *Add* to get the reclassification box. Select 'Sample' from the drop-down list (this means we're matching against the Sample number in the current data) and then enter *85* in the input box. Under Replace enter *H* (this is the value we'll replace sample 85's class with). After you click OK the assignment will be added with the reclassification table and the tool will recalculate.

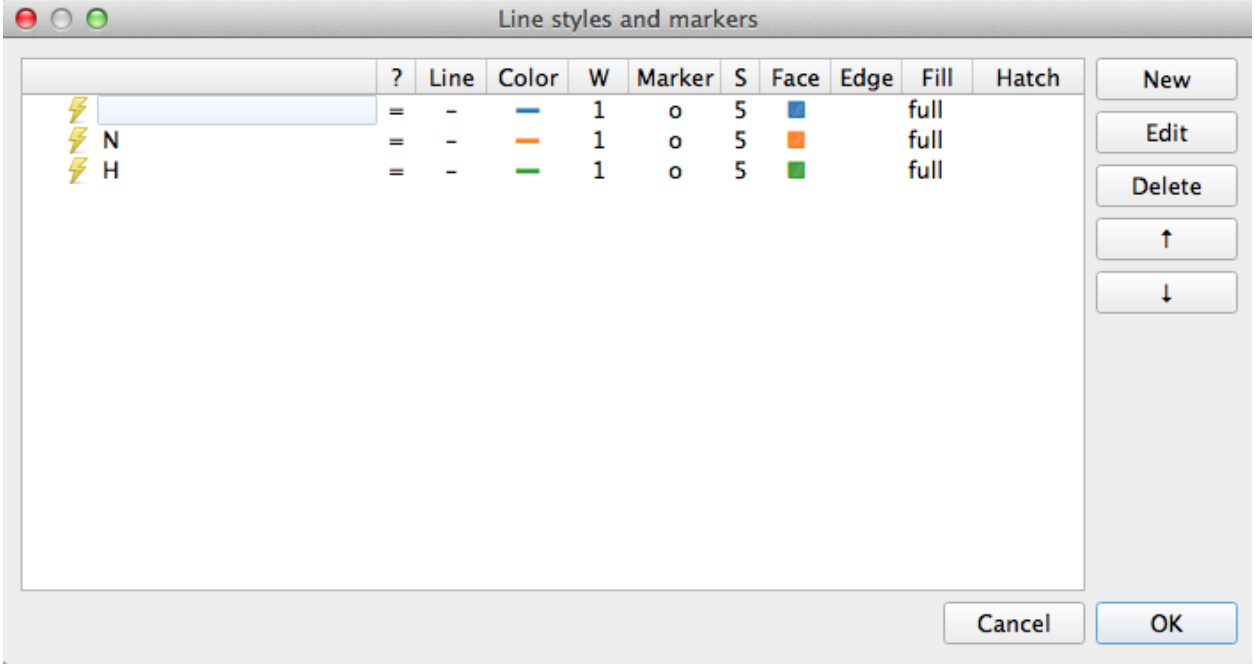


Select the *View* output and you will now see two lines: orange for the H class group and blue for the remaining unclassified samples.

That's not a huge amount of fun, so a quick way to get sample class matches is provided. To use this activate the Reclassify tool then in the configuration panel click the Open File icon (bottom right, next to Add). Select the *1d_classifications.csv* file you downloaded earlier and open it. You will be presented with a drop-down box to select the field on which to match, again choose 'Sample'. The full set of class assignments will be loaded and samples assigned properly. If you check the view again you'll get two clearly marked groups like the image below:



Except it isn't quite. Because we matched a single sample to begin with Pathomx needed a colour to identify the 'No class' group and took the first available (blue). So instead of the above figure, you've probably got one in green and orange. To fix this in the main application window select *Appearance* > *Line & Marker Styles*. You'll see this:



This dialog is the central control for the appearance of class groups in figures throughout Pathomx. Any change to the colours assigned here determines how they show up in every figure. Select the row for *N* and clicking Edit. For the Line setting click the colour button and then choose something obnoxious like pink. Save the settings by clicking OK, reselect the Reclassify tool and click the green *play* button on the control bar to re-run it. Your *N* line should now be pink.

Enough fun. Go back to *Appearance* > *Line & Marker Styles* and delete all the rows in the panel. Save it and return to your tool, hitting run once more. Now you should have the data visualisation displaying as shown.

Metabolite Identification

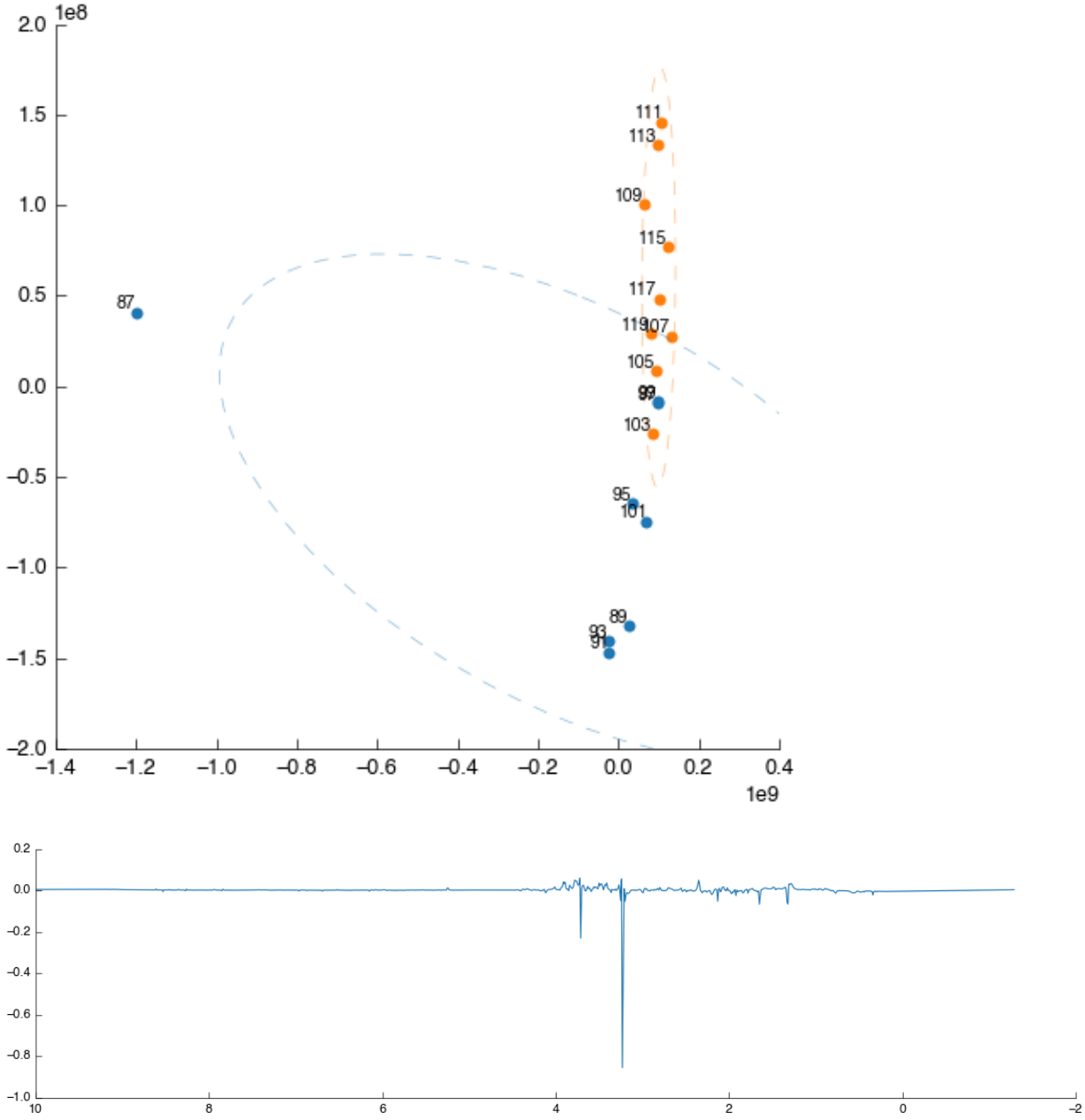
Metabolite identification from 1D NMR is difficult. The gold standard for matching is via manual identification against known compounds. Software packages such as Chenomx come a close second but are costly. Thankfully there are a number of free online matching services which, while not offering the same levels of accuracy, are sometimes *good enough* for a first-look investigation. Pathomx includes an interface to one such service: MetaboHunter.

The service can be accessed simply by dragging and dropping the *MetaboHunter* tool into the workflow editor. Note that this tool is paused by default (to avoid unneccessary requests to the server) and so you must run it manually. Either right click and select "Run" or select the tool and click the green play icon on the toolbar.

One the run is complete you can see the HMDB annotations by clicking on the *output_data* (not currently shown in the plot: coming soon). These values and annotations will persist through subsequent analysis and can be exported for use elsewhere. To do that now simply drag and drop a *Export dataframe* tool into the workflow editor. Select it and click the "Save..." icon to choose the target file.

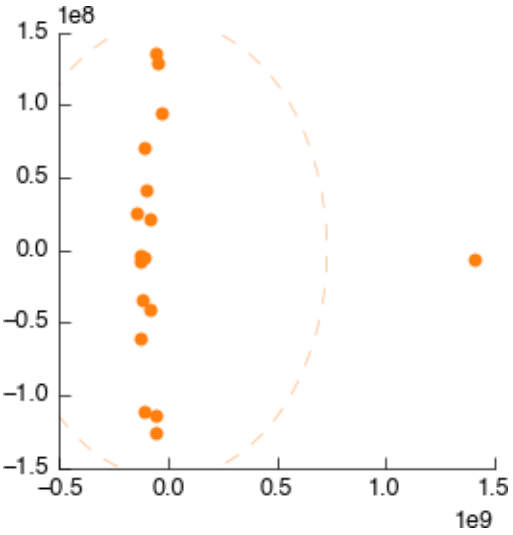
Multivariate analysis

Next we'll perform a quick multivariate analysis of our data using PLS-DA. Drag and drop the *PLS-DA* tool from the toolbox into the workflow editor. It will auto-connect to the MetaboHunter output but that is fine. Let it run and you'll get the following figures:

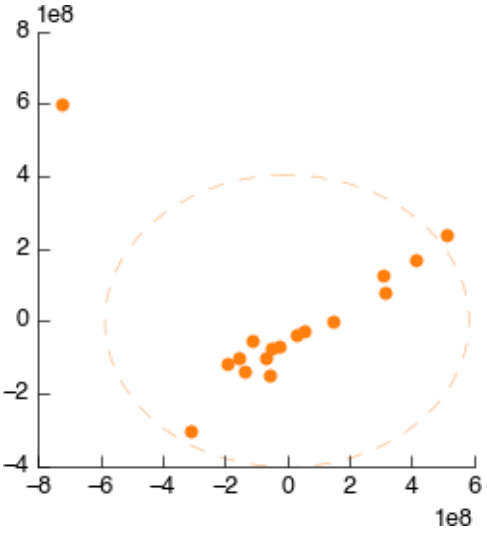


Something is wrong: one of the points (87) is way out to one side. What are the chances that this outlier is the same spectra that we saw 'looking odd' before? We can filter this sample out by number (hint: use the *Filter* app and filter by sample number) but we're smarter than that. First lets use PCA to find the source of weirdness in the data.

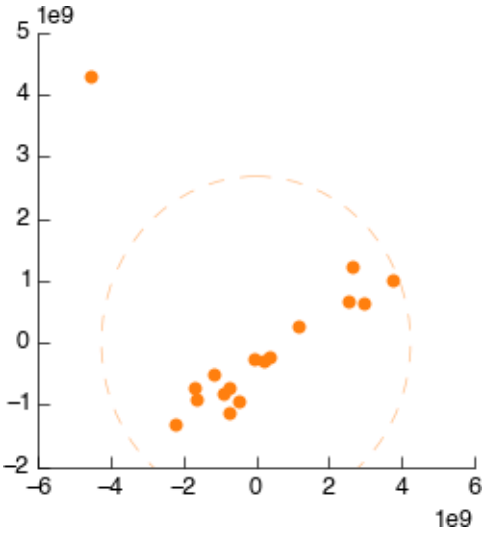
Drag and drop a *PCA* tool into the workspace. It will automatically connect to the output of MetaboHunter again, but reconnect it to the output of *Spectra normalisation*. Still looks weird.



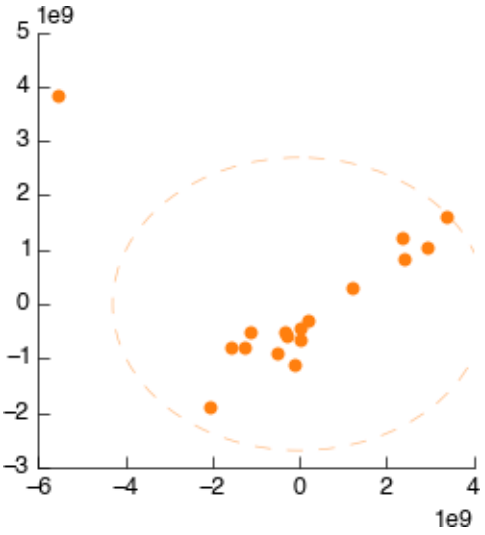
Connect it to *Spectral binning*. Still looks weird.



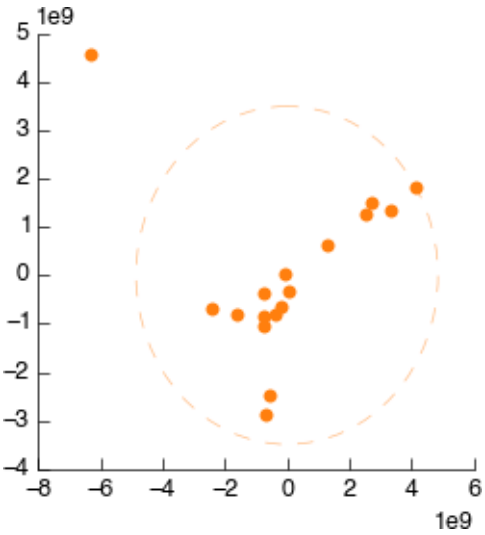
Connect it to *Spectral exclusion*. Still looks weird.



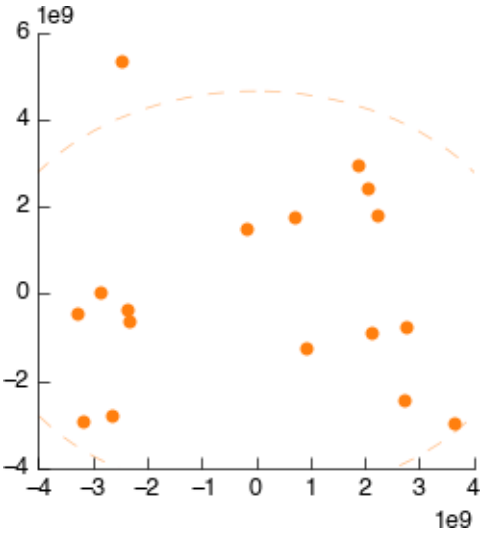
Connect it to *Icoshift*. Still looks weird.



Connect it to *Peak Scale & Shift*. Still looks weird.



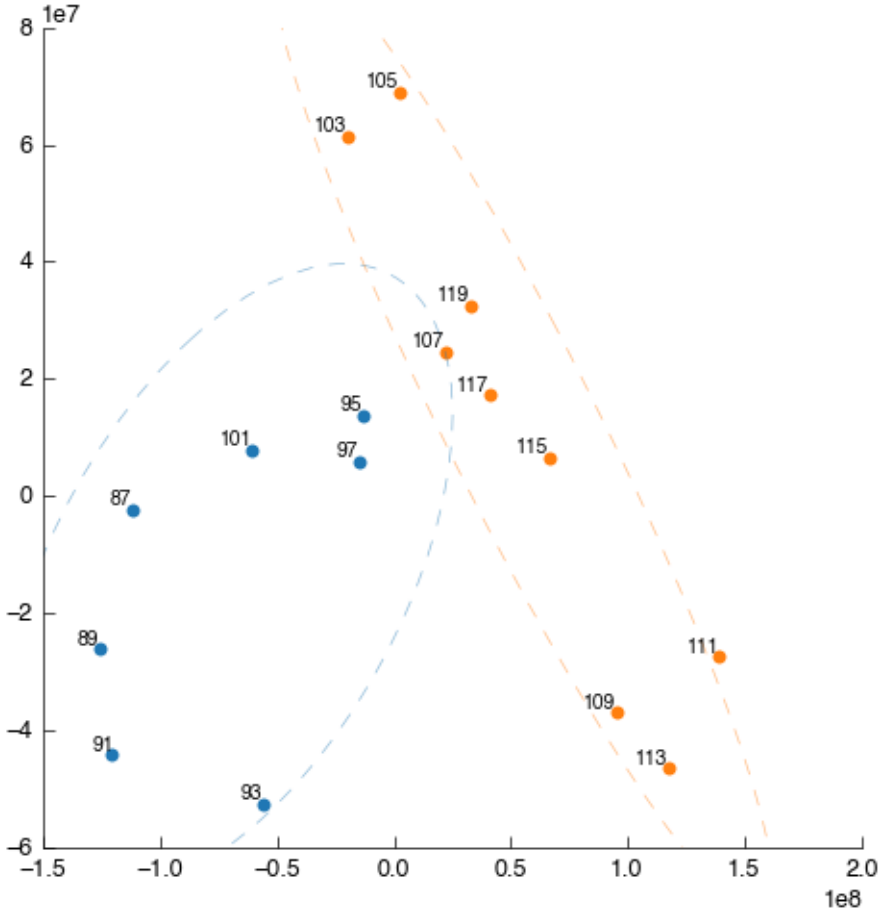
Connect it to *Bruker Import*. Still looks weird.



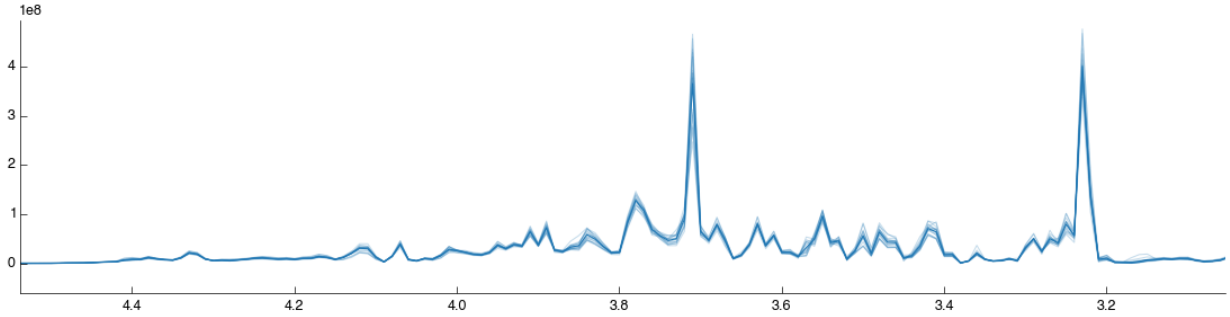
So, we've walked all the way back up our analysis and determined that the source of the weird spectra was - the spectra itself. We want to remove this data from the dataset as soon as possible to ensure it doesn't have strange effects on spectral alignment, scaling, etc. downstream. So we'll get rid of it right at the beginning. As described we could remove this sample by number, but instead we're doing to use a feature of the PCA tool to exclude dodgy samples dynamically.

On the PCA plot you'll notice a ellipse around the samples (or sample groups when classes are added). This line indicates the 95% confidence line: the line in which the model predict 95% (2sd) of samples should fall. Lets use this to automatically filter our samples. In the PCA tool select 'Filter data by covariance (2sd)'.

This will take a little while to complete, but once done you can drag the output *filtered_data* into the input of the *Peak Scale & Shift* tool. The downstream analysis will re-run automatically and update, with the dodgy sample excluded. This is our new PLS-DA:



Looking at the processed spectra (post-normalisation) we can see it is also cleaner:



That concludes this demo of 1D Bruker analysis with Pathomx. If you found anything confusing, hard to follow (or impossible) let us know.

Things to try out

If you're feeling adventurous there are a few things you can experiment with the workflow -

- Export the MetaboHunter mapped data to a CSV format file *hint: use Export dataframe*

1.3.2 1D Bruker NMR Analysis with Re-Export to Bruker format

Analysis of 1D Bruker NMR data files, including shifting, normalisation (PQN) and binning before re-exporting as Bruker format 1D NMR files using the standard Pathomx toolkit. These files can be reloaded into any normal NMR analysis software e.g. Chenomx.

You can download the completed workflow or follow the steps below to recreate it yourself. This workflow is also distributed with the latest versions of Pathomx and can be found within the software via *Help* > *Demos*.

Coming Soon

This demo walkthrough is coming soon.

1.3.3 2D-JRES NMR (Bruker) Analysis via BML-NMR

Analysis of 2D J-Resolved NMR spectra (in Bruker format) previously processed using the BML-NMR service. The resulting processed data zip-file is loaded and assigned experimental classifications. Metabolites are matched by name using the internal database and then analysed using a PLS-DA. The log2 fold change is calculated for each metabolite (replacing zero values with a local minima). The resulting dataset is visualised on GPML/WikiPathways metabolic pathways. The data is additionally analysed using a pathway mining algorithm and visualised using the MetaboViz pathway visualisation tool.

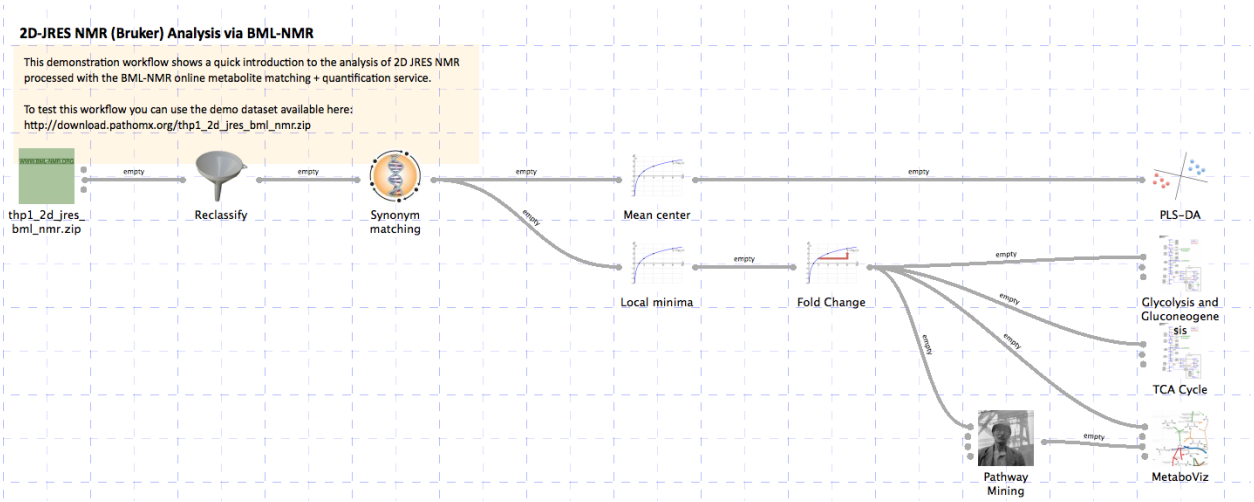
You can download the completed workflow or follow the steps below to recreate it yourself. This workflow is also distributed with the latest versions of Pathomx and can be found within the software via *Help* > *Demos*.

The data used in this demonstration was derived from the culture of THP-1 macrophage cell line under hypoxic conditions for 24hrs. Metabolites were extracted using a methanol-chloroform protocol. The class groups used here represent (N)ormoxia and (H)ypoxia respectively.

Background

The Birmingham Metabolite Library (BML-NMR) provided by the University of Birmingham also hosts an automated NMR spectra fitting service designed for 2D JRES NMR spectra. These low-resolution spectra have the advantage of being relatively quick to acquire while offering the resonant peak-separation found in full 2D spectra. Quantification and identification accuracy is in the 70-80% (dependent on source material) and may be sufficient for quick analyses.

This workflow takes the *.zip* output of the BML-NMR identification service and processes it to perform multivariate analyses, visualisation on WikiPathways and pathway-analysis-generated automated pathway visualisation. The overview of the pathway is as follows:



To test the workflow as it's built you'll need to download the demo dataset and sample classification files. You don't need to unzip the dataset, it is the exact same format that comes out of the BML-NMR service and Pathomx can handle it as-is. The sample classification file is in CSV format and simply maps the NMR sample numbers to a specific class group.

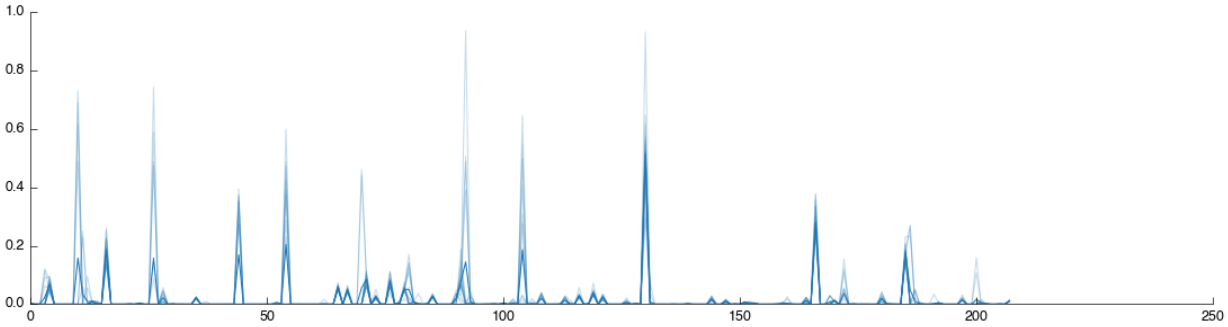
Constructing the workflow

The workflow will be constructed step-by-step using the default toolkit supplied with Pathomx and a sample set of outputs shown along the way. If you find anything difficult to follow, let us know.

Importing data

Start up Pathomx and find the BML-NMR tool in the Toolbox panel on the left (the icon is a green square). Drag and drop it into the workflow editor to create a new instance of the tool. Select it (turning it blue) to activate it and get access to the configuration panel. Here click the open file button and browse to the downloaded demo data file.

The tool will now run, extracting the data from the zip file and processing it for use in Pathomx. A number of outputs will also be generated including 3 data tables and 3 figures for the Raw, TSA-transformed and PQN-transformed datasets from the file. If you click on the PQN figure tab you will get a visualisation of the data you have just loaded.



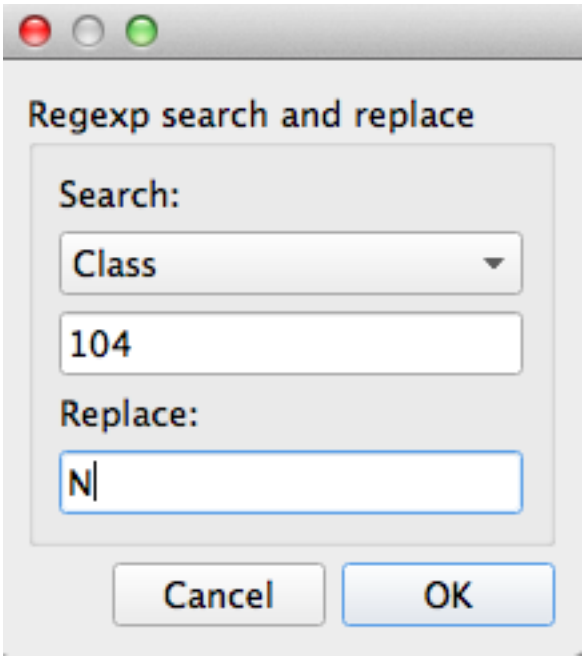
Sample classification

The plot shows data for all the samples together with the mean (shown as a thicker line). The dataset doesn't currently contain any information on sample classifications, so we'll add them now. Drag a *Reclassify* tool into the workflow editor. It will automatically take data from the *Raw* output of the BML-NMR tool but we want the *PQN* output. So

correct this by dragging from the PQN output to the input for Reclassify (the previous connection will automatically disconnect).

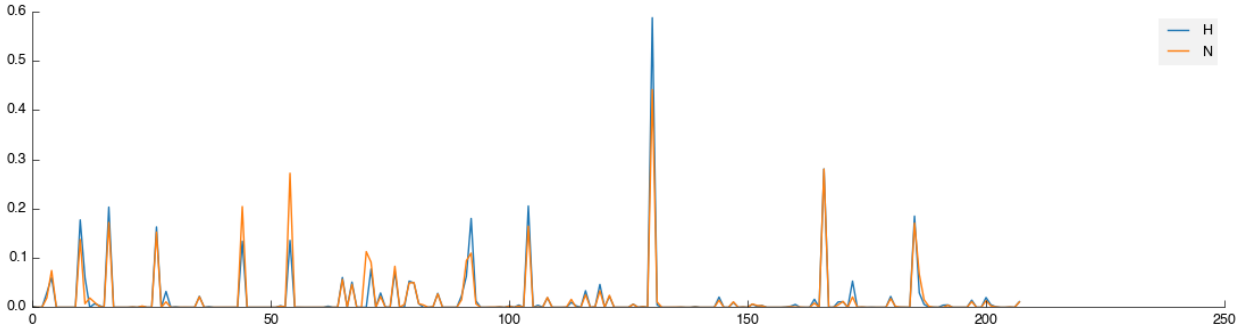
If you select the Reclassify tool and select the View output you will see exactly what you saw in the BML-NMR tool. That is because we haven't set up any reclassifications. You can do this in two ways: manual, or automatic from a CSV file import. We'll do the first one manually, then give up and do it quickly.

Select the Reclassify tool you just created. In the configuration panel on the left select *Add* to get the reclassification box. Select 'Sample' from the drop-down list (this means we're matching against the Sample number in the current data) and then enter *104* in the input box. Under Replace enter *N* (this is the value we'll replace sample 86's class with). After you click OK the assignment will be added with the reclassification table and the tool will recalculate.

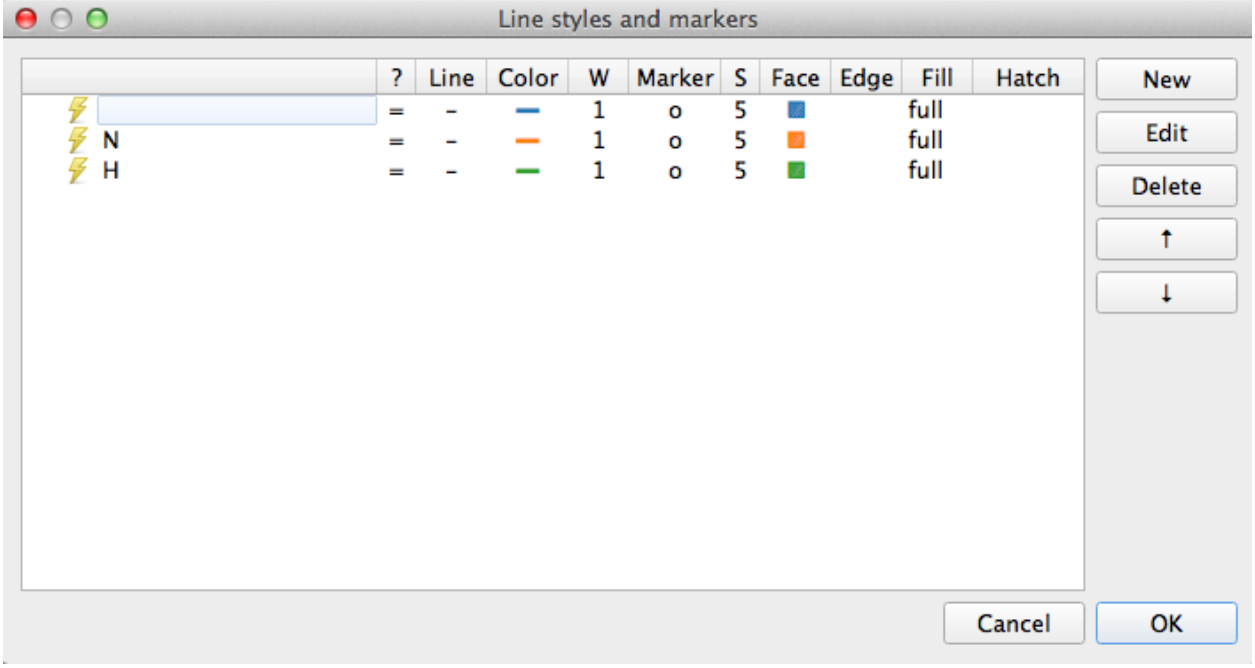


Select the *View* output and you will now see two lines: orange for the H class group and blue for the remaining unclassified samples.

That's not a huge amount of fun, so a quick way to get sample class matches is provided. To use this activate the Reclassify tool then in the configuration panel click the Open File icon (bottom right, next to Add). Select the *2d_classifications.csv* file you downloaded earlier and open it. You will be presented with a drop-down box to select the field on which to match, again choose 'Sample'. The full set of class assignments will be loaded and samples assigned properly. If you check the view again you'll get two clearly marked groups like the image below:

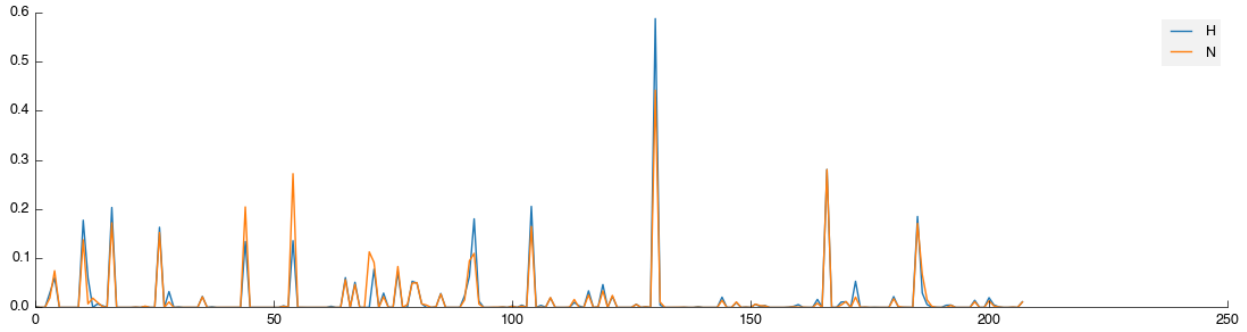


Except it isn't quite. Because we matched a single sample to begin with Pathomx needed a colour to identify the 'No class' group and took the first available (blue). So instead of the above figure, you've probably got one in green and orange. To fix this in the main application window select *Appearance > Line & Marker Styles*. You'll see this:



This dialog is the central control for the appearance of class groups in figures throughout Pathomx. Any change to the colours assigned here determines how they show up in every figure. Select the row for *N* and clicking Edit. For the Line setting click the colour button and then choose something obnoxious like pink. Save the settings by clicking OK, reselect the Reclassify tool and click the green *play* button on the control bar to re-run it. Your *N* line should now be pink.

Enough fun. Go back to *Appearance* > *Line & Marker Styles* and delete all the rows in the panel. Save it and return to your tool, hitting run once more. Now you should have the data visualisation displaying as shown.



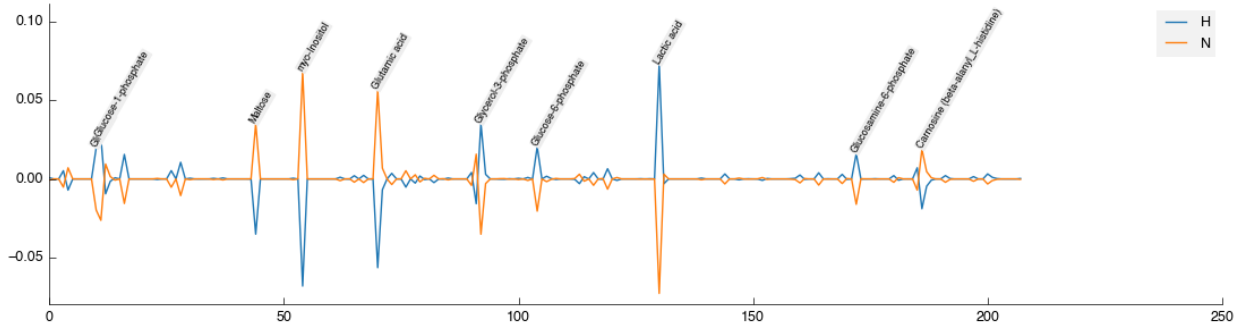
Mapping data to databases

At present, though you wouldn't know it, Pathomx knows nothing about what sort of data this is. It doesn't always matter - you can process and analyse any type of data you like with Pathomx. However to make use of the biological analysis and visualisation tools we need to map the data we've imported to biological entities. The preference in the standard toolkit is to use BioCyc reference entities for this because of the coverage and free access via the public API.

So to begin our biological analysis, let's map our data from the BML-NMR output to BioCyc entities.

Locate the *Map to BioCyc* tool in the Toolbox and drag it into the workflow editor. It will automatically connect to the Reclassify tool already in place. After attempting to process the data for a short while, the tool will finish successfully. However, it's attempting to match using BioCyc metabolite names which don't match exactly with those used in BML-NMR.

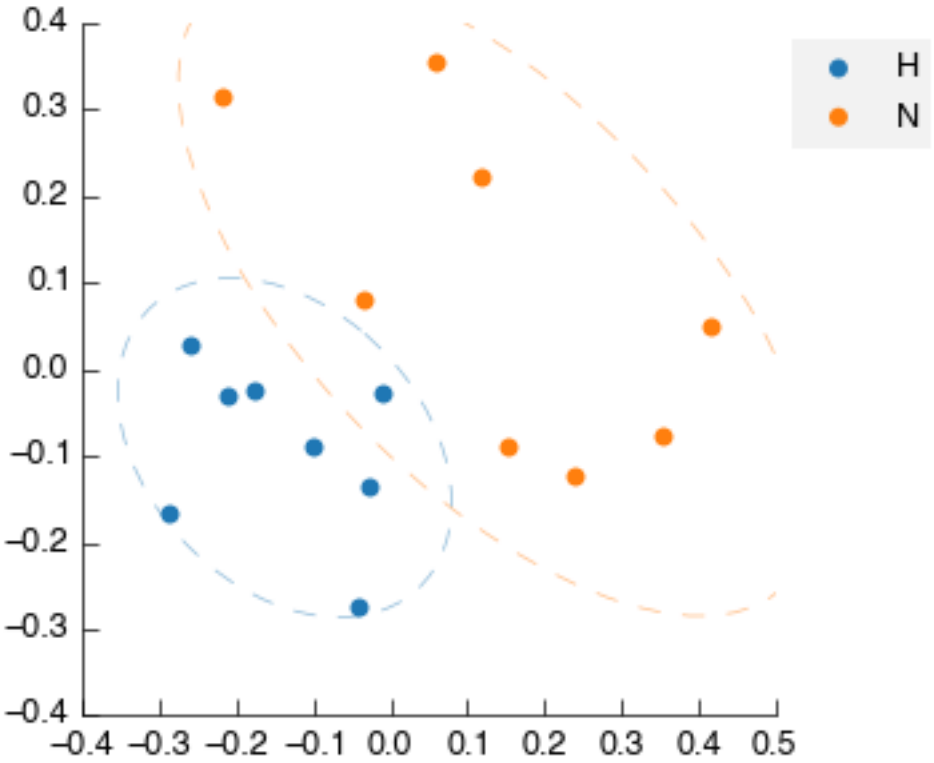
Select the tool to activate the control panel. From the drop-down list select FIMA (the name of the matching algorithm using by BML-NMR). The tool will recalculate and metabolites will be correctly matched. Unfortunately, the tool doesn't yet show what it's done (coming soon!) so for the meantime we can use another tool to get a look. We need to add the *Mean Center* tool anyway so do that now. It will accept the data and run. Select it, then the view tab to see the current state of the data:

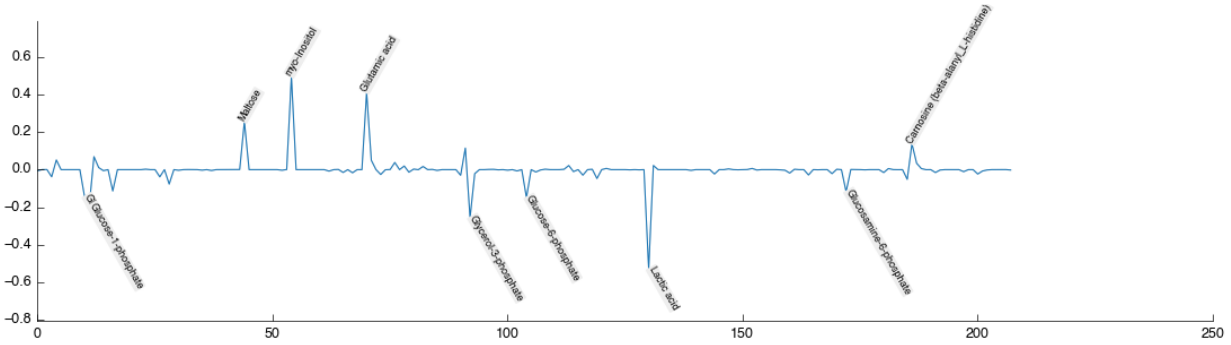


You'll note that as well as being mean centered, the top quantities are now annotated with the metabolite that BML-NMR as identified.

Multivariate analysis

Performing a multivariate analysis can also be accomplished in a quick simple step by dragging and dropping the *PLS-DA* tool from the toolbox into the workflow editor. Again it will auto-connect and auto calculate to produce the following figures:





Note again that on the latent variable plot the data is annotated with the identified and mapped metabolites. This automatic annotation is available on all plots once the data table contains the relevant information (either mapped metabolites or text labels).

Pathway analysis

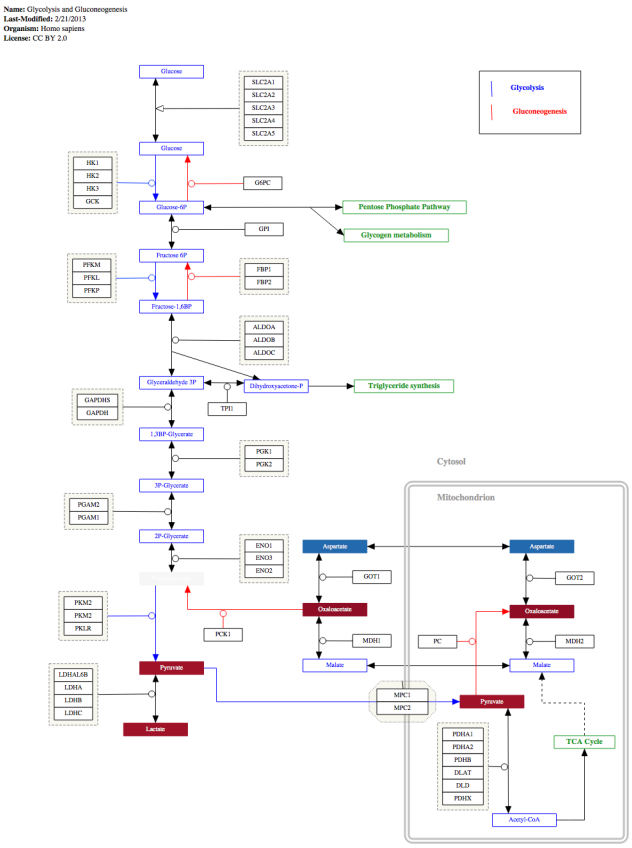
Next we're going to generate three biological analysis visualisations - two using standard pathway maps (WikiPathways) and one using a pathway-mining algorithm approach to generate. However, before we do that we need to get our data into the right shape to allow it.

For visualisation of change it's often useful to use fold-change (2-fold = doubled). However, it's not possible to calculate a fold change from 0 and so a artificial minima must be created. Because the sensitivity of the NMR approach varies for different metabolites we will here apply a local minima (on a per-metabolite basis). The simple tool *Local minima* will do this for us. Drag it into the workflow. It will automatically take input from *Mean center* but replace this by draggin the output from *Map to Biocyc* into the input for *Local minima*.

Next drag and drop a *Fold change* tool into the workflow. Select it and in the configuration panel change the Control setting to *N* and the Test setting to *H*. We'll generate the two WikiPathways views first as they are the most straightforward.

Drag 2x *WikiPathways & GPML* tools into the workflow area. They will probably raise errors (turn red) because there is currently no GPML file defined. You can download them both from WikiPathways here: Glycolysis and TCA Cycle.

Select the first *WikiPathways & GPML* tool and select the *Open GPML* button from the configuration panel. Select the Glycolysis pathway you downloaded (*WP534_74524.gpml*) and open it. You should see the following:



Repeat the process for the TCA cycle visualisation too.

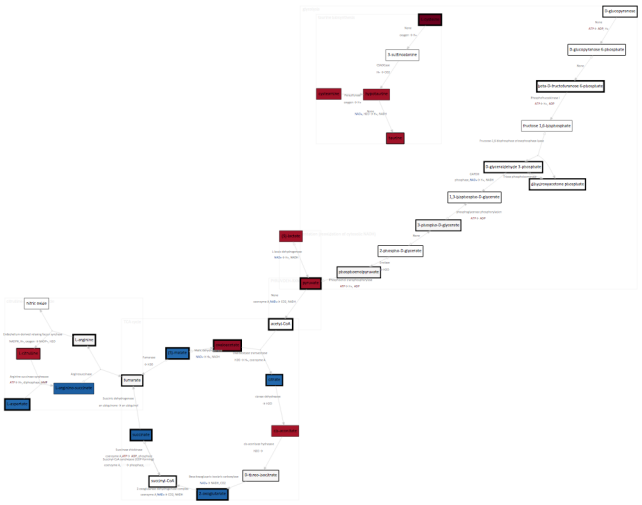
Pre-drawn pathways like these are fine if you know what you're looking for and where the likely biological changes will occur. But sometimes it's useful to be able to visualise experimental data on a pathway map and use *that* to infer the biological basis for what is happening. Pathomx ships with a pathway mining algorithm *pathminer* that allows you to identify the most altered metabolic pathways from a dataset and *metaboviz* a dynamic pathway drawing algorithm. These are available through the tools *Pathway Mining* and *MetaboViz* respectively.

First, drag the *Pathway Mining* tool to the workflow editor. Leave the settings as default for now. Next, drag in a *MetaboViz* tool. The pathway suggestions from the *Pathway Mining* tool will be correctly connected, however you'll also want to drag the output of *Fold change* to the top input *compound_data* on the *MetaboViz* tool. If you click on the MetaboViz tool you'll notice that you already have a pathway map drawn.

We can filter the pathways returned by the pathway mining algorithm to make it easier to visualise (you need <5 usually to get a clear layout). So select the *Pathway Mining* tool and on the Include/Exclude tab of the configuration select to include only the following:

- Biosynthesis/Amino acid biosynthesis
- Degradation/Amino acid degradataion
- Generation/Acetyl-coA
- Generation/Fermentation
- Generation/Glycolysis
- Generation/Other
- Generation/TCA cycle

Which should give you an output not dissimilar to the following -



Things to try out

If you're feeling adventurous there are a few things you can experiment with the workflow -

- Perform a Principal Components Analysis (PCA) *hint: use the output of the Mean Center tool*
- Export the list of mined pathways to a CSV format file *hint: use Export dataframe*
- See if metabolites in the dataset correlate *hint: use the Regression tool*

1.3.4 Short Videos

1. Monocyte differentiation to macrophage and subsequent polarization (HG-U133B). Demo analysis with a publicly available GEO dataset. Showing import and workflow construction for PCA, PLS-DA and GPML-based pathway visualisation with the visual editor. (Pathomx v2.0.0) Download Workflow - GEO dataset - GEO metadata - Glycolysis WikiPathway - Video
2. Demonstrating analysis with the new visual editor showing analysis of multiple datasets, inline views and a number of 1D NMR processing tools.
3. Short example analysis of 2D metabolomic data showing the advantages of reusing workflows for analysis. Complete preliminary analysis of the dataset in 1m 30s. (Pathomx v1.0.0)

1.4 Support

If you're having trouble getting Pathomx doing what you want or you have an idea how it can be improved, there are a number of ways you can get help.

1.4.1 BioStars & MetaStars

Support is available on the BioStars (bioinformatics) and MetaStars (metabolomics) Q&A sites. Post your question using the *pathomx* tag and it will be answered.

1.4.2 Mailing list

A low-traffic mailing list is available (hosted on Google Groups) for support and new release announcements.

1.4.3 Bugs, Issues & Suggestions

Pathomx is an open source application and uses Github (a developer community) to host the source code and track bugs and feature requests. If you have either just add it to the list and we'll work with you to resolve it.

1.4.4 Twitter

Releases and announcements are available via @pathomx on twitter.

1.4.5 Email

If you would like to get in touch directly, you can email Martin Fitzpatrick (Lead Developer) directly. Pathomx is currently a spare-time project, please bear with me.

CHAPTER 2

Developers

Below is documentation for core/plugin developers, including documentation on how to set up a developer installation and create custom tools. API documentation is provided but is currently a work in progress documentation is added to the source code. Improvements are welcomed as pull-requests on Github.

2.1 Developer Installation

If you would like to help with Pathomx development you will need to install a source version of the code. Note: This is not necessary if you just want to contribute plugins, as these can be developed against the binary installation.

2.1.1 Getting Started

The development code is hosted on Github. To contribute to development you should first create an account on Github (if you don't have one already), then fork the pathomx/pathomx repo so you have a personal copy of the code. If you're not familiar with Github, there is a useful guide available here.

On your version of the repo (should be <username>/pathomx) you will see an url to clone the repo to your desktop. Take this and then from the command line (in a folder where you want the code to live) enter:

```
git clone <repository-url>
```

After a while you will get a folder named pathomx containing the code.

The following sections list platform-specific setup instructions required to make Pathomx run. Follow the instructions from the section and then you should be ready to run from the command line using:

```
python Pathomx.py
```

2.1.2 Windows

Install Qt4 or Qt5 for Windows. Currently Qt4 is recommended due to a bug with IPython with PyQt5. Make the decision at this point whether to use 64bit or 32bit versions and stick to it.

Install Python 2.7.6 Windows installer from the Python download site.

Install **PyQt4_** or **PyQt5_** (depending on whether you have Qt4 or Qt5 installed)

You can get Windows binaries for most required Python libraries from the Pythonlibs library. At a minimum you will need to install Pip, NumPy, SciPy, Scikit-Learn, Matplotlib, IPython, pyzmq. Make sure that the installed binaries match the architecture (32bit/64bit) and the installed Python version.

With those installed you can now add the final dependencies via Pip:

    pip install ipython jsonschema jsonpointer mistune mplstyler pyqtconfig metaviz biocyc

To run Pathomx from the command line, change to the cloned git folder and then enter:

```
python Pathomx.py
```

2.1.3 Windows Using Anaconda

Install Anaconda for Windows. Link to the website is http://continuum.io/downloads. Make the decision at this point whether to use 64bit or 32bit versions and stick to it.

With Anaconda installed, open the Anaconda command prompt and you can add the final dependencies.

    pip install mplstyler yapsy pyqtconfig.

To run Pathomx from the command line, change to the cloned git folder and then enter:

```
python Pathomx.py
```

2.1.4 MacOS X

The simplest approach to setting up a development environment is through the MacOS X package manager Homebrew. It should be feasible to build all these tools from source, but I'd strongly suggest you save yourself the bother.

Install Homebrew as follows:

```
ruby -e "$(curl -fsSL https://raw.github.com/Homebrew/homebrew/go/install)"
```

Once that is in place use brew install to install python and PyQt4 (which will automatically install Qt4). From the command line enter:

```
brew install python pyqt
```

You can opt to install pyqt5 instead, however currently this offers lower performance and requires bleeding-edge matplotlib/IPython to function. Next use pip to install all required Python packages. This can be done in a one liner with pip:

```
pip install numpy scipy pandas matplotlib scikit-learn poster yapsy pyqtconfig mplstyler
pip install ipython[all]
```

You can also optionally install the following for some biological data analysis notebooks:

```
brew install graphviz
pip install pydot nmrglue gpml2svg icoshift biocyc metaviz
```

That should be enough to get Pathomx up and running from the command line. To run Pathomx from the command line, change to the cloned git folder and then enter:

```
python Pathomx.py
```

2.1.5 MacOS X Using Anaconda

Install Anaconda for MacOS X. Link to the website is http://continuum.io/downloads.

With Anaconda installed, open the terminal on Mac and you can add the final dependencies.

pip install mplstyler yapsy pyqtconfig

To run Pathomx from the command line, change to the cloned git folder and then enter:

```
python Pathomx.py
```

Troubleshooting

1. Since the master branch of Pathomx is tracking the latest dev tag of iPython, and Anaconda pulls in a release version (might not be the latest), there can be import errors. This can be fixed by performing the following steps to pull in the latest release or dev version of iPython:

   (a) **Try updating iPython to the latest release version:**

   - conda update conda
   - conda update ipython

   (b) **If this doesn't work, try pulling in the latest dev version of iPython:**

   - git clone –recursive https://github.com/ipython/ipython.git
   - cd ipython
   - pip install -e ".[notebook]" –user

2.1.6 Linux

The development version (available via git) supports Python 3 and so can now be run on Linux (tested on Ubuntu Saucy Salamander). Note: Python 3 PyQt5 is only available from 13.10. To install on earlier releases of Ubuntu you will need to install from source.

Install prerequisites:

```
sudo apt-get install g++ python3 python3-dev python3-pip git gfortran libzmq-dev
sudo apt-get install python3-pyqt5 python3-pyqt4 python3-matplotlib python3-requests python3-numpy py
sudo apt-get install libblas3gf libblas-dev liblapack3gf liblapack-dev libatlas3gf-base
```

Build and install latest matplotlib:

```
# Ensure that you have source code repositories enabled
sudo apt-get build-dep python-matplotlib

git clone git://github.com/matplotlib/matplotlib.git
cd matplotlib
sudo python3 setup.py install
cd -
rm -r matplotlib
```

Finally, let's install your develop version of Pantomx:

```
sudo pip3 install openpyxl==1.8.6 pyzmq scikit-learn
cd pantomx
sudo python3 setup.py develop
cd -
```

Note that aside from python3-pyqt5 you can also install the other packages using pip3 (the names on PyPi are the same as for the packages minus the python3- prefix). Once installation of the above has completed you're ready to go.

To run Pathomx from the command line, change to the cloned git folder and then enter:

```
python Pathomx.py
```

2.2 API Reference

The API reference is intended for developers of Pathomx and associated plugins. It lists the interfaces that are available to developers to create views and behaviours. Most complex stuff (e.g. processing thread generation) is handled by the core application and exposed as simplified interfaces for use.

Developing a plugin? Use one of the core plugins as a starting point and use this reference to interpret what it's doing.

2.2.1 Pathomx

2.2.2 Data

class pathomx.data.**DataDefinition**(*target*, *definition={}*, *title=None*, **args*, ***kwargs*)

can_consume(*data*)

check(*o*)

cmp_map = {u'>=': <built-in function ge>, u'=': <built-in function eq>, u'<=': <built-in function le>, u'aloeic': <function

get_cmp_fn(*s*)

class pathomx.data.**DataManager**(*parent*, *view*, **args*, ***kwargs*)

add_input(*interface*)

add_output(*interface*, *dso=None*, *is_public=True*)

can_consume(*source_manager*, *source_interface*, *consumer_defs=None*, *interface=None*)

can_consume_which_of(*molist*, *consumer_defs=None*)

consume(*source_manager*, *source_interface*)

consume_any_app(*app_l*)

consume_with(*data*, *consumer_def*)

consumed = <pathomx.qt.pyqtSignal object>

get(*interface*)

geto(*interface*)

has_consumable(*manager*)

interfaces_changed = <pathomx.qt.pyqtSignal object>

notify_watchers(*interface*)

output_updated = <pathomx.qt.pyqtSignal object>

provide(*target*)

put(*interface*, *dso*, *update_consumers=True*)

refresh_consumed_data()

remove_input(*interface*)

remove_output(*interface*)

reset()

source_updated = <pathomx.qt.pyqtSignal object>

stop_consuming(*target*)

unconsumed = <pathomx.qt.pyqtSignal object>

unget(*interface*)

unput(*interface*)

class pathomx.data.**DataTreeItem**(*dso*, *header*, *parentItem*)
a python object used to return row/column data, and keep note of it's parents and/or children

appendChild(*item*)

child(*row*)

childCount()

columnCount()

data(*column*)

icon()

parent()

row()

class pathomx.data.**DataTreeModel**(*dsos=[]*, *parent=None*)
a model to display a few names, ordered by sex

columnCount(*parent=None*)

data(*index*, *role*)

headerData(*column*, *orientation*, *role*)

index(*row*, *column*, *parent*)

parent(*index*)

refresh()

rowCount(*parent=<pathomx.qt.QModelIndex object>*)

setupModelData()

class pathomx.data.**ImageDataDefinition**(*target*, *definition={}*, *title=None*, **args*, ***kwargs*)
Custom matching definition for PIL Images

check(*o*)

class pathomx.data.**NumpyArrayDataDefinition**(*target*, *definition={}*, *title=None*, **args*, ***kwargs*)
Custom matching definition for numpy arrays

check(*o*)

class pathomx.data.**PandasDataDefinition**(*target*, *definition={}*, *title=None*, **args*, ***kwargs*)
Custom matching definition for pandas dataframes

check(*o*)

pathomx.data.**at_least_one_element_in_common**(*l1*, *l2*)

2.2.3 Custom Exceptions

exception pathomx.custom_exceptions.**PathomxExternalResourceTimeoutException**

exception pathomx.custom_exceptions.**PathomxExternalResourceUnavailableException**

exception pathomx.custom_exceptions.**PathomxIncorrectFileFormatException**

exception pathomx.custom_exceptions.**PathomxIncorrectFileStructureException**

2.2.4 Utils

class pathomx.utils.**UnicodeReader**(*f*, *dialect=<class csv.excel>*, *encoding=u'utf-8'*, ***kwargs*)
A CSV reader which will iterate over lines in the CSV file "f", which is encoded in the given encoding.

next()

class pathomx.utils.**UnicodeWriter**(*f*, *dialect=<class csv.excel>*, *encoding=u'utf-8'*, ***kwargs*)
A CSV writer which will write rows to CSV file "f", which is encoded in the given encoding.

writerow(*row*)

writerows(*rows*)

pathomx.utils.**find_packager**()

pathomx.utils.**invert_direction**(*direction*)

pathomx.utils.**lumina**(*R*, *G*, *B*)

pathomx.utils.**luminahex**(*hex*)

pathomx.utils.**mkdir_p**(*path*)

pathomx.utils.**nonull**(*stream*)

pathomx.utils.**sigstars**(*p*)

pathomx.utils.**swap**(*ino*, *outo*)

pathomx.utils.**which**(*program*)

2.2.5 Views

2.2.6 Plugins

2.2.7 Display Objects

class pathomx.displayobjects.**BaseObj**(*data*, ***kwargs*)

class pathomx.displayobjects.**Html**(*data*, ***kwargs*)

class pathomx.displayobjects.**Markdown**(*data*, ***kwargs*)

class pathomx.displayobjects.**Svg**(*data*, ***kwargs*)

2.2.8 Figures

pathomx.figures.**category_bar**(*data*, *figure=None*, *styles=None*)

pathomx.figures.**difference**(*data1*, *data2*, *figure=None*, *ax=None*, *styles=None*)

pathomx.figures.**extend_limits**(*a*, *b*)

pathomx.figures.**find_linear_scale**(*data*)

pathomx.figures.**get_text_bbox_data_coords**(*fig*, *ax*, *t*)

pathomx.figures.**get_text_bbox_screen_coords**(*fig*, *t*)

pathomx.figures.**heatmap**(*data*, *figure=None*, *ax=None*, *styles=None*)

pathomx.figures.**histogram**(*data*, *bins=100*, *figure=None*, *ax=None*, *styles=None*, *regions=None*)

pathomx.figures.**plot_cov_ellipse**(*cov*, *pos*, *nstd=2*, ***kwargs*)

Plots an *nstd* sigma error ellipse based on the specified covariance matrix (*cov*). Additional keyword arguments are passed on to the ellipse patch artist.

cov : The 2x2 covariance matrix to base the ellipse on pos : The location of the center of the ellipse. Expects a 2-element

sequence of [x0, y0].

nstd [The radius of the ellipse in numbers of standard deviations.] Defaults to 2 standard deviations.

Additional keyword arguments are pass on to the ellipse patch.

A matplotlib ellipse artist

pathomx.figures.**plot_point_cov**(*points*, *nstd=2*, ***kwargs*)

Plots an *nstd* sigma ellipse based on the mean and covariance of a point "cloud" (points, an Nx2 array).

points : An Nx2 array of the data points. nstd : The radius of the ellipse in numbers of standard deviations.

Defaults to 2 standard deviations.

Additional keyword arguments are pass on to the ellipse patch.

A matplotlib ellipse artist

pathomx.figures.**scatterplot**(*data*, *figure=None*, *ax=None*, *styles=None*, *lines=[]*, *label_index=None*)

pathomx.figures.**spectra**(*data*, *figure=None*, *ax=None*, *styles=None*, *regions=None*)

2.2.9 Kernel Helpers

class pathomx.kernel_helpers.**PathomxTool**(*name*, **args*, ***kwargs*)

Simple wrapper class that holds the output data for a given tool; This is for user-friendliness not for use

class pathomx.kernel_helpers.**open_with_progress**(*f*, **args*, ***kwargs*)

check_and_emit_progress()

read(**args*, ***kwargs*)

pathomx.kernel_helpers.**pathomx_notebook_start**(*vars*)

pathomx.kernel_helpers.**pathomx_notebook_stop**(*vars*)

pathomx.kernel_helpers.**progress**(*progress*)
Output the current progress to stdout on the remote core this will be read from stdout and displayed in the UI

2.2.10 Run Queue

2.2.11 Translate

pathomx.translate.**tr**(*s*, **args*, ***kwargs*)

2.2.12 UI

2.3 Creating Custom Tools

This is a brief guide to creating custom tools within Pathomx. This will become easier in the future. However, if you need to create a custom tool *now* this is the way to do it.

2.3.1 Do I need a custom tool?

Custom tools allow you to access the full capabilities of the Pathomx software. The goal of a custom tool will be to create a reusable component that you can use, re-use and share with other users of the software (preferably by adding it to the main repository). In particular they give you access to -

- Tool configuration including widgets (control panels) and defaults
- Define custom plots + plot types

You don't need to create a custom tool if you just want to -

- Do some custom scripting
- Do a one-off custom plot
- Do a one-off anything

For those type of things you're better off just using the built-in *custom script* tool.

2.3.2 What do I need to get started?

Any standard installation of Pathomx will be OK. If you are using Python packages not in the standard installation you may need to use either the developer installation or add custom Python path definitions to Pathomx. But to learn the basics it's best to stick to exploring with NumPy, SciPy and Pandas.

2.3.3 The tool stub

All tools follow a basic structure we're going to call the *tool stub*. To get started on custom tool, simply download the tool stub to your local machine. Unzip the file somewhere convenient, preferably in a specific folder for custom Pathomx tools. You should end up with the following folder structure:

<root>

- .pathomx-plugin
- __init__.py
- loader.py
- stub.py
- stub.md
- icon.png

A brief description of each follows -

`.pathomx-plugin` indicates that this folder is a Pathomx plugin folder. It also holds some metadata about the plugin in the Yapsy plugin format. However, you don't need to know about that to use it just make your changes to the example provided.

`__init__.py` is an empty file required by Python to import the folder as a module. Leave empty.

`loader.py` contains the code required to initialise the plugin and start up. You can also define config panels, dialogs and custom views (figure plots, etc.) in this file.

`stub.py` contains the actual code for the tool that will run on the IPython kernel. `stub.md` contains the descriptive text in Markdown format.

`icon.png` is the default icon for all tools in this plugin. You can add other icons and define them specifically on a per-tool basis if you require.

You can have more than one tool per plugin using the same loader to initialise them all. This is useful when you have a number of tools that are conceptually related. This is seen in the standard 'Spectra' toolkit that offers a number of tools for dealing with frequency data.

2.3.4 Customising the stub

To create your custom tool start with the stub file and customise from there. For this demo we'll create a custom tool that randomly reorders and drops data on each iteration. We'll call it 'Gremlin'.

Open up the `.pathomx-plugin` file and edit the metadata. The only line you have to edit is `Name` but feel free to edit the other data to match. Do not change the `Module` line as this is needed to load the tool. Next rename `stub.md` and `stub.py` to *gremlin.md* and `gremlin.py` respectively. Then open up `loader.py` in a suitable text editor. We're going to add some features to the Gremlin tool to show how it is done.

In the `loader.py` file you will find the following:

```
class StubTool(GenericTool):
    name = "Stub"
    shortname = 'stub'

    def __init__(self, *args, **kwargs):
        super(StubTool, self).__init__(*args, **kwargs)

        self.config.set_defaults({
        })

        self.data.add_input('input_data')  # Add input slot
        self.data.add_output('output_data')  # Add output slot

class Stub(ProcessingPlugin):

    def __init__(self, *args, **kwargs):
```

```
        super(Stub, self).__init__(*args, **kwargs)
        self.register_tool_launcher(StubTool)
```

There are two parts to the tool. The `StubTool` class that defines the tool and configures set up, etc. and the `Stub` loader which handles registration of the launcher for creating new instances of the tool. You can define as many tools in this file as you want (give them unique names) and register them in the same Stub class __init__.

The name of the tool is defined by the `name` parameter to the tool definition. If none is supplied the tool will take the name of the plugin by default. The `shortname` defines the name of the files that source code and information text are loaded from e.g. `stub.py` and `stub.md`. So change the `shortname` value to *gremlin* and the `name` to *Gremlin*.

Below is this is the default config definition. Here you can set default values for any configuration parameters using standard Python dictionary syntax. We'll add a parameter `evilness` that defines how much damage the gremlin does to your data, and `gremlin_type` that defines what it does. Edit the `self.config` definition to:

```
self.config.set_defaults({
'gremlin_type': 1,
'evilness': 1,
})
```

We've defined the parameters and given them both a default value of 1. These will now be available from within the run kernel as `config['evilness']` and `config['gremlin_type']`.

Below the config definition there are two lines defining the input and output ports of the tool respectively. You can name them anything you like as long as you follow standard Python variable naming conventions. Data will be passed into the run kernel using these names. They are defined as `input_data` and `output_data` by default and that is enough for our gremlin tool.

2.3.5 How to train your Gremlin

The runnable source code for tools is stored in a file named <*shortname*>*.py* in standard Python script style. We've already renamed *stub.py* to *gremlin.py* so you can open that now. In it you'll find:

```
import pandas as pd
import numpy as np
import scipy as sp

# This is your stub source file. Add your code here!
```

That does not a lot. The first three lines simply import a set of standard libraries for working with data: Pandas, NumPy and SciPy. You might not need them all but it's worth keeping them available for now. To start our custom tool we need to add some code to mess up the data. First we need a copy of the input_data to output, then we want to mess it up. Add the following code to the file:

```
import pandas as pd
import numpy as np
import scipy as sp

# This is your stub source file. Add your code here!

from random import randint, choice

# Define the gremlin types, these must be matched in the
# loader config definition
GREMLIN_RANDOM = 1
GREMLIN_DELETE_ROW = 2
```

```
GREMLIN_DELETE_COLUMN = 3
GREMLIN_RANDOM_ROWS = 4
GREMLIN_RANDOM_COLUMNS = 5

output_data = input_data

# Repeat the gremlin action 'evilness' times
for n in range( config['evilness'] ):

    if config['gremlin_type'] == GREMLIN_RANDOM:
        gremlin_type = randint(1,5)
    else:
        gremlin_type = config['gremlin_type']

    if gremlin_type == GREMLIN_DELETE_ROW:
        # Delete random row(s) in the pandas dataframe
        output_data.drop( choice( output_data.columns ), axis=1, inplace=True )

    elif gremlin_type == GREMLIN_DELETE_COLUMN:
        # Delete random column(s) in the pandas dataframe
        output_data.drop( choice( output_data.index ), inplace=True )

    elif gremlin_type == GREMLIN_RANDOM_ROWS:
        # Randomly switch two rows' data
        if output_data.shape[0] < 2:
            raise Exception('Need at least 2 rows of data to switch')

        i1 = randint(0, output_data.shape[0]-1)
        i2 = randint(0, output_data.shape[0]-1)

        output_data.iloc[i1,:], output_data.iloc[i2,:] = output_data.iloc[i2,:], output_data.iloc[i1,


    elif gremlin_type == GREMLIN_RANDOM_COLUMNS:
        # Randomly switch two columns' data
        if output_data.shape[0] < 2:
            raise Exception('Need at least 2 columns of data to switch')

        i1 = randint(0, output_data.shape[0]-1)
        i2 = randint(0, output_data.shape[1]-1)

        output_data.iloc[:,i1], output_data.iloc[:,i2] = output_data.iloc[:,i2], output_data.iloc[:,


# Generate simple result figure (using pathomx libs)
from pathomx.figures import spectra

View = spectra(output_data, styles=styles);
```

This is the main guts of our gremlin. A copy of the `input_data` is made to `output_data` and then a simple loop iterates *evilness* times while performing some or other task on the `output_data`. The choice of actions are: delete row, delete column, switch two rows, switch two columns. An option is available to make a random selection from these transformations. Setting *evilness* to 10 and *gremlin_type* to 1 will perform 100 random operations on the data. Enough to drive anyone quite mad.

Finally, we use built in standard figure plotting tools to output a view of the transformed data.

2.3.6 Initial test

To see what damage the gremlin can do we need a set of data to work with. Download the sample dataset, a set of processed 2D JRES NMR data with class assignments already applied.

Start up Pathomx as normal. Before we can use our Gremlin tool we'll need to tell Pathomx where to find it so it can be loaded. On the main toolbar select "Plugins" then "Manage plugins..." to get to the plugin management view. Here you can activate and deactivate different plugins and add/remove them from the Toolkit view. To find the Gremlin tool we'll need to tell Pathomx about the folder it is in.

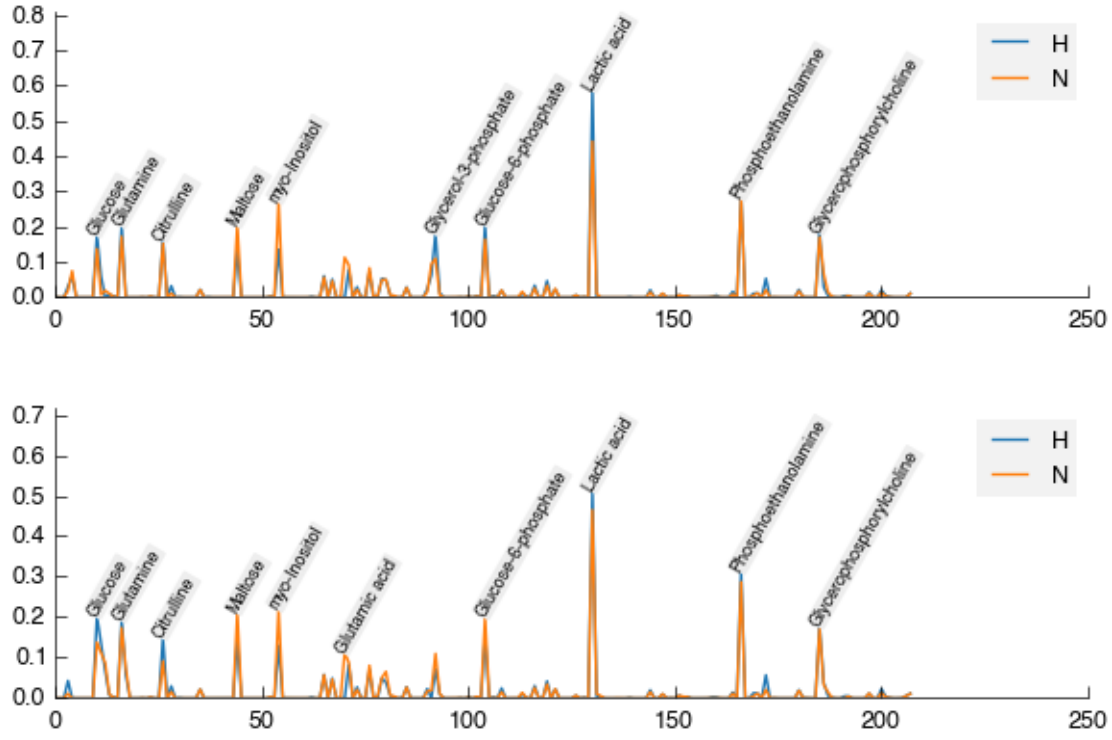
Add the folder containing the Gremlin tool, or alternatively a parent folder if you want to create more tools in the same place. Pathomx will automatically search through the entire tree to find plugins so it's probably best not to add an entire drive.

Once added the plugin list will refresh and be listed (and automatically activated) in the plugin list. You can now close the plugin management list and see that your new tool is ready and waiting in the Toolkit viewer. It will be there every time you run Pathomx.

Drag it into the workspace and click on it. You'll notice that there isn't much to see: there is no configuration UI defined and we haven't updated the about text. But it's still a fully-operational gremlin. So let's see it in action.

Drag an *Import Text/CSV* tool into the workspace and select it. Using the open file widget select the file you downloaded earlier containing the demo dataset. Have a look at the Spectra view output to see how it *should* look.

Now drag from the *Import Text/CSV* `output_data` port to the Gremlin ``input_data`` port. The gremlin tool will automatically calculate using the new data and display a modified plot called 'View'. If you can't see the different between this and the earlier plot try pressing the green *play* button a few times to re-run the tool. You will see the data change each time.



2.3.7 Adding configuration

A tool is not a lot of use without the ability to control it. All tools can be modified by editing the source directly (see the # tab) but that isn't particular convenient. Pathomx tools can define configuration panels, containing multiple widgets that are linked to the defined config settings.

Add the following code to the `loader.py` file.

```
# Configuration settings for the Gremlin
class GremlinConfigPanel(ConfigPanel):

    def __init__(self, *args, **kwargs):
        super(GremlinConfigPanel, self).__init__(*args, **kwargs)

        gd = QGridLayout()

        choices = {
            'Random': 1,
            'Delete row': 2,
            'Delete column': 3,
            'Randomise rows': 4,
            'Randomise columns': 5,
        }

        gremlin_type_cb = QComboBox()
        gremlin_type_cb.addItems( choices.keys() )
        self.config.add_handler('gremlin_type', gremlin_type_cb, choices)
        gd.addWidget( QLabel('Gremlin type'), 0, 0)
        gd.addWidget(gremlin_type_cb, 0, 1)

        evilness_sb = QSpinBox()
        self.config.add_handler('evilness', evilness_sb)
        gd.addWidget( QLabel('Evilness'), 1, 0)
        gd.addWidget(evilness_sb, 1, 1)

        self.layout.addLayout(gd)

        self.finalise()
```

This block of code defines the configuration panel for the tool. This is done using standard Qt (PyQt) widgets and layout code, which won't be gone into detail here. However, the bits unique to Pathomx tool code are worth a bit of explanation:

As previously described tools have an in-built config handler (based on the *pyqtconfig* package available on PyPi). This keeps track of settings and also allows widgets to be attached and automatically synced with configuration settings. This is achieved with `self.config.add_handler` linee. The first parameter is the config key to set, the second the widget and the (optional) third is a mapping dictionary/lambda tuple that converts between the displayed and stored value.

This is used for the drop-down so that when *Random* is displayed, the stored value in the config is actually `1`. These mappings can be applied to any widget and can apply any transformation required. The widget is synced to the config value as it is bound.

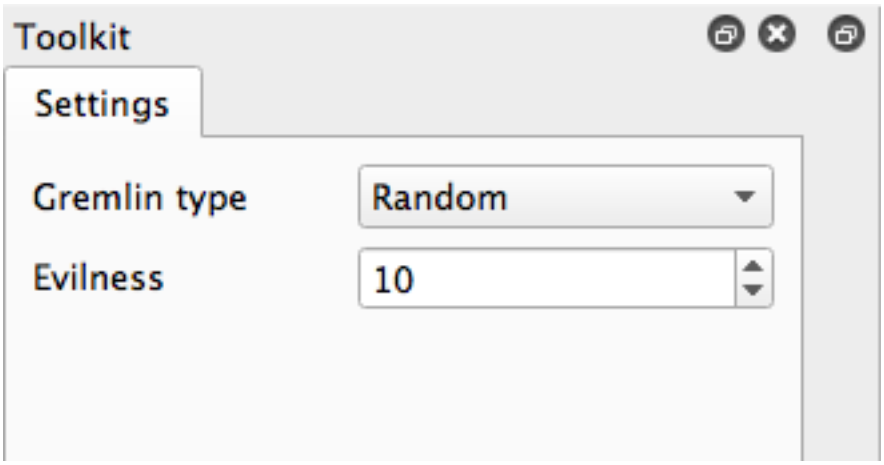
Each `ConfigPanel` has a default `layout` object defined to which your widgets are attached. They can be placed directly using `self.layout.addWidget(widget)` or, as above, by defining a new layout and assigning that. It's usually useful to use a `GridLayout` to place widgets on the panel alongside labels.

Finally, the `self.finalise()` call is required to apply the layouts and wrap up the initialisation.

Next, add the following line to the `__init__` function of the GremlinTool class:

```
self.addConfigPanel(GremlinConfigPanel, 'Settings')
```

...and you're good to go. Restart Pathomx and the Gremlin tool will auto-reload automatically. Drag the tool into the workspace and then select it. On the left hand side you should see your shiny new control panel. Connect the tool up with the sample data as before, and then experiment with the config settings to see the effect.



Since we output the result of the transformation via the `output_data` port you can also connect up other tools and see the effect there. For example, connect up a PCA or PLS-DA tool and see the effect that the gremlin has on the ability of those algorithms to separate the two classes in the dataset.

2.3.8 Polish

Open up the `gremlin.md` file and edit the file to say whatever you would like it to. You can also replace the `icon.png` with a PNG format image more appropriate to an evil gremlin tool.

2.3.9 The end

This doesn't cover everything that is possible within a custom tool, but it should give you enough to get started on your own. If you have any suggestions for improvements of this documentation or want to share your own demos, get in touch.

The complete Gremlin tool is available for download.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

p