dnpLab Documentation

Release 1.0.0

Timothy Keller

Aug 18, 2020

Documentation:

Welcome to the dnpLab documentation. dnpLab is an Open Source Python package for importing and processing Dynamic Nuclear Polarization (DNP) data. The aim of the project is to provide a free, turn-key processing package for easy processing and analysis of DMP-NMR data.

dnpLab is a collaborative project created by

-
- The at University of California, Santa Barbara
- The at Syracuse University
- License: MIT License

The source code for the project is published at: XXXX Need GitHub Link XXXX

Current Release	1.0.0
Documentation Build Date	08/18/2020, 10:01:24
Author(s)	The dnpLab Team

CHAPTER 1

Features

- Import NMR spectra in various formats (Bruker - TopSpin, Varian - (Open) VnmrJ, Magritek - Kea)
- Process NMR data
- Extract Hydration Dynamics Information
- Create Publication Quality Figures

1.1 Introduction to dnpLab

The aim of dnpLab is to provide a turn-key data processing environment for DNP-NMR data. The software package is entirely written in Python and no proprietary software is required. dnpLab is published under the open-source MIT license.

In the following section, we introduce the intended workflow for processing DNP-NMR data with dnpLab. The dnpLab python package supports data formats of all major NMR platforms.

The general workflow is as follows:

1. Import DNP-NMR Data
2. Create Workspace
3. Process Data
4. Save Data in h5 Format
5. Further Processing and Analysis

A key-feature of dnpLab is creating a workspace. The imported data is stored in a dnpdata object and the first object that is created during the import process is the *raw* object. It contains the raw data from the spectrometer and will be accessible at any time. All processing steps are automatically documented and the entire workspace can be saved as a single file in the h5 format.

1.1.1 Workflow

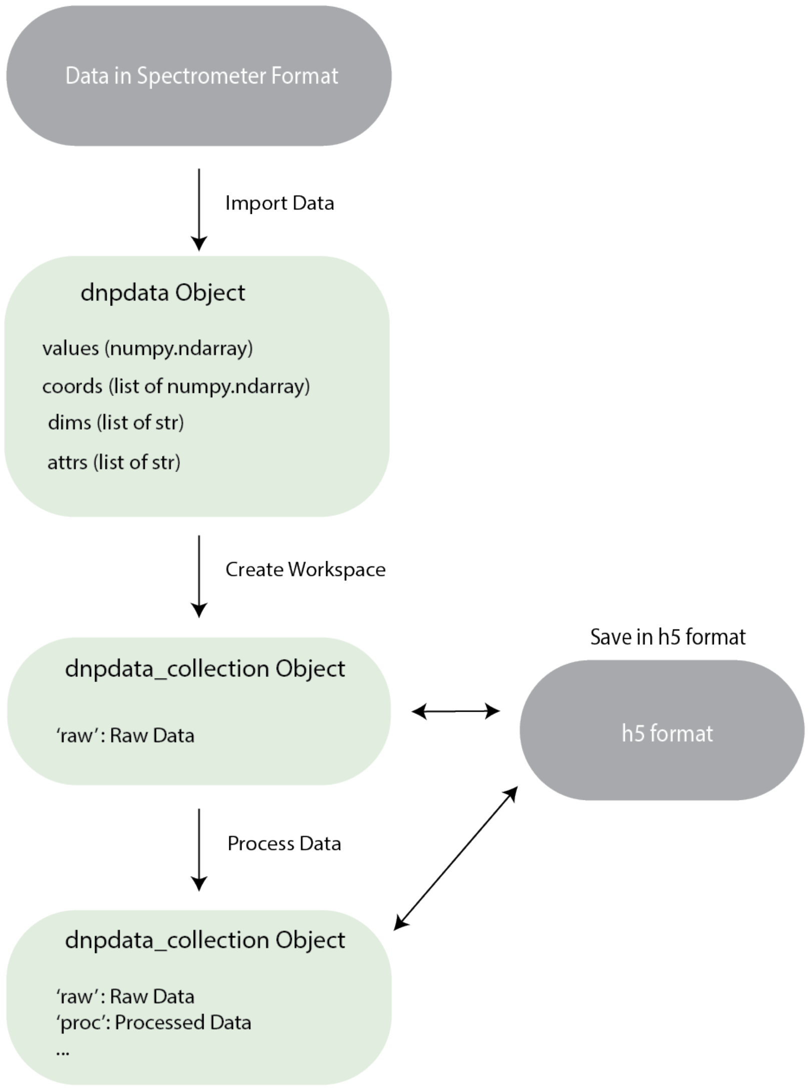


Fig. 1: Overview of the dnpLab Workflow

Importing Data

The data is imported using the *dnpImport* sub-package. This sub-package contians modules for importing various spectrometer formats (e.g. *Topspin*, *VnmrJ*, *Prospa*).

The data is imported as a *dnpdata* object. The dnpdata object is a container for data (values), coordinates for each dimension (coords), dimension labels (dims), and experimental parameters (attrs). In addition, each processing step applied to the data is saved in the dnpdata object (stored as proc_attrs).

The dnpdata object is a flexible data format which can handle N-dimensional data and coordinates together.

Creating a workspace

The workspace can be created with the "create_workspace" function in dnpLab. Once the data is imported, it is added to a workspace which is a python dictonary-like class that stores multiple dnpdata objects. A workspace is a collection of dnpdata objects and allows for raw and processed data to be saved in the same h5 file. That way, the raw data is always available, even if the data on the spectrometer does not exist anymore.

Creating a single h5 file has the advantage that data can be easily shared among collaborators.

Processing Data

The dnpLab workspace has the concept of a "processing_buffer" (typically called proc). The processing buffer specifies the data which is meant for processing. Typically one will add (raw) data to the workspace and copy or move the data to the processing buffer (proc). dnpLab is primarily designed for processing and analyzing DNP-NMR data. Processing DNP-NMR data is performed using the the *dnpNMR* module.

Saving Data in h5 format

Once the data is processed, the entire workspace can be saved in a single file in the h5 format. This is done using the *h5* module. The workspace can then be loaded, subsequent processing can be performed and the data can be saved again.

1.2 Installing dnpLab

1.2.1 Required Packages

The following packages are required to run dnpLab:

Package	Version
NumPy	1.19 or higher
SciPy	1.5 or higher
Matplotlib	3.3 or higher
h5py	2.10 or higher
PyQt5	5.15

1.2.2 Installing with pip

dnpLab can be installed using pip. In a terminal simply type the following command:

```
$ python -m pip install dnpLab
```

If you prefer to install dnpLab from the source code, check out our GitHub repository: .

1.3 Quick-Start Guide

1.3.1 Importing the Package

Once the package has been *installed via pip*, you should be able to import dnpLab in a python terminal/script.

```
import dnpLab as dnp
```

1.3.2 Importing data

```
# Import module
import dnpLab as dnp

# import Topspin Data
path = 'path/to/data'
data = dnpLab.dnpImport.topspin.import_topspin(path)

# create workspace for processing data
workspace = dnp.create_workspace('raw', data)
```

1.3.3 Processing NMR Data

```
# Remove DC offset from FID
workspace = dnpLab.dnpNMR.remove_offset(workspace, {})
# Apply Exponential Apodization to data
workspace = dnpLab.dnpNMR.window(workspace, {})
# Apply Fourier Transform to direct dimension by default (t2)
workspace = dnpLab.dnpNMR.fourier_transform(workspace, {})
```

1.3.4 Example Script

1.4 dnpLab Examples

dnpLab comes with many example scripts to demonstrate how the package can be used to import data from different spectrometer platform, process NMR data and extract enhancement data or hydration information. The example scripts are located in the *examples* folder using sample data located in the *data* folder.

If you installed dnpLab using pip you can download the example scripts and data from the GitHub repository:

Example Scripts: Example Scripts Example Data: Example Data

1.4.1 Import Data and Process FID (Bruker Format)

This example uses the example script: *example_process_1Dbruker.py*. The script demonstrates the following features of dnpLab:

1. Load a single FID (Bruker format)
2. Perform an offset correction
3. Apply apodization to the FID
4. Perform a Fourier transformation
5. Phase correct the resulting spectrum

If you installed dnpLab using pip. Otherwise, you have to specify the path to the package explicitly:

```
import numpy as np
import dnpLab as dnp
```

Note: If you downloaded dnpLab via GitHub and haven't installed, you must add the directory for dnpLab to the system path before importing dnpLab. Add the following lines to the beginning of the script:

```
import sys
sys.path.append('path/to/dnpLab/package')
```

In the next step load a single FID in Bruker format:

```
path = 'path/to/data/topspin/'
folder = 20

data = data.dnpImport.topspin.import_topspin(path,folder)
```

The topspin import module requires the path and the folder number. In the next step the workspace is set up and the imported data is added to the *raw* workspace and the same data is copied to the *proc* workspace.

```
ws = dnp.create_workspace()
ws.add('raw', data)
ws.copy('raw', 'proc')
```

Note: When working with dnpLab one of the first steps is to copy the imported data to the *raw* workspace. That way the raw data and all it's attributes will be always accessible to the user. When saving data with dnpLab the raw data is safed toegether with the processed data. dnpLab uses the h5 format to store data.

In the following steps, the FID is processed and the spectrum is plotted.

```
dnp.dnpNMR.remove_offset(ws,{})
dnp.dnpNMR.window(ws,{'linewidth' : 10})
dnp.dnpNMR.fourier_transform(ws,{'zero_fill_factor' : 2})
dnp.dnpNMR.autophase(ws,{})
```

In this example first a baseline correction is performed (dnpNMR.remove_offset) and apodization is applied ot the FID (dnpNMR.window). In this example a line broadening of 10 Hz is applied. The next step is to Fourier transform the FID (dnpNMR.fourier_transform) and phase the spectrum (dnpNMR.autophase).

To plot the NMR spectrum:

```
dnp.dnpResults.figure()
dnp.dnpResults.plot(ws['proc'].real)
dnp.dnpResults.xlim([-35,50])
dnp.dnpResults.plt.xlabel('Chemical Shift [ppm]')
dnp.dnpResults.plt.ylabel('Signal Amplitude [a.u.]')
dnp.dnpResults.show()
```

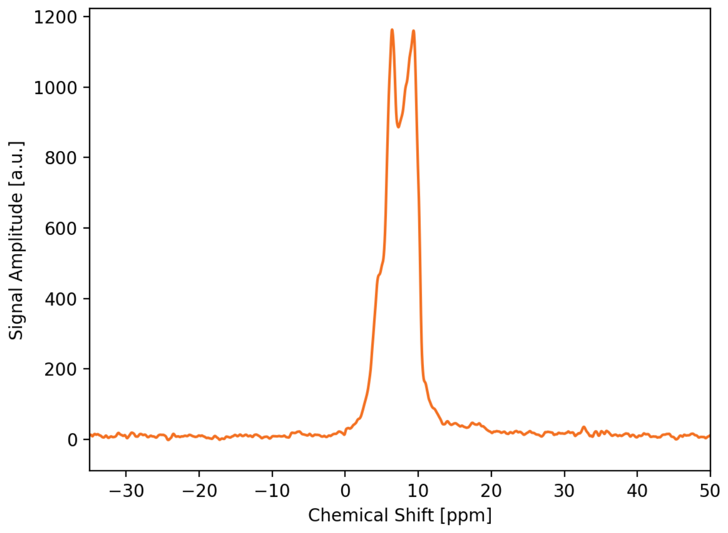


Fig. 2: 1D NMR Spectrum Imported in Bruker Format

Here only the real part of the spectrum is displayed (dnpResults.plot(ws['proc'].real)). The imaginary part of the spectrum can be displayed by changing the second line to

```
dnpResults.plot(ws['proc'].imag)
```

To display the unprocessed raw FID:

```
dnp.dnpResults.figure()
dnp.dnpResults.plot(ws['raw'].real)
dnp.dnpResults.plt.xlabel('t2 [s]')
dnp.dnpResults.plt.ylabel('Signal Amplitude [a.u.]')
dnp.dnpResults.show()
```

1.4.2 Determine T1 from an Inversion Recovery Experiment

In this example, the data from an inversion recovery experiment is analyzed to extract the longitudinal relaxation time T1 from the polarization build up. This example uses the example script: *example_process_IRbruker.py*.

First, import the experimental data (Bruker format) (if dnpLab is installed through pip, ignore the first two lines):

```
import sys
sys.path.append('path/to/dnpLab/package')

import numpy as np
import dnpLab as dnp
```

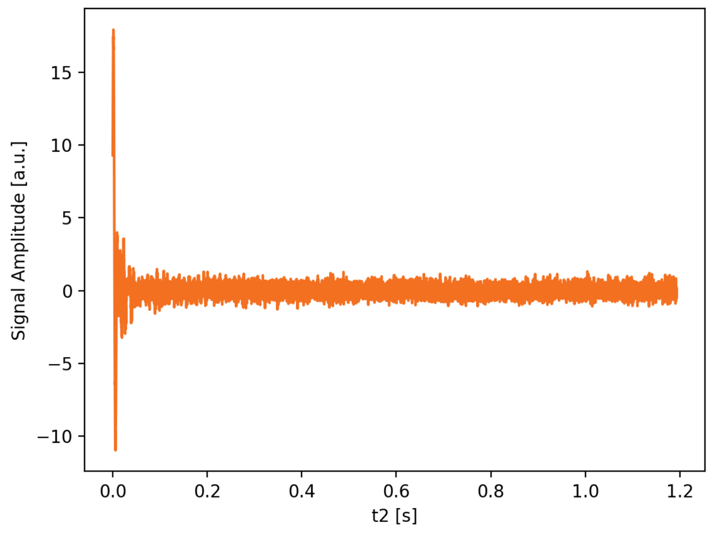


Fig. 3: 1D FID from raw data (Bruker Format)

In the next step load a single FID in Bruker format:

```
path = 'path/to/data/topspin/'
folder = 304

data = dnp.dnpImport.topspin.import_topspin(path,folder)
```

Next, create the workspace:

```
ws = dnp.create_workspace()
ws.add('raw', data)
ws.copy('raw', 'proc')
```

Next, process the FID, perform Fourier transformation, align and phase the NMR spectra:

```
dnp.dnpNMR.remove_offset(ws,{})
dnp.dnpNMR.window(ws,{'linewidth' : 10})
dnp.dnpNMR.fourier_transform(ws,{'zero_fill_factor' : 2})
dnp.dnpNMR.align(ws, {})
dnp.dnpNMR.autophase(ws,{})
```

To plot the processed NMR spectra:

```
dnp.dnpResults.plot(ws['ft'].real)
dnp.dnpResults.xlim([-30,50])
dnp.dnpResults.plt.xlabel('Chemical Shift [ppm]')
dnp.dnpResults.plt.ylabel('Signal Amplitude [a.u.]')
dnp.dnpResults.figure()
```

Next, the processed NMR spectra are copied to *ft* within the workspace, the signal amplitude for each NMR spectrum is integrated and the data is fitted to a function, describing inversion recovery polarization build-up.

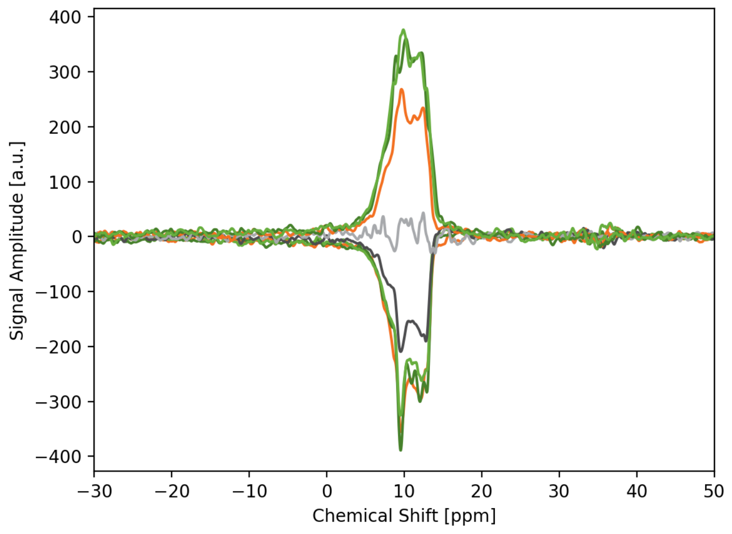


Fig. 4: Processed inversion recovery spectra (Bruker Format)

```
ws.copy('proc', 'ft')
dnp.dnpNMR.integrate(ws, {'integrate_width' : 100, 'integrate_center' : 0})
dnp.dnpFit.t1Fit(ws)
```

The T1 value can be displayed using:

```
print('T1 value (sec) = ' + str(ws['fit'].attrs['t1']))
T1 value (sec) = 2.045498109768188
```

To plot the inversion-recovery build-up curve (experimental and fitted data):

```
dnp.dnpResults.plot(ws['proc'].real, 'o')
dnp.dnpResults.plot(ws['fit'])
dnp.dnpResults.show()
```

1.5 dnpData

1.5.1 dnpdata Class Overview

The dnpdata class is a flexible data container for N-dimensional data. The dnpdata class stores data, axes, parameters and processing information in a single object.

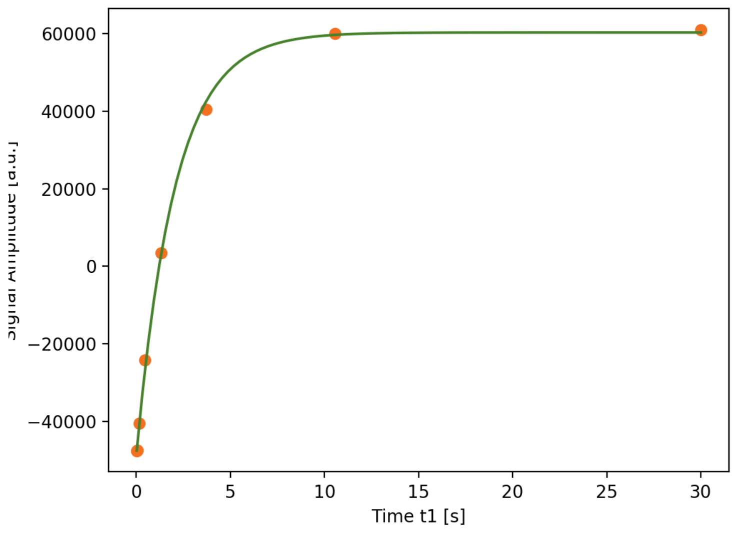


Fig. 5: Inversion recovery build-up (experimental and fit)

1.5.2 dnpdata Attributes

The attributes in the dnpdata object are named according to the convention by Pandas and xarray.

attribute	type	description
values	numpy.ndarray	Numpy array of data values
dims	list of str	Names for each of the N-dimensions in values
coords	list of numpy.ndarray	Axes values for each of the N-dimensions
attrs	dict	Dictionary of miscellaneous
proc_attrs	list of tuples (str, dict)	List which stores each processing step

1.5.3 dnpdata Examples

A dnpdata object can be defined as follows:

```python
import dnpLab as dnp
import numpy as np

x = np.r_[-10:10:100j]
values = x**2.
coords = [x]
dims = ['x']

data = dnp.dnpdata(values, coords, dims)

print(data)
```

The dnpdata class has a number of methods for manipulating the data.

The dimensions can be renamed:

```
data.rename('x', 't')
```

A N-dimensional data set can be reshaped by it's dimension labels. If a data set has

```
data.reorder(['z', 'x', 'y'])
```

1.5.4 Indexing

A number of methods can be used to index the data based on the coordinates.

```
data_slice = data['t', 0:10] # return first 10 points of data down 't' dimension

data_slice = data['t', (0.1, 0.5)] # return data in range from 0.1 to 0.5

data_slice = data['t', 0.1] # for single float, return data at index nearest to 0.1␣
↪in time coordinates

data_slice = data['t', 10] # for single int, return data at index 10
```

1.5.5 dnpdata Methods

class dnpLab.**dnpdata**(*values=array([], dtype=float64), coords=[], dims=[], attrs={}, procList=[]*)

Bases: `dnpLab.core.nddata.nddata_core`

dnpdata Class for handling dnp data

The dnpdata class is inspired by pyspecdata nddata object which handles n-dimensional data, axes, and other relevant information together.

This class is designed to handle data and axes together so that performing NMR processing can be performed easily.

Attributes: values (numpy.ndarray): Numpy Array containing data coords (list): List of numpy arrays containing axes of data dims (list): List of axes labels for data attrs (dict): Dictionary of parameters for data

add_proc_attrs(*proc_attr_name, proc_dict*)

Stamp processing step to dnpdata object

Parameters

- **proc_attr_name** (*str*) -- Name of processing step (e.g. "fourier_transform"
- **proc_dict** (*dict*) -- Dictionary of processing parameters for this processing step.

autophase()

Multiply dnpdata object by phase

phase()

Return phase of dnpdata object

Returns phase of data calculated from sum of imaginary divided by sum of real components

Return type phase (float,int)

squeeze()

Remove all length 1 dimensions from data

Warning: Axes information is lost

Example: data.squeeze()

`align`(*b*)

Align two data objects for numerical operations

Parameters **`b`** -- Ojbect to align with self

Returns self and b aligned data objects

Return type tuple

`argmax`(*dim*)

Return argmax for given dim

`argmin`(*dim*)

Return argmin for given dim

`chunk`(*dim*, *new_dims*, *new_sizes*)

Note: This is a placeholder for a function that's not yet implemented

Parameters

- **`dim`** (`str`) -- Assume that the dimension *dim* is a direct product of the dimensions given in *new_dims*, and chunk it out into those new dimensions.
- **`new_dims`** (`list of str`) -- The new dimensions to generate. Note that one of the elements of the list can be *dim* if you like.

  It's assumed that the ordering of *dim* is a direct product given in C-ordering (*i.e.* the inner dimensions are listed last and the outer dimensions are listed first -- here "inner" means that changes to the index of the inner-most dimension correspond to adjacent positions in memory and/or adjacent indeces in the original dimension that you are chunking)
- **`new_sizes`** (`list of int`) -- sizes of the new dimensions`

Returns **self** -- The new nddata object. Note that uniformly ascending or descending coordinates are manipulated in a rational way, *e.g.* *[1,2,3,4,5,6]* when chunked to a size of *[2,3]* will yield coordinates for the two new dimensions: *[1,4]* and *[0,1,2]*. Coordinates that are not uniformly ascending or descending will yield and error and must be manually modified by the user.

Return type nddata_core

`concatenate`(*b*, *dim*)

`copy`()

Return deepcopy of dnpdata object

Returns deep copy of data object

`get_coord`(*dim*)

Return coord corresponding to given dimension name

Parameters **`dim`** (`str`) -- Name of dim to retrieve coordinates from

Returns array of coordinates

Return type numpy.ndarray

`index`(*dim*)
Find index of given dimension name

`is_sorted`(*dim*)
Determine if coords corresponding to give dim are sorted in ascending order :param dim: Dimension to check if sorted :type dim: str

Returns True if sorted, False otherwise.

Return type bool

`merge_attrs`(*b*)
Merge the given dictionaries

Parameters **b** (*nddata_core*) -- attributes to merge into object

`new_dim`(*dim*, *coord*)

`rename`(*dim*, *new_name*)
Rename dim

Parameters

- **dim** (*str*) -- Name of dimension to rename
- **new_name** (*str*) -- New name for dim

`reorder`(*dims*)

`property size`
Returns values.size. Total number of elements in numpy array.

`smoosh`(*old_dims*, *new_name*)

Note: Not yet implemented.

smoosh does the opposite of *chunk* -- see :func`:~nddata_core.chunk`

`sort`(*dim*)
Sort the coords corresponding to the given dim in ascending order

Parameters **dim** (*str*) -- dimension to sort

`sort_dims`()
Sort the dimensions

`sum`(*dim*)
Perform sum down given dimension

1.5.6 dnpdata_collection (Workspace)

To store multiple data objects, the user can create a workspace which is a dict like object for storing dnpdata objects.

```
import dnpLab as dnp

ws = dnp.create_workspace()
ws['raw'] = data
```

1.5.7 Processing Buffer

The workspace has an attribute called processing_buffer. The processing buffer indicates for functions which operate on the workspace, which dnpdata object should be operated on. By default, the processing_buffer is called "proc".

```
ws.copy('raw', 'proc') # copy some data into the default processing buffer
```

At any time, the processing buffer can be changed, however you need to make sure to move data into the processing buffer before any processing steps.

```
ws.processing_buffer = 'new_proc'
```

1.5.8 Saving the Workspace

The workspace can be saved in h5 format with the saveh5 function:

```
dnplab.dnpImport.saveh5('test.h5', ws)
```

1.5.9 Loading the Workspace

A workspace can also be loaded with the loadh5 function.

```
dnplab.dnpImport.loadh5('test.h5')
```

1.5.10 dnpdata_collection Methods

class dnpLab.**dnpdata_collection**(**args, **kwargs*)

Bases: `collections.abc.MutableMapping`

Dictionary-like workspace object for storing dnpdata objects

copy(*key, new_key=None*)

Copy data from key to new_key. If new_key is not given, by default key will be copied to processing buffer

Parameters

- **key** (`str`) -- Key to be copied
- **new_key** (`str, None`) -- New key for copied data

move(*key, new_key*)

Move data from key to new_key

Parameters

- **key** (*str*) -- Name of data to move
- **new_key** (*str*) -- Name of new key to move data

pop(*key*)
Pop key. Removes data corresponding to key.

dict()
Return dictionary for storing data in dnpdata_collection

clear()
Removes all items

items()
Return items

keys()
Return keys.

popitem()
Pops item from end of dnpdata_collection

Returns key, item pair that was removed

Return type tuple

values()
Return Values

add(*key*, *data*)
Adds new data

Parameters

- **key** (*str*) -- key corresponding to new data
- **data** (dnpdata) -- data object corresponding to key

1.6 dnpImport

1.6.1 Topspin Module

dnpLab.dnpImport.topspin.**find_group_delay**(*decim*, *dspfvs*)
Determine group delay from tables

Parameters

- **decim** -- Decimation factor of the digital filter (factor by which oversampling rate exeeds sampling rate).
- **dspfvs** -- Firmware version for Bruker Console.

Returns Group delay. Number of points FID is shifted by DSP. The ceiling of this number (group delay rounded up) is the number of points should be removed from the start of the FID.

Return type float

dnpLab.dnpImport.topspin.**load_title**(*path*, *expNum=1*, *titlePath='pdata/1'*, *titleFilename='title'*)
Import Topspin Experiment Title File

Parameters

- **`path`** (*str*) -- Directory of title
- **`expNum`** (*int*) -- Experiment number to return title
- **`titlePath`** (*str*) -- Path within experiment of title
- **`titleFilename`** (*str*) -- filename of title

Returns Contents of experiment title file

Return type str

`dnpLab.dnpImport.topspin.`**`load_acqu`**(*path*, *expNum=1*, *paramFilename='acqus'*)

Import Topspin JCAMPDX file

Parameters

- **`path`** (*str*) -- directory of acqusition file
- **`expNum`** (*int*) -- Experiment number
- **`paramFilename`** (*str*) -- Acqusition parameters filename

Returns Dictionary of acqusition parameters

Return type dict

`dnpLab.dnpImport.topspin.`**`load_proc`**(*path*, *expNum=1*, *procNum=1*, *paramFilename='procs'*)

`dnpLab.dnpImport.topspin.`**`dir_data_type`**(*path*, *expNum*)

Determine type of data in directory

Parameters

- **`path`** (*str*) -- Directory of data
- **`expNum`** (*int*) -- Experiment number

Returns String identifying filetype

Return type str

`dnpLab.dnpImport.topspin.`**`import_topspin`**(*path*, *expNum*, *paramFilename='acqus'*)

Import topspin data and return dnpdata object

Parameters

- **`path`** (*str*) -- Directory of data
- **`expNum`** (*int*) -- Experiment number
- **`paramFilename`** (*str*) -- Parameters filename

Returns topspin data

Return type *dnpdata*

`dnpLab.dnpImport.topspin.`**`topspin_fid`**(*path*, *expNum*, *paramFilename='acqus'*)

Import topspin fid data and return dnpdata object

Parameters

- **`path`** (*str*) -- Directory of data
- **`expNum`** (*int*) -- Experiment number
- **`paramFilename`** (*str*) -- Parameters filename

Returns Topspin data

Return type *dnpdata*

`dnpLab.dnpImport.topspin.`**`topspin_jcamp_dx`**(*path*)

Return the contents of topspin JCAMP-DX file as dictionary

Parameters **`path`** -- Path to file

Returns Dictionary of JCAMP-DX file

Return type dict

`dnpLab.dnpImport.topspin.`**`topspin_vdlist`**(*path*, *expNum*)

Return topspin vdlist

Parameters

- **`Path`** (`str`) -- Directory of data
- **`expNum`** (`int`) -- Experiment number

Returns vdlist as numpy array

Return type numpy.ndarray

`dnpLab.dnpImport.topspin.`**`import_ser`**(*path*, *expNum*, *paramFilename='acqus'*)

Import topspin ser file

Parameters

- **`path`** (`str`) -- Directory of data
- **`expNum`** (`int`) -- Experiment number
- **`paramFilename`** (`str`) -- Filename of parameters file

Returns Topspin data

Return type *dnpdata*

`dnpLab.dnpImport.topspin.`**`topspin_ser_phase_cycle`**(*path*, *expNum*, *paramFilename='acqus'*)

Import Topspin data with phase cycle saved as different dimension

Parameters

- **`path`** (`str`) -- Directory of data
- **`expNum`** (`int`) -- Experiment number
- **`paramFilename`** (`str`) -- Filename of parameters file

Returns Topspin data

Return type *dnpdata*

`dnpLab.dnpImport.topspin.`**`import_topspin_dir`**(*path*)

Import directory of Topspin data and return as dictionary

Parameters **`path`** (`str`) -- Directory of data

Returns Topspin data. Keys correspond to folder name. Values correspond to dnpdata with topspin data for each folder.

Return type dict

1.6.2 (Open) VnmrJ Module

`dnpLab.dnpImport.vnmrj.`**`array_coords`**(*attrs*)
Return array dimension coords from parameters dictionary

Parameters **`attrs`** (*`dict`*) -- Dictionary of procpar parameters

Returns dim and coord for array

Return type tuple

`dnpLab.dnpImport.vnmrj.`**`import_fid`**(*path*, *filename='fid'*)
Import VnmrJ fid file

Parameters

- **`path`** (*`str`*) -- Directory of fid file
- **`filename`** (*`str`*) -- Name of fid file. "fid" by default

Returns Array of data

Return type numpy.ndarray

`dnpLab.dnpImport.vnmrj.`**`import_procpar`**(*path*, *filename='procpar'*)
Import VnmrJ procpar parameters file

Parameters **`path`** (*`str`*) -- Directory of file

Returns Dictionary of procpar parameters

Return type dict

`dnpLab.dnpImport.vnmrj.`**`import_vnmrj`**(*path*, *fidFilename='fid'*, *paramFilename='procpar'*)
Import VnmrJ Data

Parameters

- **`path`** (*`str`*) -- path to experiment folder
- **`fidFilename`** (*`str`*) -- FID file name
- **`paramFilename`** (*`str`*) -- process parameter filename

Returns data in dnpdata object

Return type *dnpdata*

1.6.3 Prospa Module

`dnpLab.dnpImport.prospa.`**`import_prospa`**(*path*, *parameters_filename=None*, *verbose=False*)
Import Kea data

Parameters

- **`path`** (*`str`*) -- Path to data
- **`num`** (*`int`*) -- Experiment number
- **`verbose`** (*`bool`*) -- If true, prints additional information for troubleshooting

Returns dnpdata object with Kea data

`dnpLab.dnpImport.prospa.`**`import_prospa_dir`**(*path*, *exp_list=None*)
Import directory of prospa experiments

`dnpLab.dnpImport.prospa.`**`import_nd`**(*path*)

Import Kea 1d, 2d, 3d, 4d files

Parameters **`path`** (*`str`*) -- Path to file

Returns x (None, numpy.array): Axes if included in binary file, None otherwise data (numpy.array): Numpy array of data

Return type tuple

`dnpLab.dnpImport.prospa.`**`import_par`**(*path*)

Import Kea parameters .par file

Parameters **`path`** (*`str`*) -- Path to parameters file

Returns Dictionary of Kea Parameters

Return type dict

`dnpLab.dnpImport.prospa.`**`import_csv`**(*path*, *return_raw=False*, *is_complex=True*)

Import Kea csv file

Parameters **`path`** (*`str`*) -- Path to csv file

Returns x(numpy.array): axes if return_raw = False data(numpy.array): Data in csv file

Return type tuple

1.6.4 h5 Module

`dnpLab.dnpImport.h5.`**`saveh5`**(*dataDict*, *path*, *overwrite=False*)

Save workspace in .h5 format

Parameters

- **`dataDict`** (`dnpdata_collection`) -- dnpdata_collection object to save.
- **`path`** (*`str`*) -- Path to save data
- **`overwrite`** (*`bool`*) -- If True, h5 file can be overwritten. Otherwise, h5 file cannot be overwritten

`dnpLab.dnpImport.h5.`**`write_dnpdata`**(*dnpDataGroup*, *dnpDataObject*)

Takes file/group and writes dnpData object to it

Parameters

- **`dnpDataGroup`** -- h5 group to save data to
- **`dnpDataObject`** -- dnpdata object to save in h5 format

`dnpLab.dnpImport.h5.`**`write_dict`**(*dnpDataGroup*, *dnpDataObject*)

Writes dictionary to h5 file

`dnpLab.dnpImport.h5.`**`loadh5`**(*path*)

Returns Dictionary of dnpDataObjects

Parameters **`path`** (*`str`*) -- Path to h5 file

Returns workspace object with data

Return type *dnpdata_collection*

1.7 dnpNMR

1.7.1 Summary

The following table summarizes all available functions in this module

1.7.2 Detailed Description of Functions

`dnpLab.dnpNMR.`**`update_parameters`**(*proc_parameters*, *requiredList*, *default_parameters*)

Add default parameter to processing parameters if a processing parameter is missing

Parameters

- **`proc_parameters`** (`dict`) -- Dictionary of initial processing parameters
- **`requiredList`** (`list`) -- List of requrired processing parameters
- **`default_parameters`** (`dict`) -- Dictionary of default processing parameters

Returns Updated processing parameters dictionary

Return type dict

`dnpLab.dnpNMR.`**`remove_offset`**(*all_data*, *proc_parameters*)

Remove DC offset from FID by averaging the last few data points and subtracting the average

Parameters

- **`all_data`** (`dnpdata, dict`) -- Data container for data
- **`proc_parameters`** (`dict,procParam`) -- Processing _parameters

parameter	type	default	description
dim	str	't2'	Dimension to calculate DC offset
offset_points	int	10	Number of points at end of data to average for DC offset

Returns If workspace is given returns dnpdata_collection with data in processing buffer updated dnpdata: If dnpdata object is given, return dnpdata object.

Return type *dnpdata_collection*

Example:

```
proc_parameters = {}
proc_parameters['dim'] = 't2'
proc_parameters['offset_points'] = 10

workspace = dnpLab.dnpNMR.remove_offset(workspace, proc_parameters)
```

`dnpLab.dnpNMR.`**`fourier_transform`**(*all_data*, *proc_parameters*)

Perform Fourier Transform down dim dimension given in proc_parameters

Note: Assumes dt = t[1] - t[0]

Parameters

- **`all_data`** (`dnpdata, dict`) -- Data container

- **proc_parameters** (*dict, procParam*) -- Processing parameters

parameter	type	default	description
dim	str	't2'	dimension to Fourier transform
zero_fill_factor	int	2	factor to increase dim with zeros
shift	bool	True	Perform fftshift to set zero frequency to center
convert_to_ppm	bool	True	Convert dim from Hz to ppm

Returns Processed data in container

Return type all_data (*dnpdata*, dict)

Example:

```
proc_parameters['dim'] = 't2'
proc_parameters['zero_fill_factor'] = 2
proc_parameters['shift'] = True
proc_parameters['convert_to_ppm'] = True

all_data = dnpLab.dnpNMR.fourier_transform(all_data, proc_parameters)
```

dnpLab.dnpNMR.**window**(*all_data, proc_parameters*)

Apply Apodization to data down given dimension

Parameters

- **all_data** (*dnpdata, dict*) -- data container
- **proc_parameters** (*dict, procParam*) -- parameter values

Note: Axis units assumed to be seconds

parameter	type	default	description
dim	str	't2'	Dimension to apply exponential apodization
linewidth	float	10	Linewidth of broadening to apply in Hz

Returns data object with window function applied

Return type *dnpdata_collection* or *dnpdata*

Example:

```
proc_parameters = {
        'linewidth' : 10,
        'dim' : 't2',
        }
all_data = dnpLab.dnpNMR.window(all_data,proc_parameters)
```

dnpLab.dnpNMR.**integrate**(*all_data, proc_parameters*)

Integrate data down given dimension

Parameters

- **all_data** (*dnpdata,dict*) -- Data container
- **proc_parameters** (*dict, procParam*) -- Processing Parameters

parameter	type	default	description
dim	str	't2'	dimension to integrate
integrate_center	float	0	center of integration window
integrate_width	float	100	width of integration window

Returns Processed data

Return type all_data (*dnpdata*,dict)

Example:

```
proc_parameters = {
    'dim' : 't2',
    'integrate_center' : 0,
    'integrate_width' : 100,
    }
dnpLab.dnpNMR.integrate(all_data,proc_parameters)
```

`dnpLab.dnpNMR.`**`align`**(*all_data*, *proc_parameters*)

Alignment of NMR spectra down given dim dimension

Example:

```
data = dnp.dnpNMR.align(data, {})
```

`dnpLab.dnpNMR.`**`autophase`**(*workspace*, *parameters*)

Automatically phase data

Parameters

- **`workspace`** (`dnpdata_collection, dnpdata`) -- Data object to autophase
- **`parameters`** (*`dict`*) --

Returns Autophased data

Return type *dnpdata_collection*, *dnpdata*

Example:

```
ws = dnp.dnpNMR.autophase(ws, {})
```

1.8 dnpFit

1.8.1 Summary

The following table summarizes all available functions in this module

1.8.2 Detailed Description of Functions

`dnpLab.dnpFit.`**`t1Fit`**(*dataDict*)

Fits inversion recovery data to extract T1 value in seconds

$$f(t) = M_0 - M_\infty e^{-t/T_1}$$

Parameters **`after processing inversion recovery data, after integration with dnpNMR.integrate`**(*`workspace`*) --

Returns Processed data in container, updated with fit data attributes: T1 value and T1 standard deviation

Return type all_data (*dnpdata*, dict)

Example:

```
### INSERT importing and processing ###
dnpLab.dnpNMR.integrate(workspace, {})

dnpLab.dnpFit.t1Fit(workspace)

T1_value = workspace['fit'].attrs['t1']
T1_standard_deviation = workspace['fit'].attrs['t1_stdd']
T1_fit = workspace['fit'].values
T1_fit_xaxis = workspace['fit'].coords
```

`dnpLab.dnpFit.`**`enhancementFit`**(*dataDict*)

Fits enhancement curves to return Emax and power and one half maximum saturation

$$f(p) = E_{max} p/(p_{1/2} + p)$$

Parameters **`workspace`** --

Returns

Processed data in container, updated with fit data attributes: Emax value and Emax standard deviation

p_one_half value and p_one_half standard deviation

Return type all_data (*dnpdata*, dict)

Example:

```
### INSERT importing and processing ###
dnpLab.dnpNMR.integrate(workspace, {})

workspace.new_dim('power', power_list)

dnpLab.dnpFit.enhancementFit(workspace)

Emax_value = workspace['fit'].attrs['E_max']
Emax_standard_deviation = workspace['fit'].attrs['E_max_stdd']
p_one_half_value = workspace['fit'].attrs['p_half']
p_one_half_standard_deviation = workspace['fit'].attrs['p_half_stdd']
Emax_fit = workspace['fit'].values
Emax_fit_xaxis = workspace['fit'].coords
```

1.9 dnpHydration

1.9.1 Summary

The following table summarizes all available functions in this module

1.9.2 Detailed Description of Functions

dnpHydration module

This module calculates hydration related quantities using processed ODNP data.

`dnpLab.dnpHydration.`**`hydration`**`(`*ws*`)`

Calculating Hydration Results

Parameters **`ws`** -- Workspace

Returns A dictionary of hydration results

Return type dict

`class` `dnpLab.dnpHydration.`**`HydrationParameter`**

Hydration Parameters

Franck, JM, et. al.; "Anomalously Rapid Hydration Water Diffusion Dynamics Near DNA Surfaces" J. Am. Chem. Soc. 2015, 137, 1201312023.

`ksigma_bulk = 95.4`

unit is s^-1 M^-1 (Figure 3 caption)

Type float

`klow_bulk = 366`

unit is s^-1 M^-1 (Figure 3 caption)

Type float

`tcorr_bulk = 54`

Corrected bulk tcorr, unit is ps, (section 2.5)

Type float

`D_H2O = 2.3e-09`

(Eq. 19-20) bulk water diffusivity, unit is d^2/s where d is distance in meters.

Type float

`D_SL = 4.1e-10`

(Eq. 19-20) spin label diffusivity, unit is d^2/s where d is distance in meters.

Type float

`field = None`

Static magnetic field in mT, needed to find omega_e and _H

Type float

`spin_C = None`

(Eq. 1-2) unit is microM, spin label concentration for scaling relaxations to get "relaxivities"

Type float

T10 = None
T1 with spin label but at 0 mw E_power, unit is sec

Type float

T100 = None
T1 without spin label and without mw E_power, unit is sec

Type float

property t1_interp_method
Method used to interpolate T1, either *linear* or *second_order*

Type str

property smax_model
Method used to determine smax. Either *tethered* or *free*

Type str

class dnpLab.dnpHydration.**HydrationCalculator**(*T1: numpy.array*, *T1_power: numpy.array*, *E: numpy.array*, *E_power: numpy.array*, *hp:* dnpLab.dnpHydration.HydrationParameter)

Bases: object

Hydration Results Calculator

T1
T1 array. Unit: second.

Type numpy.array

T1_power
E_power in Watt unit, same length as T1.

Type numpy.array

E
Enhancements.

Type numpy.array

E_power
E_power in Watt unit, same length as E.

Type numpy.array

hp
Parameters for calculation, including default values.

Type *HydrationParameter*

results
Hydration results.

Type *HydrationResults*

run()
Run calculator

interpolate_T1(*E_power: numpy.array*, *T1_power: numpy.array*, *T1: numpy.array*)
Returns the one-dimensional piecewise interpolant to a function with given discrete data points (T1_power, T1), evaluated at E_power.

Points outside the data range will be extrapolated

Parameters

- **`E_power`** -- The x-coordinates at which to evaluate.
- **`T1_power`** -- The x-coordinates of the data points, must be increasing. Otherwise, T1_power is internally sorted.
- **`T1`** -- The y-coordinates of the data points, same length as T1_power.

Returns The evaluated values, same shape as E_power.

Return type interplatedT1 (np.array)

`static get_tcorr`(*coupling_factor: float*, *omega_e: float*, *omega_H: float*)
Returns correlation time tcorr in pico second

Parameters

- **`coupling_factor`** (*`float`*) --
- **`omega_e`** (*`float`*) --
- **`omega_H`** (*`float`*) --

Returns correlation time in pico second

Return type float

Raises ***`FitError`*** -- If no available root is found.

`static get_ksigma`(*ksig_sp: numpy.array*, *power: numpy.array*)
Get ksigma and E_power at half max of ksig

Parameters

- **`ksig`** (*`numpy.array`*) -- Array of ksigma.
- **`power`** (*`numpy.array`*) -- Array of E_power.

Returns fit results pcov: covariance matrix

Return type popt

Asserts: ksigma (popt[0]) is greater than zero

`static get_uncorrected_xi`(*Ep: numpy.array*, *power: numpy.array*, *T10: float*, *T100: float*, *wRatio: float*, *s_max: float*)
Get coupling_factor and E_power at half saturation

Parameters

- **`Ep`** (*`numpy.array`*) -- Array of enhancements.
- **`power`** (*`numpy.array`*) -- Array of E_power.
- **`T10`** (*`float`*) -- T10
- **`T100`** (*`float`*) -- T100
- **`wRatio`** (*`float`*) -- ratio of electron & proton Larmor frequencies
- **`s_max`** (*`float`*) -- maximal saturation factor

Returns A tuple of float (coupling_factor, p_12).

Raises ***`FitError`*** -- If least square fitting is not succeed.

class dnpLab.dnpHydration.**HydrationResults**(**args*, ***kwargs*)

Bases: *dnpLab.dnpHydration.AttrDict*

Class for handling hydration related quantities

uncorrected_Ep
Fit of Ep array

Type numpy.array

interpolated_T1
T1 values interpolated on E_power,

Type numpy.array

ksigma_array
numpy array that is the result of ~(1-E) / [(constants*T1)], used in ksigma(E_power) fit,

Type numpy.array

ksigma_fit
ksig_fit,

Type numpy.array

ksigma
ksigma,

Type float

ksigma_stdd
ksigma_stdd,

Type float

ksigma_bulk_ratio
ksigma/ksigma_bulk,

Type float

krho
krho,

Type float

klow
klow,

Type float

klow_bulk_ratio
klow / klow_bulk,

Type float

coupling_factor
coupling_factor,

Type float

tcorr
tcorr,

Type float

tcorr_bulk_ratio
tcorr / tcorr_bulk,

Type float

`Dlocal`
Dlocal

Type float

class `dnpLab.dnpHydration.`**`FitError`**
Bases: `Exception`

Exception of Failed Fitting

class `dnpLab.dnpHydration.`**`AttrDict`**(**args*, ***kwargs*)
Bases: `object`

Class with Dictionary-like Setting and Getting

`update`(*init=None*, ***kwargs*)
Update existing parameters

Parameters **`init`** -- If init is present and has a .keys() method, then does: for k in init: D[k] = E[k]. If init is present and lacks a .keys() method, then does: for k, v in init: D[k] = v

1.10 dnpResults

1.10.1 Summary

The following table summarizes all available functions in this module

1.10.2 Detailed Description of Functions

`dnpLab.dnpResults.`**`imshow`**(*data*, **args*, ***kwargs*)
Image Plot for dnpdata object

Parameters

- **`data`** (`dnpdata`) -- dnpdata object for image plot
- **`args`** -- args for matplotlib imshow function
- **`kwargs`** -- kwargs for matplotlib imshow function

Example:

```
# Plotting a dnpdata object
dnp.dnpResults.plt.figure()
dnp.dnpResults.imshow(data)
dnp.dnpResults.plt.show()

# Plotting a workspace (dnpdata_collection)
dnp.dnpResults.plt.figure()
dnp.dnpResults.imshow(ws['proc'])
dnp.dnpResults.plt.show()
```

`dnpLab.dnpResults.`**`plot`**(*data*, **args*, ***kwargs*)
Plot function for dnpdata object

Parameters

- **`data`** (`dnpdata`) -- dnpdata object for matplotlib plot function

- **args** -- args for matplotlib plot function
- **kwargs** -- kwargs for matplotlib plot function

Example:

```
# Plotting a dnpdata object
dnp.dnpResults.plt.figure()
dnp.dnpResults.plot(data)
dnp.dnpResults.plt.show()

# Plotting a workspace (dnpdata_collection)
dnp.dnpResults.plt.figure()
dnp.dnpResults.plot(ws['proc'])
dnp.dnpResults.plt.show()

# Plotting two curves on the same figure
dnp.dnpResults.plt.figure()
dnp.dnpResults.plot(ws['proc1'])
dnp.dnpResults.plot(ws['proc2'])
dnp.dnpResults.plt.show()

# Plotting with some custom parameters
dnp.dnpResults.plt.figure()
dnp.dnpResults.plot(ws['proc'], 'k-', linewidth = 3.0, alpha = 0.5)
dnp.dnpResults.plt.show()
```

1.11 hydrationGUI

Type hydrationGUI at the command line to open an interactive tool for processing ODNP data and calculating hydration parameters. All data processing and calculating is done using buttons, checkboxes, sliders, and edit fields.

Type the command to start the hydrationGUI:

```
> hydrationGUI
```

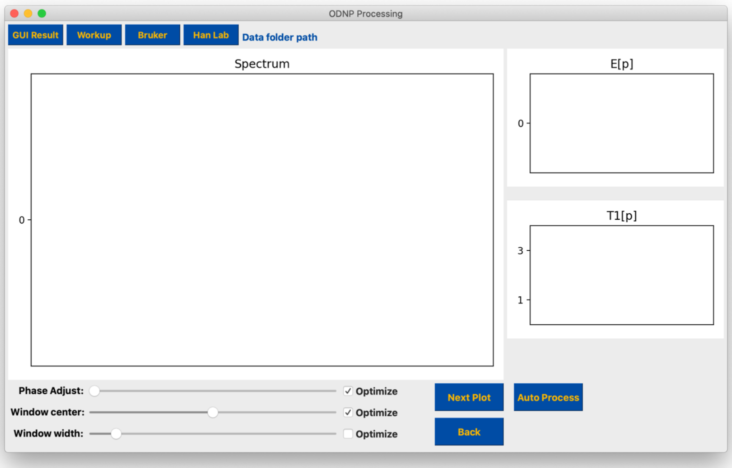
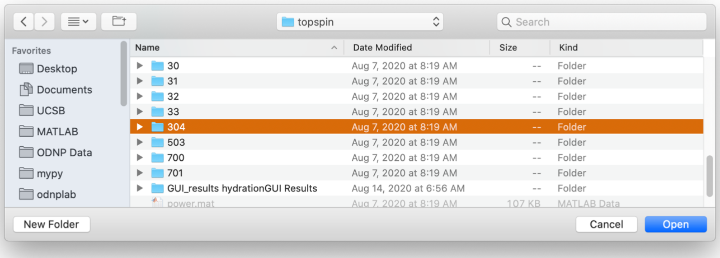


Fig. 6: hydrationGUI

1.11.1 Processing a single topspin data folder, 1D spectrum or 2D inversion recovery data

To work on a single topspin spectrum use the Bruker button to select a numbered folder containing a single spectrum, either 1D or 2D. You may make adjustments to the data phase and integration window center using the sliders. Use the “Optimize” checkboxes to search for and apply the “optimal” parameters.



Name	Date Modified	Size	Kind
30	Aug 7, 2020 at 8:19 AM	--	Folder
31	Aug 7, 2020 at 8:19 AM	--	Folder
32	Aug 7, 2020 at 8:19 AM	--	Folder
33	Aug 7, 2020 at 8:19 AM	--	Folder
304	Aug 7, 2020 at 8:19 AM	--	Folder
503	Aug 7, 2020 at 8:19 AM	--	Folder
700	Aug 7, 2020 at 8:19 AM	--	Folder
701	Aug 7, 2020 at 8:19 AM	--	Folder
GUI_results hydrationGUI Results	Aug 14, 2020 at 6:56 AM	--	Folder
power.mat	Aug 7, 2020 at 8:19 AM	107 KB	MATLAB Data

Fig. 7: Importing 1d or 2d data

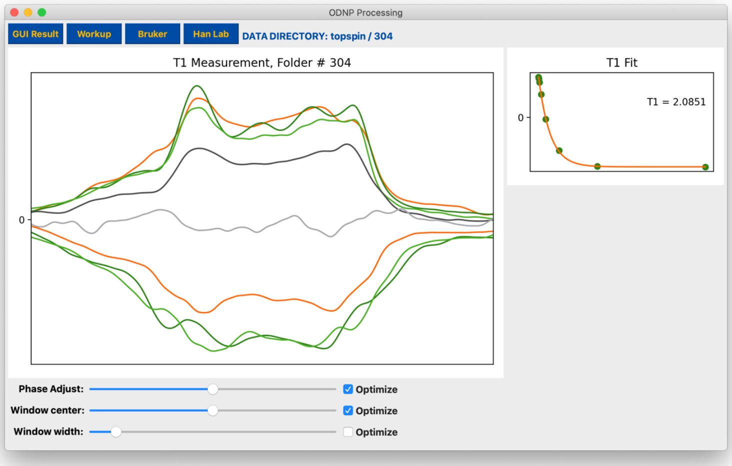


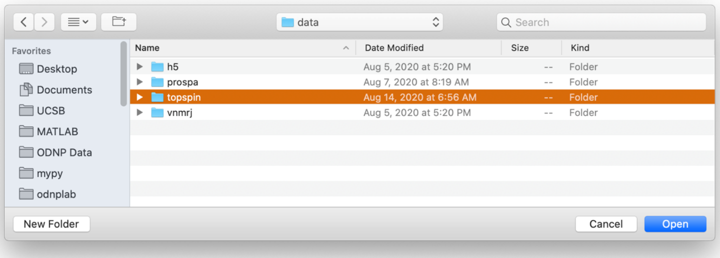
Fig. 8: Processing T1 experiment

1.11.2 Processing Han lab datasets

To load a dataset collected in the CNSI facility at University of California Santa Barbara using the ‘rb_dnp1’ command, use the Han Lab button and select the base folder.

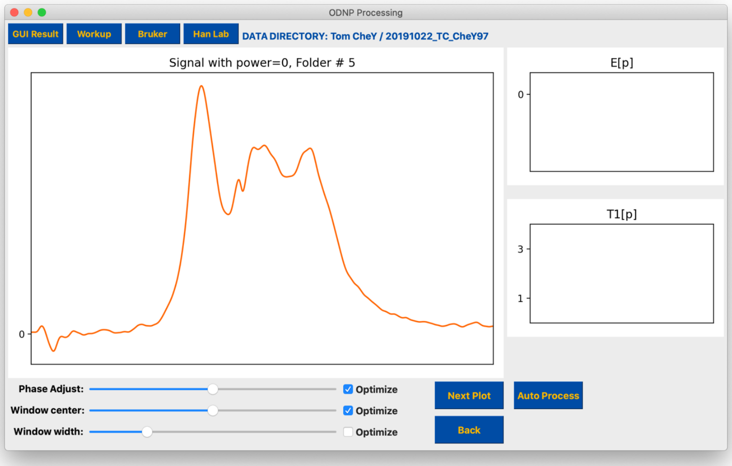
The title of the main plot will let you know which folder you are currently working on. Use the Next button to advance through the dataset towards calculating hydration parameters, and the Back button to regress through the dataset. Auto Process will run through the entire dataset automatically and calculate hydration parameters.

You may make adjustments to the data phase, integration window width, and integration window center using the sliders. Use the “Optimize” checkboxes to search for and apply the “optimal” parameters. For optimizing the width, checking Optimize selects the window that encompasses roughly 2/3 of the peak while unchecking selects the default width. If processing an ODNP dataset the width that is displayed in the plot will be used if the Next or Auto Process buttons are pressed.



Name	Date Modified	Size	Kind
h5	Aug 5, 2020 at 5:20 PM	--	Folder
prospa	Aug 7, 2020 at 8:19 AM	--	Folder
topspin	Aug 14, 2020 at 6:56 AM	--	Folder
vnmrj	Aug 5, 2020 at 5:20 PM	--	Folder

Fig. 9: Importing "rb_dnp1" experiment



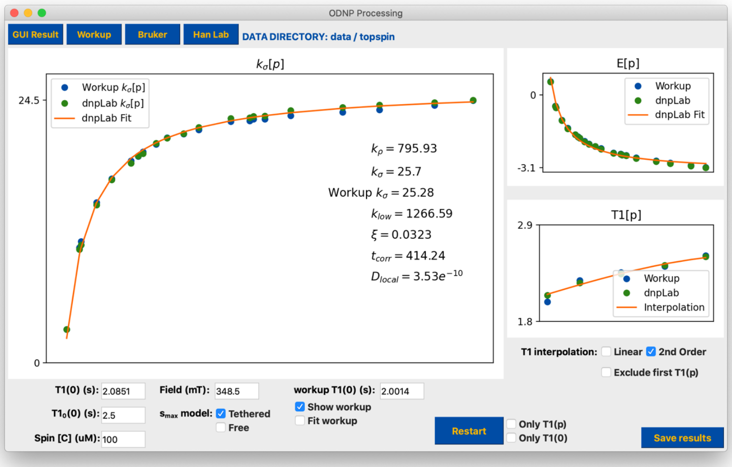
DATA DIRECTORY: Tom CheY / 20191022_TC_CheY97

Phase Adjust: [✓ Optimize]
Window center: [✓ Optimize]
Window width: Optimize

Fig. 10: Advance through the individual datasets to process the data

The results are displayed when finished. If a "Workup" is also present in the data folder it will be imported for comparison. Use the corresponding checkboxes to interact with the Workup results. Interaction with any parameter edit field or checkbox, as well as the T1 interpolation checkboxes, automatically updates the calculations.

The title of the main plot will let you know which folder you are currently working on. Use the Next button to advance through the dataset towards calculating hydration parameters, and the Back button to regress through the dataset. Auto Process will run through the entire dataset automatically and calculate hydration parameters.



DATA DIRECTORY: data / topspin

$k_\rho = 795.93$
$k_\sigma = 25.7$
Workup $k_\sigma = 25.28$
$k_{low} = 1266.59$
$\xi = 0.0323$
$t_{corr} = 414.24$
$D_{local} = 3.53e^{-10}$

T1 interpolation: [ ] Linear [✓ 2nd Order]
[ ] Exclude first T1(p)

T1(0) (s): 2.0851
$T1_0(0)$ (s): 2.5
Spin [C] (uM): 100
Field (mT): 348.5
s_{max} model: [✓ Tethered] [ ] Free
workup T1(0) (s): 2.0014
[✓ Show workup]
[ ] Fit workup
[ ] Only T1(p)
[ ] Only T1(0)

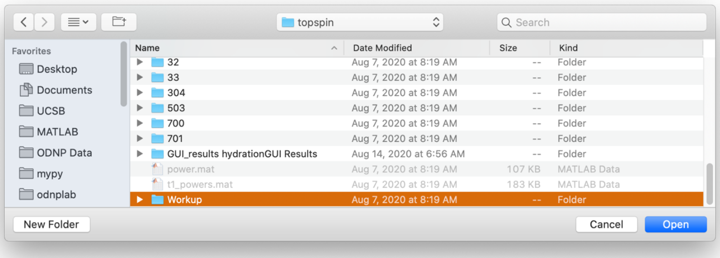
Fig. 11: Hydration Results

The results are displayed when finished. If a "Workup" is also present in the data folder it will be imported for comparison. Use the corresponding checkboxes to interact with the Workup results. Interaction with any parameter edit field or checkbox, as well as the T1 interpolation checkboxes, automatically updates the calculations.

The Restart button will return you to the beginning of processing. If the Only T1(0) checkbox is selected, Restart will return you to the final folder that is the T1(0) measurement while all other processing will be retained. If the Only T1(p) is selected you will return to the beginning of the series of T1 measurements and previous processing of the enhancement points is retained.

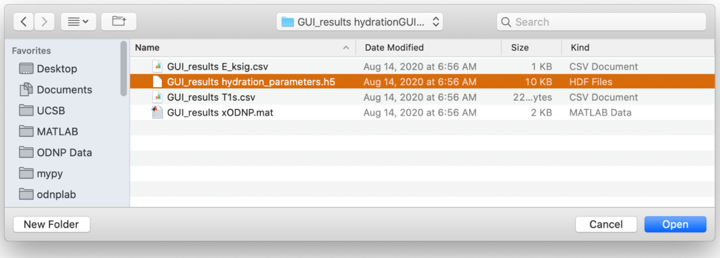
1.11.3 Analyzing previous GUI results Workup results

You may also load only the results of "Workup" code processing with the Workup button, or you may select the .mat or .h5 files of a previously saved session with the GUI Result button.



Name	Date Modified	Size	Kind
32	Aug 7, 2020 at 8:19 AM	--	Folder
33	Aug 7, 2020 at 8:19 AM	--	Folder
304	Aug 7, 2020 at 8:19 AM	--	Folder
503	Aug 7, 2020 at 8:19 AM	--	Folder
700	Aug 7, 2020 at 8:19 AM	--	Folder
701	Aug 7, 2020 at 8:19 AM	--	Folder
GUI_results hydrationGUI Results	Aug 14, 2020 at 6:56 AM	--	Folder
power.mat	Aug 7, 2020 at 8:19 AM	107 KB	MATLAB Data
t1_powers.mat	Aug 7, 2020 at 8:19 AM	183 KB	MATLAB Data
Workup	Aug 7, 2020 at 8:19 AM	--	Folder

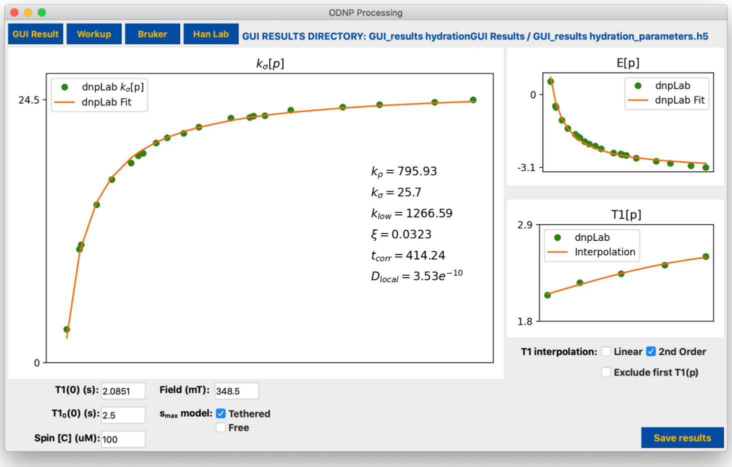
Fig. 12: Hydration Results from workup



Name	Date Modified	Size	Kind
GUI_results E_ksig.csv	Aug 14, 2020 at 6:56 AM	1 KB	CSV Document
GUI_results hydration_parameters.h5	Aug 14, 2020 at 6:56 AM	10 KB	HDF Files
GUI_results T1s.csv	Aug 14, 2020 at 6:56 AM	22...ytes	CSV Document
GUI_results xODNP.mat	Aug 14, 2020 at 6:56 AM	2 KB	MATLAB Data

Fig. 13: Hydration Results from h5

The results of previous processing will be used to calculate hydration parameters.



GUI RESULTS DIRECTORY: GUI_results hydrationGUI Results / GUI_results hydration_parameters.h5

$k_\rho = 795.93$
$k_\sigma = 25.7$
$k_{low} = 1266.59$
$\xi = 0.0323$
$t_{corr} = 414.24$
$D_{local} = 3.53e^{-10}$

T1 interpolation: Linear [✓ 2nd Order]
Exclude first T1(p)

T1(0) (s): 2.0851
T1$_0$(0) (s): 2.5
Spin [C] (uM): 100
Field (mT): 348.5
s_{max} model: [✓ Tethered] Free

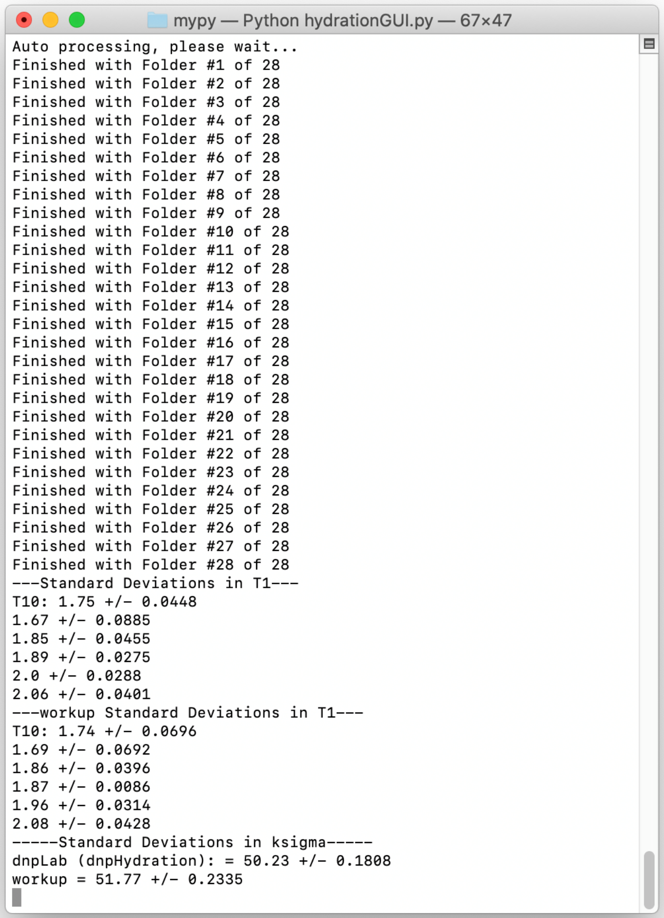
Fig. 14: Imported results from h5 file

1.11.4 Terminal outputs

The terminal will display processing and calculation progress as well as standard deviations of the T1 fits and , including the imported if a Workup was found.

1.11.5 Saving Results

After processing is complete and hydration parameters are calculated, the Save results button is available. Your results are saved in .csv, .h5, and .mat formats. The .mat file can be read by the MATLAB app called xODNP that is available at MathWorks File Exchange. The .h5 and .mat files can be read by hydrationGUI.



```
Auto processing, please wait...
Finished with Folder #1 of 28
Finished with Folder #2 of 28
Finished with Folder #3 of 28
Finished with Folder #4 of 28
Finished with Folder #5 of 28
Finished with Folder #6 of 28
Finished with Folder #7 of 28
Finished with Folder #8 of 28
Finished with Folder #9 of 28
Finished with Folder #10 of 28
Finished with Folder #11 of 28
Finished with Folder #12 of 28
Finished with Folder #13 of 28
Finished with Folder #14 of 28
Finished with Folder #15 of 28
Finished with Folder #16 of 28
Finished with Folder #17 of 28
Finished with Folder #18 of 28
Finished with Folder #19 of 28
Finished with Folder #20 of 28
Finished with Folder #21 of 28
Finished with Folder #22 of 28
Finished with Folder #23 of 28
Finished with Folder #24 of 28
Finished with Folder #25 of 28
Finished with Folder #26 of 28
Finished with Folder #27 of 28
Finished with Folder #28 of 28
---Standard Deviations in T1---
T10: 1.75 +/- 0.0448
1.67 +/- 0.0885
1.85 +/- 0.0455
1.89 +/- 0.0275
2.0 +/- 0.0288
2.06 +/- 0.0401
---workup Standard Deviations in T1---
T10: 1.74 +/- 0.0696
1.69 +/- 0.0692
1.86 +/- 0.0396
1.87 +/- 0.0086
1.96 +/- 0.0314
2.08 +/- 0.0428
-----Standard Deviations in ksigma-----
dnpLab (dnpHydration): = 50.23 +/- 0.1808
workup = 51.77 +/- 0.2335
```

Fig. 15: Terminal Output from processing

CHAPTER 2

Index

- genindex
- modindex
- search

CHAPTER 3

Acknowledgements

Development of dnpLap is sponsored by grants by the United States and the Germany. In particular:

- acknowledges support from the U.S. National Institutes of Health (NIH), grant .
- The acknowledges support from the U.S. National Science Foundation (NSF), Award No. CHE-1800596 and the German Science Foundation (DFG) as part of the Excellence Initiative under RESOLVDFG-EXC-2033 Project No. 390677874.
- The acknowledges the received Startup Funds by Syracuse University.

Python Module Index

Index