
NmrQuant

Release 1.2.7.1

Loic Le Gregam

Dec 15, 2021

USAGE:

1	Quickstart	3
1.1	Installation	3
1.2	Environment installation	3
2	Inputs	9
2.1	The Data File	9
2.2	The Database File	10
2.3	The Template File	11
3	Usage	13
3.1	Jupyter Notebook interface	13
3.2	Command Line Interface	13
4	Reference	15
	Python Module Index	19
	Index	21

NMRQuant is a software for the quantification of metabolites from 1D Proton NMR experiments. It takes as input integrated areas from spectra and proton counts from a database from excel and csv files. It can be used with either internal or external calibration, using a standard molecule (usually d4-TSP). It outputs an excel file containing the calculated concentrations, and svg files containing plots that help with the visualisation and interpretation of the data.

It is one of the routine tools used on the [MetaToul platform](#).

The code is open-source, and available on Github under a GPLv3 license.

The documentation relative to NMRQuant's usage can be found on [ReadTheDocs](#).

It can be used from a command line interface or through an interactive jupyter notebook GUI. The notebook can be downloaded from the [GitHub page](#).

Key Features

- Calculation of metabolite concentrations from 1D H+ NMR integrated data
- Creation of figures (lineplots and barplots) that help with interpretation

QUICKSTART

1.1 Installation

NmrQuant requires Python 3.8 or higher. If you do not have a Python environment configured on your computer, we recommend that you follow the instructions from [Anaconda](#). Then, open a terminal (e.g. run Anaconda Prompt if you have Anaconda installed) and type :

```
pip install nmrquant
```

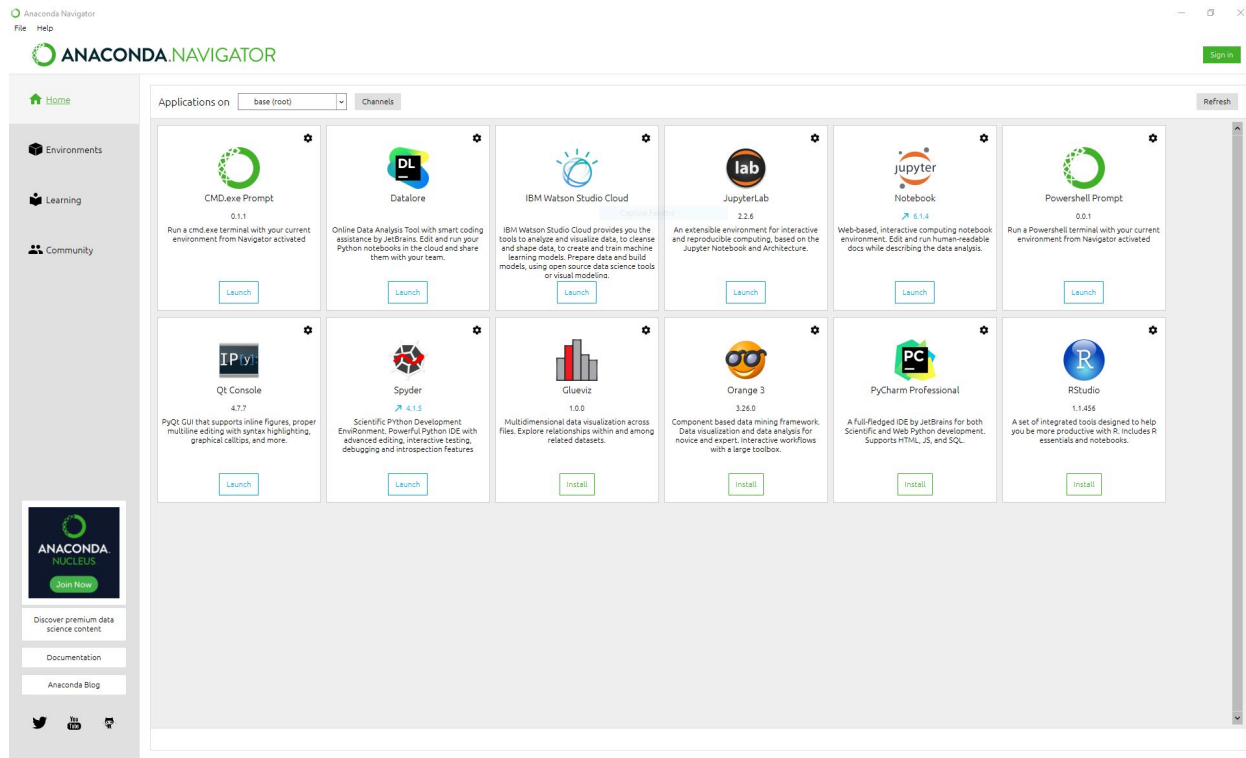
Next, jupyter notebook (JN) needs to be installed. You can find documentation on how to install JN on their [website](#). It is also possible to install JN through Anaconda (see Environments for best practices). Finally, the notebook file containing the gui interface can be found on the [GitHub page](#).

1.2 Environment installation

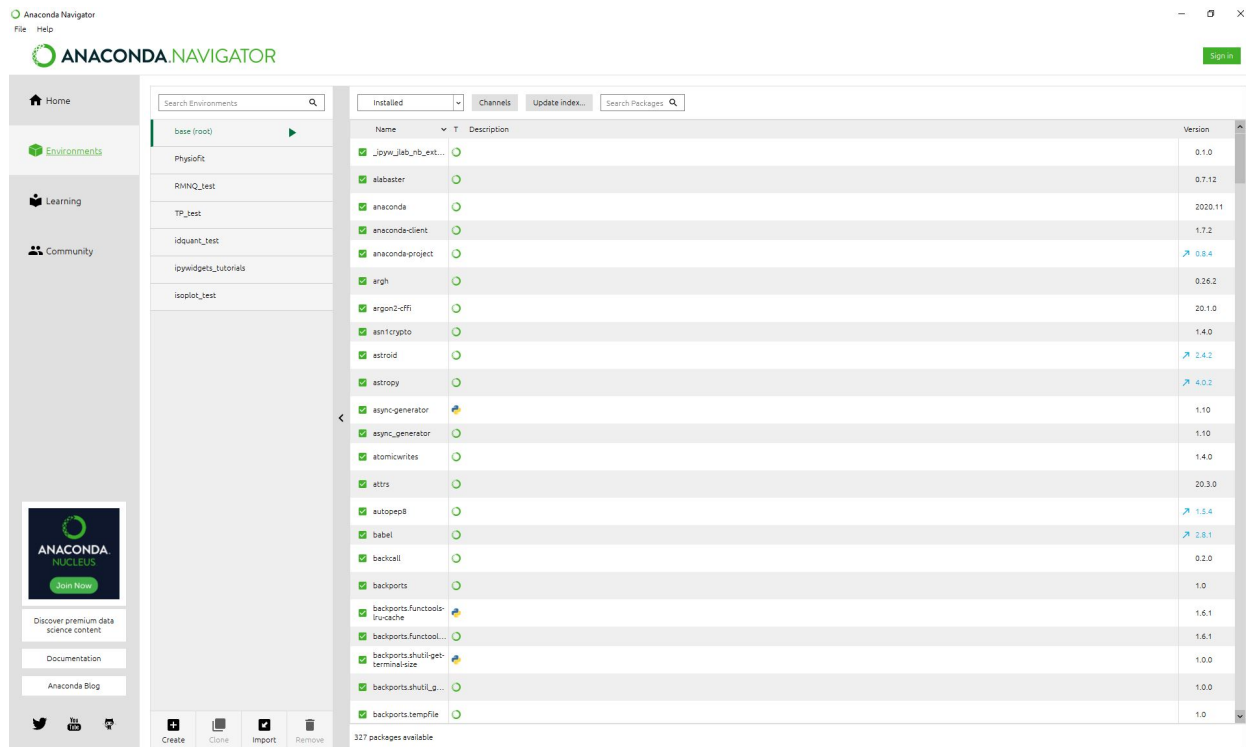
One of the advantages of the Anaconda Suite is that it gives access to a user-friendly GUI for the creation and maintenance of python environments. Python environments give the user a way to separate different installations of tools so that different package dependencies do not overlap with each other. This is especially useful if packages share the same dependencies but in different versions. The Anaconda Suite provides a quick and intuitive way of separating these installations.

1.2.1 How to create an environment in Anaconda

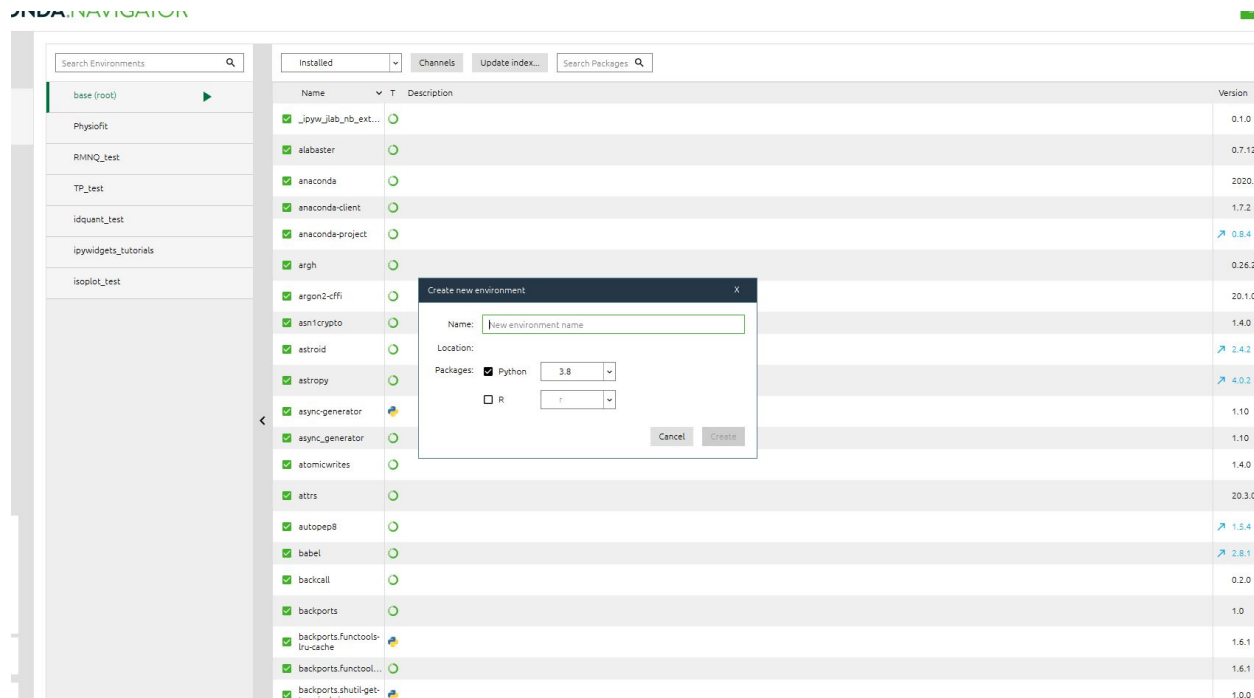
When the user opens up the Anaconda software, she/he ends up on the main menu:



The main window shows all the tools available for installation in the Navigator. To get to the environments page, the user must click on the “Environments” panel that is in the left-side menu.



Once on the Environments page, the user can click on the “create” button that is present at the bottom left of the screen. A pop up menu will then appear and allow the user to select a python version and a name for the environment.



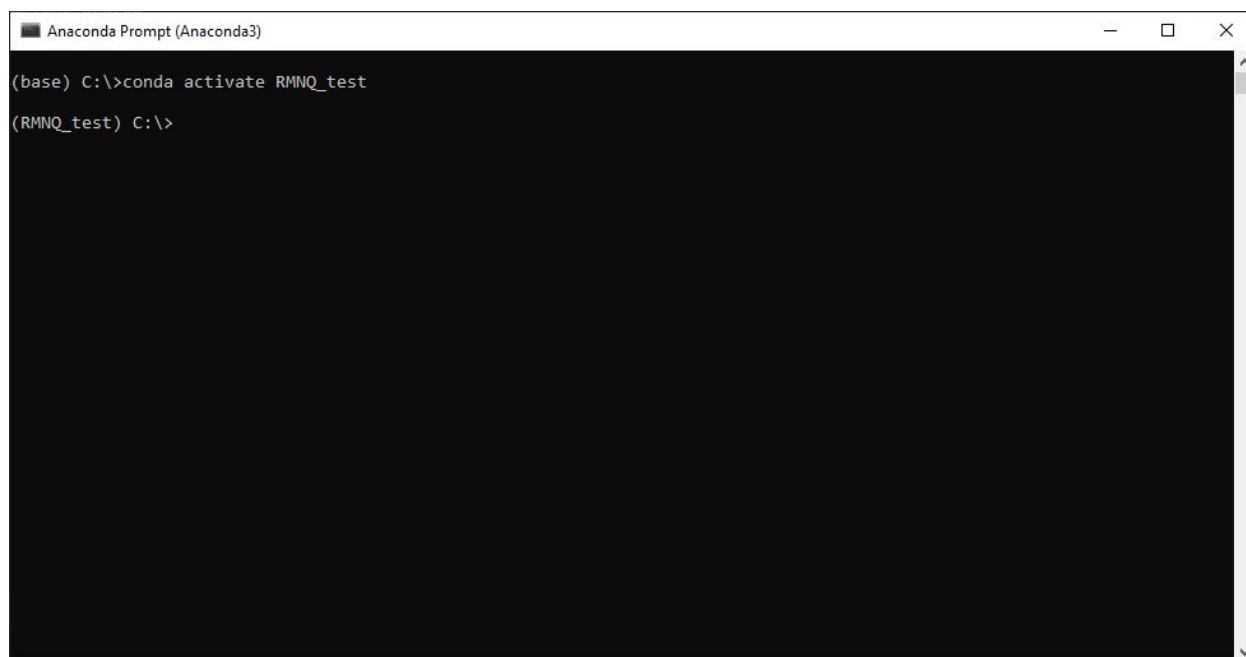
Once the user clicks on the “create” button the environment is created and ready for use!

1.2.2 Installing packages in the environment

Now that the environment exists, it is time to populate it with the tools needed. The first thing to do is to open up a command-line interface, preferably Anaconda Prompt (it is the one that will be used in this tutorial. Other command-line interfaces might use different names for commands). Once the interface is open, the first thing to do is to activate the desired environment. The command for this is as follows:

```
conda activate <name-of-environment>
```

Once this is done the environment name should be seen on the left of the screen behind the name of the directory the interface is open in.

A screenshot of an Anaconda Prompt window. The title bar reads "Anaconda Prompt (Anaconda3)". The command prompt shows the command `(base) C:\>conda activate RMNQ_test` being entered, followed by the prompt `(RMNQ_test) C:\>` on the next line. The background of the terminal is black, and the text is white. The window has standard Windows window controls (minimize, maximize, close) in the top right corner.

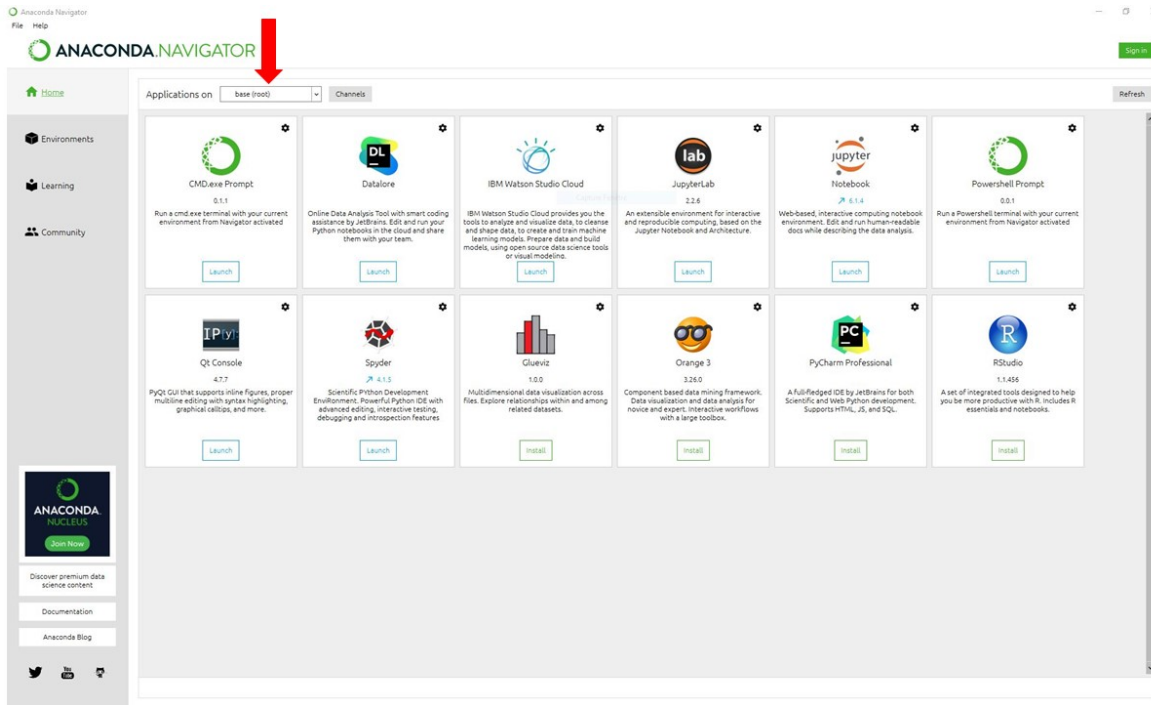
```
(base) C:\>conda activate RMNQ_test
(RMNQ_test) C:\>
```

Once the environment is activated, the user can install using pip or conda any of the desired tools. The dependencies and the tool itself will now be installed in a safe and separate set of folders which will ensure that other installations are not affected by anything happening in the environment. Once the user is done, she/he can now close the prompt.

1.2.3 Installing and launching Jupyter Notebooks through the Navigator

There are two ways of installing Jupyter Notebook (JN): through the command-line or through the Anaconda Navigator. We will here explain the way to do it the second way.

First, the user must open the Anaconda Navigator. Once on the main page, the first thing to do is to select the desired environment of installation of JN by clicking on the dropdown menu shown below and selecting the environment.



Once the environment is selected, the user can search for JN in the main page and click the install button underneath. Once JN is installed, the last step is to launch the application.

INPUTS

2.1 The Data File

2.1.1 File structure

The data accepted by NMRQuant should be passed in as an excel file (.xlsx), a comma-separated text file (.csv) or a tabulation-separated text file (.tsv). The structure of the file must be as follows:

# Spectrum#	Metabolite1	Metabolite2	Metabolite3
1	xxx	xxx	xxx
2	xxx	xxx	xxx
3	xxx	xxx	xxx
4	xxx	xxx	xxx

The number of metabolites processed at a time has no limit. The important column here is the first one: “# Spectrum#”, as it will be responsible for linking the recorded values per metabolite with the different times, conditions and replicates that will be given through the metadata file (ref).

Note: The column header is case-sensitive! Since in most cases the spectrum data comes from spectra integrated using Bruker’s TopSpin software and this is the column header used by their software, for practical reasons the same formalism was kept in NMRQuant.

Every other column must have as header the name of a metabolite, and the associated areas should be given in each row. If a metabolite has more than one integrated area, **it should be given in two or more columns and a number should be assigned to each of the column headers with an underscore between the metabolite name and the number**. For example:

# Spectrum#	Phenylalanine_1	Phenylalanine_2	Phenylalanine_3
1	xxx	xxx	xxx
2	xxx	xxx	xxx
3	xxx	xxx	xxx
4	xxx	xxx	xxx

Note: The number of protons for each area group should be given in the database file using the same nomenclature as in the data file, including the spaces and numbers.

Warning: To calculate concentrations for a metabolite using multiple integration areas (as for phenylalanine in the example above), make sure that the proton count for each area is referenced in the database.

2.1.2 Calibration molecule input

To calculate concentrations, NMRQ needs to know if the calibration is internal or external. To know this, it searches in the data file for a column named “Strd” (for *Standard*). This column is added in manually by the user, and must have at least in its first row the number 1 (if the calibration is internal) or 9 (if the calibration is external). If the value is equal to 1, the Strd’s concentration is not needed and the user does not need to input it in the notebook or the Command-Line Interface (CLI). On the contrary, if the value is equal to 9, the user will have to give the TSP concentration through the notebook or the CLI.

2.2 The Database File

To calculate concentrations from 1D H NMR spectra areas, it is necessary to have the number of protons for each integrated region (corresponding to an equivalent group of protons in a molecule). Consequently, it is necessary to pass this information to NMRQuant through **an excel or csv file** containing these proton numbers for each metabolite we are quantifying. The structure of the excel file should at least be as follows:

Metabolite	Heq
Formate	1
Phenylalanine	5
Tyrosine	2

As always, the headers for each column are case-sensitive, as are the names of each metabolite which have to be exactly the same as the ones used in the data file. The two columns shown above are the minimal requirements for NMRQ to function. Good practices dictate that users also add a column with the different ppm positions of each region for informational purposes. This is not a problem and will not interfere with the software’s ability to read the file.

Note: For metabolites that are quantified using two or more regions, the formalism is to add a number **separated from the metabolite name by an underscore**. The same must be done for the metabolite names in the data file:

Metabolite	Heq
Formate	1
Phenylalanine_1	3
Phenylalanine_2	2
Tyrosine	2

Note: If there is no corresponding metabolite in the database file for a given metabolite in the datafile, RMNQ will notify you by adding *_Area* after the metabolite’s name in the output file, and keep the areas in the final results. In this case, the software will also use the area values for plotting. This means that if the user uses an arbitrary name for an unknown integration area (“*unknown*” for example) it will still be plotted and put in the results.

2.3 The Template File

Once the datafile is uploaded into the notebook or given in the command-line interface (CLI), the user can generate a structured template file in which the Time Points, Conditions and Replicate numbers must be referenced. To do this, the program reads the '# Spectrum#' field in the data file and generates the required number of rows for each spectrum with the correct formalism for the headers.

Warning: Do not change the names of the columns in the generated template file as this will stop NMRQ from reading the file correctly!

The metadata given through the template file will then be used to separate the datas in groups for plotting.

Note: The Replicates column must number each individual sample of similar Times + Conditions from 1 to n (n being the last replicate). This will let the software mean the concentrations and also create the summary plots and meaned histograms with error bars.

3.1 Jupyter Notebook interface

The first step is to use the buttons to upload the files into the notebook (or generate the template).

1. **Upload datafile.** This is always the first thing to do when you first launch the notebook. Use the “*Upload datafile*” button to select the data file. This will enable the other upload buttons.
2. **Uplaod Database.** Use this button to upload the database file into the notebook.
3. **Generate Template** Next, if needed, click the “*Generate Template*” button to create the Template in the folder containing the notebook file. Fill out the file with the experiment’s metadata
4. **Uplaod Template** Use this button to upload the template into the notebook.

Now that the different input files are loaded, you can choose if the means should be calculated for the run by clicking or not on the *mean export* checkbox. Once this is done, fill out the run name (to name the end folder), the dilution factor and the concentration in standard molecule (if calibration is external). Once this is done, press the “Calculate” button to generate the results file. To visualize the data using the plotting module, finish by choosing the plots you want from the multiple selection list, and click the “Make plots” button.

3.2 Command Line Interface

To process your data, type in a terminal:

```
nmrquant [command line options]
```

Here after the available options with their full names are enumerated and detailed.

```
usage: nmrquant [-h] [-d DATABASE] [-F DILUTION_FACTOR] [-t TEMPLATE]
               [-k MAKE_TEMPLATE] [-f FORMAT] [-b {individual,meaned}]
               [-l {individual,meaned}] [-m] [-c TSP_CONCENTRATION]
               [-e EXPORT] [-v]
               datafile
```

3.2.1 Positional Arguments

datafile Path to data file to process

3.2.2 Named Arguments

-d, --database Path to proton database

-F, --dilution_factor Dilution factor used to calculate concentrations
Default: 1.11

-t, --template Path to template file.

-k, --make_template Input path to export template to

-f, --format Choose a format for the plots. Choices: svg, png, jpeg
Default: "svg"

-b, --barplot Possible choices: individual, meaned
Choose histogram to build. Enter "individual" or "meaned"

-l, --lineplot Possible choices: individual, meaned
Choose lineplot to build. Enter "individual" or "meaned"

-m, --mean Add if means and stds should be calculated on replicates
Default: False

-c, --tsp_concentration Add tsp concentration if calibration is external

-e, --export Name for exported file

-v, --verbose Add option for debug mode
Default: False

NmrQuant proceeds automatically to the data processing and displays progress and important messages in the standard output.

REFERENCE

file *calculator.py*

Module containing the main Data Analyzer

class `nmrquant.engine.calculator.Quantifier` (*verbose=False*)

Bases: `object`

RMNQ main class to quantify and visualize data

calculate_concentrations (*strd_conc=1*)

Calculate concentrations using number of protons and dilution factor

Parameters **strd_conc** – Standard concentration for external calibration. If calibration is internal, concentration is equal to one.

Return **self.conc_data** Dataframe containing calculated concentrations

compute_data (*strd_conc=None, mean=False*)

Run data preparation and computation of concentrations (if *strd_conc* is not `None`, else just prepare data)

Parameters **strd_conc** – Concentration of standard molecule used (1 if concentration is not needed). Set to `None` if

concentrations must not be calculated :param *mean*: should means be computed

export_data (*destination, file_name="", fmt='excel', export_mean=False*)

Export final data in desired format

generate_metadata (*path*)

Generate template in excel format

get_data (*data, excel_sheet=0*)

Get data from path or excel file

get_db (*database*)

Get database from file or path

Parameters **database** – Can be a file directly or a str containing the path to the file

import_md (*md*)

Import metadata file after modification from path or file

Parameters **md** – Can be a file directly or a str containing the path to the file

file *visualizer.py*

class `nmrquant.engine.visualizer.HistPlot` (*input_data, metabolite, display*)

Bases: `abc.ABC`

Histogram Abstract base class. All histograms will derive from this class. The initial cleanup and preparation of data is done here. Any modifications will be passed down to children classes. Global checkups should be performed in the constructor of this class. The class implements repr and call dunder methods. The call dunder requests plot creation from the build_plot method so that self() directly creates the plot.

abstract build_plot()

class nmrquant.engine.visualizer.**IndHistA**(input_data, metabolite, display)

Bases: *nmrquant.engine.visualizer.HistPlot*

Class for histogram with one or more conditions, no replicates and one or no time points

build_plot()

class nmrquant.engine.visualizer.**IndHistB**(input_data, metabolite, display)

Bases: *nmrquant.engine.visualizer.HistPlot*

Class for histograms with one or more conditions but only one (or no) time points and multiple replicates (individual representation)

build_plot()

class nmrquant.engine.visualizer.**IndLine**(input_data, metabolite, display)

Bases: *nmrquant.engine.visualizer.LinePlot*

Class to generate lineplots from kinetic data. Each plot is specific to one condition and displays each replicate in a separate line.

build_plot()

class nmrquant.engine.visualizer.**LinePlot**(input_data, metabolite, display=False)

Bases: abc.ABC

Line Plot Abstract base class. All line plots will derive from this class. The initial cleanup and preparation of data is done here. Any modifications will be passed down to children classes. Global checkups should be performed in the constructor of this class. The class implements repr and call dunder methods. The call dunder requests plot creation from the build_plot method so that self() directly creates the plot.

abstract build_plot()

static show_figure(fig)

In case figures are generated but not used directly, we need a way to visualize them later (from inside a list of figures for example)

class nmrquant.engine.visualizer.**MeanLine**(input_data, metabolite, display)

Bases: *nmrquant.engine.visualizer.IndLine*

Line plots with meaned replicates for each time point. We inherit from IndLine to initialize the dict containing all the data for all the replicates. We will then calculate means and SDs from this data.

build_plot()

class nmrquant.engine.visualizer.**MultHistB**(input_data, std_data, metabolite, display)

Bases: *nmrquant.engine.visualizer.HistPlot*

Class for histograms with one or more conditions but only one (or no) time points and multiple replicates (meaned representation)

build_plot()

class nmrquant.engine.visualizer.**NoRepIndLine**(input_data, metabolite, display)

Bases: *nmrquant.engine.visualizer.LinePlot*

Class to generate line plots for kinetic data with only 1 replicate per condition

build_plot()

file *utilities.py*

Module containing extra tools

`nmrquant.engine.utilities.append_value(dict_obj, key, value)`

Add/append values to an existing key to a dictionary

`nmrquant.engine.utilities.check_for_sum(y)`

Check if two args given in y through '+' operator

Parameters y –

`nmrquant.engine.utilities.is_empty(any_structure)`

Check if container is empty

Parameters any_structure – data container to analyze

Returns if empty True, if not False

`nmrquant.engine.utilities.list_average(lst)`

`nmrquant.engine.utilities.read_data(path, excel_sheet=0)`

Function to read incoming data

Parameters

- **path** (*str or pathlib.PurePath*) – path to data to read
- **excel_sheet** (*int*) – excel sheet to read (if data is excel file with multiple sheets)

file *notebook.py*

class `nmrquant.ui.notebook.Rnb(verbose=False)`

Bases: `object`

Class to control RMNQ notebook interface

build_plots(event)

Control plot creation. Make destination folders and generate plots.

generate_template(event)

Generate template from input data spectrum count

load_events()

Load events for all the different buttons

make_gui()

Display the widgets and build the GUI

process_data(event)

Make destination folder, clean data and calculate results

reset(verbose)

Function to reset the object in notebook (only for notebook use because otherwise cell refresh doesn't reinitialize the object)

- search

PYTHON MODULE INDEX

n

`nmrquant.engine.calculator`, [15](#)
`nmrquant.engine.utilities`, [17](#)
`nmrquant.engine.visualizer`, [15](#)
`nmrquant.ui.notebook`, [17](#)

A

`append_value()` (in module `nmrquant.engine.utilities`), 17

B

`build_plot()` (`nmrquant.engine.visualizer.HistPlot` method), 16

`build_plot()` (`nmrquant.engine.visualizer.IndHistA` method), 16

`build_plot()` (`nmrquant.engine.visualizer.IndHistB` method), 16

`build_plot()` (`nmrquant.engine.visualizer.IndLine` method), 16

`build_plot()` (`nmrquant.engine.visualizer.LinePlot` method), 16

`build_plot()` (`nmrquant.engine.visualizer.MeanLine` method), 16

`build_plot()` (`nmrquant.engine.visualizer.MultHistB` method), 16

`build_plot()` (`nmrquant.engine.visualizer.NoRepIndLine` method), 16

`build_plots()` (`nmrquant.ui.notebook.Rnb` method), 17

C

`calculate_concentrations()` (`nmrquant.engine.calculator.Quantifier` method), 15

`check_for_sum()` (in module `nmrquant.engine.utilities`), 17

`compute_data()` (`nmrquant.engine.calculator.Quantifier` method), 15

E

`export_data()` (`nmrquant.engine.calculator.Quantifier` method), 15

G

`generate_metadata()` (`nmrquant.engine.calculator.Quantifier` method), 15

H

`generate_template()` (`nmrquant.ui.notebook.Rnb` method), 17

`get_data()` (`nmrquant.engine.calculator.Quantifier` method), 15

`get_db()` (`nmrquant.engine.calculator.Quantifier` method), 15

I

`HistPlot` (class in `nmrquant.engine.visualizer`), 15

L

`import_md()` (`nmrquant.engine.calculator.Quantifier` method), 15

`IndHistA` (class in `nmrquant.engine.visualizer`), 16

`IndHistB` (class in `nmrquant.engine.visualizer`), 16

`IndLine` (class in `nmrquant.engine.visualizer`), 16

`is_empty()` (in module `nmrquant.engine.utilities`), 17

M

`LinePlot` (class in `nmrquant.engine.visualizer`), 16

`list_average()` (in module `nmrquant.engine.utilities`), 17

`load_events()` (`nmrquant.ui.notebook.Rnb` method), 17

N

`make_gui()` (`nmrquant.ui.notebook.Rnb` method), 17

`MeanLine` (class in `nmrquant.engine.visualizer`), 16

module
`nmrquant.engine.calculator`, 15
`nmrquant.engine.utilities`, 17
`nmrquant.engine.visualizer`, 15
`nmrquant.ui.notebook`, 17

`MultHistB` (class in `nmrquant.engine.visualizer`), 16

O

`nmrquant.engine.calculator`
module, 15

`nmrquant.engine.utilities`
module, 17

`nmrquant.engine.visualizer`

module, [15](#)
nmrquant.ui.notebook
module, [17](#)
NoRepIndLine (*class in nmrquant.engine.visualizer*),
[16](#)

P

process_data() (*nmrquant.ui.notebook.Rnb*
method), [17](#)

Q

Quantifier (*class in nmrquant.engine.calculator*), [15](#)

R

read_data() (*in module nmrquant.engine.utilities*),
[17](#)
reset() (*nmrquant.ui.notebook.Rnb method*), [17](#)
Rnb (*class in nmrquant.ui.notebook*), [17](#)

S

show_figure() (*nmrquant.engine.visualizer.LinePlot*
static method), [16](#)